16 Der strukturierte Datentyp "record"

Gliederung

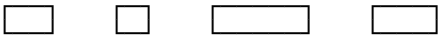
16.1 Kurze Wiederholung zu "Datentypen"

A) Unstrukturierte Datentypen

A.1 Ordinal-Typen: Integer, ShortInt, Byte, LongInt, Char, Boolean, Teilbereichstypen und Aufzählungstypen.

A.2 Real-Typen: Real, Single, Double, Extended und Comp. Mit Ausnahme von "Real" wird bei den anderen Real-Typen ein mathematischer Coprozessor vorausgesetzt oder dessen softwaremäßige Emulation.

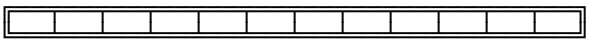
Symbolische Darstellung von unstrukturierten Typen:



B) Strukturierte Datentypen

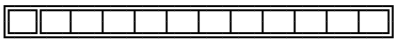
B.1 **array:** Zusammenfassung von Objekten gleichen Typs, z.B. "**array of** Integer", "**array of** Real", "**array of** Boolean" usw.

Symbolische Darstellung eines Arrays:



B.2 **string**: Spezielle Zusammenfassung von Char, "array of Char")

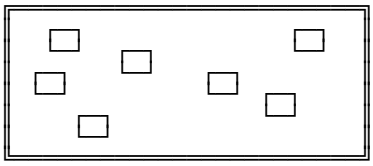
Symbolische Darstellung eines Strings:



Das erste Byte des Strings (hat die Zählnummer Null) enthält das Längenbyte der aktuellen Stringlänge.

B.3 **set**: Zusammenfassung von Ordinaltypen ohne Rangfolge = Menge, wobei die Ordinalwerte zwischen 0 und 255 liegen müssen, z.B. "**set of** Char", "**set of** Byte", "**set of** Boolean", usw.

Symbolische Darstellung einer Menge:



16.2 Allgemeines zum strukturierten Datentyp "record"

Ein "**record**" ist eine Zusammenfassung von Objekten, die im Gegensatz zu "array", "string" und "set" *verschiedene* Datentypen enthalten können, aber dennoch mit einem gemeinsamen (Haupt-) Namen, dem Record-Bezeichner, angesprochen werden.

Symbolische Darstellung des Records:

In der kommerziellen Datenverarbeitung bezeichnet man einen "record" mit "Datensatz" oder kurz "Satz". Die Eindeutschung von "record" wird oft aber auch mit "Verbund" vorgenommen.

Die einzelnen Teile eines Records nennt man Felder oder Komponenten. Jedes Feld hat einen eigenen (Zusatz-) Bezeichner.

Der Zugriff auf ein Feld eines Records erfolgt durch Angabe des Record-Bezeichners und des Feld-Bezeichners. Beide Bezeichner sind mit einem Trennpunkt voneinander zu trennen. Beispiel:

Der Record-Bezeichner: `Student1`
Der Feld-Bezeichner: `Note`
Der Zugriff somit: `Student1.Note`
Der Trennpunkt ——┘

Die noch zu erklärende "with"-Anweisung dient zur Vereinfachung des Zugriffs.

Die Feld-Bezeichner ist ein selbständiger Bezeichner und kollidiert somit bei richtiger Verwendung nicht mit anderen Bezeichnern des Programms. Dennoch sollte man Namensgleichheit vermeiden.

Zur Veranschaulichung: Ein Record mit dem Namen "`Student1`":

Huber	Anton	2000	2.13

Nachname	Vorname	Jahr	Note
1234567890123	1234567890	2 B.	6 Byte

Das erste Feld soll mit "Nachname" bezeichnet werden. Es enthält einen String mit max. 13 Zeichen. Der Bezeichner des zweiten Feldes soll "Vorname" lauten; dieses Feld enthält ebenfalls einen String. Das dritte Feld enthält das Abschlußjahr, das kurz "Jahr" genannt werden soll. Der Datentyp wird zweckmäßigerweise als Integer-Teilbereich (2 Byte) vereinbart. Das vierte Feld soll die Abschlußnote enthalten und mit "Note" bezeichnet werden. Dafür sei der Datentyp Real vorgesehen (6 Byte).

Die Zugriffsmöglichkeiten bei einer Zuweisung:

```
Student1.Nachname := 'Huber';
Student1.Vorname  := 'Anton';
Student1.Jahr     := 2000;
Student1.Note     := 2.13;
```

└—— Der Trennpunkt

Die Reihenfolge des Zugriffs ist beliebig.

Ein Record kann also Felder mit praktisch beliebige Datentypen enthalten, so z.B. auch "array" oder andere "record" und diese wiederum usw. Somit sind sehr komplexe Datenstrukturen möglich.

Trotz dieser internen Datentypen-Vielfalt wird ein Record von außen als *ein* Datentyp betrachtet. Somit kann z.B. auch ein Array mit Records gebildet werden.

Zur Erinnerung: Ein Array kann nur Objekte "gleichen" Datentyps enthalten.

Allerdings sind mit Records nicht möglich: Vergleiche (auch nicht mit typgleichen Records), Ein- und Ausgaben mit kompletten Records. Diese Operationen sind nur feldweise möglich. Somit sind folgende Anweisungen **nicht zulässig**:

```
~~  if Student1 = Student2~~
~~    then ....~~
~~  WriteLn(Student1);~~
```

Zuweisungen von kompletten Records an typgleiche ist jedoch möglich. Damit erspart man sich die aufwendigere feldweise Zuweisung. Beispiel:

```
Student2 := Student1;
```

Bei der Bearbeitung eines Records mit vielen Feldern erspart die with-Anweisung viel Tipparbeit, wie folgendes Beispiel im Vergleich mit der früheren Sequenz andeutet:

```
  with Student1 do
    begin
      Nachname := 'Huber';
      Vorname  := 'Anton';
      Jahr     := 2000;
      Note     := 2.13;
    end;
```

Das folgende Programm demonstriert die Behandlung von Records in der bislang erklärten Weise:

```
program Pas16021;   { Turbo-Pascal, Kap. 16: Datentyp "record" }
                    { Demo: Konstanter Record }
uses
  CRT;
const
  Studentenzahl = 3;
type
  TStudent = record         { Typ-Bezeichner "TStudent" des Records }
    Nachname: string[13];   { Bezeichner Feld 1    }
    Vorname:  string[10];   { Bezeichner Feld 2    }
    Jahr:     1970..2005;   { Bezeichner Feld 3    }
    Note:     Real;         { Bezeichner Feld 4    }
  end;         { Die Record-Deklaration muß mit "end" beendet werden }
var
  Student1,           { Die beiden Variablen sind vom Typ "TStudent" }
  Student2: TStudent; { und nach obiger Typ-Vereinbarung Record-     }
                      { Variablen                                    }
  Semester: array[1..Studentenzahl] of TStudent;
                      { Ein Array von Records !!! }
  i:        Byte;
```

```
procedure WriteXY(Sp, Ze: Byte; Meldung: string);
begin
  GotoXY(Sp, Ze);
  Write(Meldung);
end;


procedure Warte_auf_Tastendruck;
var
  Spalte, Zeile: Byte;
  Ch:            Char;
begin
  Spalte := WhereX;
  Zeile  := WhereY + 1;
  while Keypressed do      { Für absolute Sicherheit   }
    Ch := ReadKey;         { den Tastaturpuffer leeren }
  TextColor(Yellow);
  WriteXY(30, 25, 'Drücke beliebige Taste: ');
  TextColor(White);
  Ch := ReadKey;
  GotoXY(Spalte, Zeile);
end;

begin
  TextBackground(Blue); TextColor(White); ClrScr;

       {  +-- der Trennpunkt !!! }
  Student1.Nachname := 'Huber'; { Man beachte den Punkt zwischen  }
  Student1.Vorname  := 'Anton'; { dem Recordbezeichner "Student1" }
  Student1.Jahr     := 2000;    { und den Feldbezeichnern         }
  Student1.Note     := 2.13;

  WriteLn('--- 1. --------------');
  WriteLn(Student1.Nachname);   { ¦Huber       }
  WriteLn(Student1.Vorname);    { ¦Anton       }
  WriteLn(Student1.Jahr);       { ¦2000        }
  WriteLn(Student1.Note:4:2);   { ¦2.13        }

  Warte_auf_Tastendruck;

  Student2 := Student1; { Zuweisung eines Records an einen anderen. }
                        {  "Student2" hat offensichtlich durch      }
                        {  Abschreiben oder Kopieren Unterschleif   }
                        {  begangen und wird später disqualifiziert }

  Writeln('--- 2. --------------');
  WriteLn(Student2.Nachname);   { ¦Huber       }
  WriteLn(Student2.Vorname);    { ¦Anton       }
  WriteLn(Student2.Jahr);       { ¦2000        }
  WriteLn(Student2.Note:4:2);   { ¦2.13        }

  Warte_auf_Tastendruck;

  { Die "with"- Anweisung dient nur zur Vereinfachung: }

  with Student2 do           { hier Record-Bezeichner mit "with"  }
    begin
```

```
    Nachname := 'Meier';   { hier Zuweisung an Feld-Bezeichner  }
    Vorname  := 'Josef';
    Jahr     := 1992;
    Note     := 6.00;      { Die unbotmäßige Rache des Dozenten }
  end;

  Writeln('--- 3. --------------');
  with Student2 do
    begin
      WriteLn(Nachname);     { ¦Meier       }
      WriteLn(Vorname);      { ¦Josef       }
      WriteLn(Jahr);         { ¦2000        }
      WriteLn(Note:4:2);     { ¦6.00        }
    end;

  Warte_auf_Tastendruck;

  with Student2 do          { Kleiner Amtsirrtum:                   }
    begin                   { Der Abkupferer heißt in Wirklichkeit  }
      Nachname := 'Mayer';  { "Mayer" und nicht "Meier".            }
      Note     := 5.00;     { Die Note muß natürlich "5.00" lauten. }
    end;

  WriteLn('--- 4. --------------');
  WriteLn(Student2.Nachname);   { ¦Mayer       }
  WriteLn(Student2.Vorname);    { ¦Josef       }
  WriteLn(Student2.Jahr);       { ¦2000        }
  WriteLn(Student2.Note:4:2);   { ¦5.00        }
  { Diese Sequenz ist schöner mit "with" zu programmieren }

  Warte_auf_Tastendruck;
  { Ein Array mit Records ... }
  for i := 1 to Studentenzahl do  { ... da kommt Freude auf ....    }
    with Semester[i] do           { ... aber leider auch nur kurz   }
      begin                       { Das ganze Semester hat die      }
        Nachname := 'Help';       { Studienarbeit von Frau J. Help  }
        Vorname  := 'Julia';      { kopiert. Für die Beherrschung   }
        Jahr     := 2000;         { des MS-DOS-Befehls "copy" gibt  }
        Note     := 1.14;         { in Pascal keinen Punkt!         }
      end;

  WriteXY(40, 1, '--- 5. --------------');
  for i := 1 to Studentenzahl do
    with Semester[i] do
       begin
         WriteXY(40, 1 + i, Nachname + ' ' + Vorname + ' ');
         WriteLn(Jahr, Note:6:2);
       end;
  { Die Bildschirmausgabe:      }
  { ¦Help Julia 2000  1.14      }
  { ¦Help Julia 2000  1.14      }
  { ¦Help Julia 2000  1.14      }

  WriteXY(40, i + 4, '--- Ende ------------');
  Warte_auf_Tastendruck;
end.
```

16.3 Zum varianten Record

Beim bisherigen Beispiel lag eine feste Feldaufteilung des Records vor. Um Speicherplatz zu sparen, läßt es Pascal aber zu, daß komplementäre Felder ein und desselben Records einen gemeinsamen Speicherplatz benutzen. Derartige Records nennt man "variante" Records".

Komplementär bedeutet in diesem Zusammenhang, daß im varianten Record der "variante" Teil entweder mit X belegt ist *oder* mit Y oder mit Z usw.; nie aber mit X, Y, Z usw. gleichzeitig. Die Datentypen von X, Y, Z usw. können verschieden sein. Der "variante" Teil kann aus einem oder mehreren Feldern bestehen.

Im allgemeinen wird ein derartiger Record aber auch einen "konstanten" Teil besitzen. In diesem Fall ist der konstante Teil *vor* dem varianten Teil zu deklarieren.

Der variante Teil beginnt mit dem reservierten Wort "case". Die Feldbezeichner des varianten Teils und die zugehörigen Datentypen müssen in runde Klammern gesetzt werden. Besteht der variante Teil aus mehreren Feldern, dann ist das Semikolon als Trennzeichen zu benutzen. Weitere Details siehe folgendens Programm:

```
program Pas16031;  { Demo "varianter" Record }

uses
  CRT;

type
  Geschlecht = (Maennlich, Weiblich);
  Reize      = (Manchmal_reizend, Ziemlich_reizend, Sehr_reizend);
  Haarfarben = (Blond, Lila, Rot, Schwarz);
  Autos      = (Keines, Trabi, Golf, BMW_720,
                Mercedes_450_SE, Porsche_911);
  Personendaten = record
    Sex:      Geschlecht;   { Konstantes Feld 1, Bezeichner: "Sex"   }
    Name:     string[15];   { Konstantes Feld 2, Bezeichner: "Name"  }
    Alter:    20..80;       { Konstantes Feld 3, Bezeichner: "Alter" }
    case Geschlecht of
      Maennlich: (Vermoegen: Real;            { Runde Klammern für   }
                  Auto:      Autos;           { die varianten Felder }
                  Groesse:   150..220);
      Weiblich:  (Reiz:      Reize;           { Hier ebenso          }
                  Haarfarbe: Haarfarben);     { runde Klammern       }
  end; { von "record" }

  { Der vorliegende variante Record "Personendaten" besitzt einen
    konstanten Teil mit drei Feldern. Der variante Teil besteht im
    Fall "Weiblich" aus  zwei Feldern mit den Bezeichnern "Reiz" und
    "Haarfarbe" und im Fall "Maennlich" aus drei Feldern mit den
    Bezeichnern "Vermoegen", "Auto" und "Groesse". }

var
  Kandidat: array[1..7] of Personendaten;   { Array von Records !!!! }
  i:        Byte;

procedure WriteXY(Spalte, Zeile: Byte; Meldung: string);
begin
  GotoXY(Spalte, Zeile);
```

```
  Write(Meldung);
end;

procedure Warte_auf_Tastendruck;
var
  Spalte, Zeile: Byte;
  Ch:            Char;
begin
  Spalte := WhereX;
  Zeile  := WhereY + 1;
  while Keypressed do      { Für absolute Sicherheit   }
    Ch := ReadKey;         { den Tastaturpuffer leeren }
  WriteXY(10, 25, 'Drücke beliebige Taste: ');
  Ch := ReadKey;
  GotoXY(Spalte, Zeile);
end;

begin { ------------------------------------------------------------ }
  TextBackground(Blue); TextColor(White); ClrScr;

  Kandidat[1].Sex       := Weiblich;          { konstantes Feld 1 }
  Kandidat[1].Name      := 'Julia Help';      { konstantes Feld 2 }
  Kandidat[1].Alter     := 23;                { konstantes Feld 3 }
  Kandidat[1].Reiz      := Sehr_reizend;      { variantes  Feld 1 }
  Kandidat[1].Haarfarbe := Blond;             { variantes  Feld 2 }

  Kandidat[2].Sex       := Maennlich;         { konstantes Feld 1 }
  Kandidat[2].Name      := 'Anton Huber';     { konstantes Feld 2 }
  Kandidat[2].Alter     := 25;                { konstantes Feld 3 }
  Kandidat[2].Vermoegen := -47.11;            { variantes  Feld 1 }
  Kandidat[2].Auto      := Keines;            { variantes  Feld 2 }
  Kandidat[2].Groesse   := 178;               { variantes  Feld 3 }

  Kandidat[3].Sex       := Weiblich;
  Kandidat[3].Name      := 'Helga Punk';
  Kandidat[3].Alter     := 21;
  Kandidat[3].Reiz      := Ziemlich_reizend;
  Kandidat[3].Haarfarbe := Lila;

  Kandidat[4].Sex       := Weiblich;
  Kandidat[4].Name      := 'Paula Putz';
  Kandidat[4].Alter     := 48;
  Kandidat[4].Reiz      := Manchmal_reizend;
  Kandidat[4].Haarfarbe := Schwarz;

  Kandidat[5].Sex       := Maennlich;
  Kandidat[5].Name      := 'Waldemar Bonze';
  Kandidat[5].Alter     := 68;
  Kandidat[5].Vermoegen := 4.711E+7;
  Kandidat[5].Auto      := Mercedes_450_SE;
  Kandidat[5].Groesse   := 172;

  with Kandidat[6] do   { Zur Abwechslung mal wieder mit "with": }
    begin
      Sex       := Maennlich;
      Name      := 'Klaus Yuppie';
      Alter     := 32;
      Vermoegen := 1.0E+6;
      Auto      := Porsche_911;
      Groesse   := 182;
    end;
```

```
  Kandidat[7].Sex       := Weiblich;
  Kandidat[7].Name      := 'Claudia Meier';
  Kandidat[7].Alter     := 20;
  Kandidat[7].Reiz      := Sehr_reizend;
  Kandidat[7].Haarfarbe := Rot;

  WriteXY(10, 7, '------- Demo: Varianter Record ---------');
  WriteLn(#13#10);

  for i := 1 to 7 do
    begin
      if (Kandidat[i].Sex       =  Weiblich)      and  { Anspruchs- }
         (Kandidat[i].Reiz      =  Sehr_reizend)  and  { voller     }
         (Kandidat[i].Alter     <  30)            and  { Wunsch     }
         (Kandidat[i].Haarfarbe =  Blond)         then { des Herrn  }
         begin
           WriteXY(10, WhereY, 'Traumfrau: ');
           WriteLn(Kandidat[i].Name:16);
         end;

      if (Kandidat[i].Sex       =  Maennlich)     and
         (Kandidat[i].Vermoegen >= 1.0E+6)        then { Bescheiden }
         begin                                         { der Wunsch }
           WriteXY(10, WhereY, 'Supermann: ');         { der Dame   }
           WriteLn( Kandidat[i].Name:16, '  DM ',
                    Kandidat[i].Vermoegen:8:0);
         end;
    end;
  { Die Bildschirmausgabe bei obigen Daten:        }
  { ¦... Traumfrau:       Julia Help               }
  { ¦... Supermann:   Waldemar Bonze  DM 47110000  }
  { ¦... Supermann:     Klaus Yuppie  DM  1000000  }

  Warte_auf_Tastendruck;
end.
```

16.4 Übergabe eines Records als Parameter

Für die Übergabe eines Records als formalen Parameter in einer Funktion oder Prozedur ist unbedingt vorher mit "type" ein eigener Record-Typ zu deklarieren. Das folgende Programm zeigt die Vorgehensweise:

```
program Pas16041; { Records. Im Gegensatz zu "Pas16021.PAS" hier     }
        { zusätzlich auch direkte Deklaration eines Records in der   }
uses    { Variablendeklaration. Besser ist es mit "type" einen       }
  CRT;  { eigenen Record-Datentyp zu vereinbaren. Bei der Übergabe   }
        { eines Records als Parameter einer Routine ist ein eigener  }
        { Typbezeichner unbedingt notwendig! Strukturierte Typen     }
        { können nur so übergeben werden.                            }
type
  TStudent = record                { Indirekte Deklaration, eigener }
               Nachname: string[20]; { Datentyp. Die spätere Record-  }
               Vorname:  string[20]; { variable "Student2" bekommt    }
               Jahr:     1970..2005; { diesen Typ und kann als Para-  }
               Note:     Real;       { meter an eine Routine über-    }
             end;                    { geben werden. }
```

```
var
  Student1: record                    { Direkte Deklaration. Kein     }
              Nachname:  string[20]; { eigener Datentyp. Diese       }
              Vorname:   string[20]; { Recordvariable kann n i c h t }
              Jahr:      1970..2005; { als Parameter an eine Routine }
              Note:      Real;       { übergeben werden. }
            end;
  Student2: TStudent;

procedure Demo(Student3: TStudent);
begin
  WriteLn(Student3.Nachname);
  WriteLn(Student3.Vorname);
  WriteLn(Student3.Jahr);
  WriteLn(Student3.Note:4:2);
end;

begin
  ClrScr;
  Student1.Nachname := 'Huber';
  Student1.Vorname  := 'Anton';
  Student1.Jahr     := 2000;
  Student1.Note     := 2.13;

  WriteLn(Student1.Nachname);
  WriteLn(Student1.Vorname);
  WriteLn(Student1.Jahr);
  WriteLn(Student1.Note:4:2);
  WriteLn;

  Student2.Nachname := 'Help';
  Student2.Vorname  := 'Julia';
  Student2.Jahr     := 2000;
  Student2.Note     := 1.47;

  WriteLn(Student2.Nachname);
  WriteLn(Student2.Vorname);
  WriteLn(Student2.Jahr);
  WriteLn(Student2.Note:4:2);
  WriteLn;

  Demo(Student2);  { Nicht möglich: "Demo(Student1)", sonst      }
                   { Meldung "Typen nicht miteinander vereinbar" }
      { Die Ausgabe: 1 * Daten von "Anton Huber" }
      {              2 * Daten von "Julia Help"  }

  repeat
  until KeyPressed;
end.
```

16. 5 Record mit weiterem Record und Array

```
program Pas16051; { Ein Record mit Unterrecord,    }
                  { der u.a. einen "array" enthält }
uses
  CRT;

const
  n     = 2;
```

```
  kMax  = 6;
  Dummy = 'Dummy';
type
  RekordTypB = record
                 Kind:   array[1..kMax] of string[20];
                 Wunsch: (Pferd, Reise, Fernseher);
               end;
  RekordTypA = record
                 Vater,
                 Mutter: string[30];
                 Kinder: RekordTypB;
               end;
var
  Familien: array[1..2] of RekordTypA;
  i, k:     Byte;
begin
  ClrScr;

  Familien[1].Vater          := 'Anton Huber';
  Familien[1].Mutter         := 'Anna Huber';
  Familien[1].Kinder.Kind[1] := 'Max';
  Familien[1].Kinder.Kind[2] := 'Moritz';
  Familien[1].Kinder.Kind[3] := 'Julia';
  Familien[1].Kinder.Kind[4] := Dummy;
  Familien[1].Kinder.Wunsch  := Pferd;

  Familien[2].Vater          := 'Josef Meier';
  Familien[2].Mutter         := 'Maria Meier';
  Familien[2].Kinder.Kind[1] := 'Kurt';
  Familien[2].Kinder.Kind[2] := 'Helga';
  Familien[2].Kinder.Kind[3] := Dummy;
  Familien[2].Kinder.Wunsch  := Fernseher;

  for i := 1 to n do
    with Familien[i] do
      begin
        WriteLn('Vater:  ', Vater);
        WriteLn('Mutter: ', Mutter);
        if Kinder.Kind[1] <> Dummy then
           begin
             Write('Wunsch der Kinder: ');
             case Kinder.Wunsch of
               Pferd:     WriteLn('Ein Pferd');
               Fernseher: WriteLn('Einen Fernseher');
               Reise:     WriteLn('Eine Reise');
             end;
             for k := 1 to kMax do
               if Kinder.Kind[k] <> Dummy
                  then WriteLn('Kind ', k, ': ', Kinder.Kind[k])
                  else Break;  { k-Schleife vorzeitig beenden }
           end;
        WriteLn;
      end;

  { Die Bildschirmausgabe:              }
  { ¦Vater:  Anton Huber                }
  { ¦Mutter: Anna Huber                 }
```

```
  { ¦Wunsch der Kinder: Ein Pferd       }
  { ¦Kind 1: Max                        }
  { ¦Kind 2: Moritz                     }
  { ¦Kind 3: Julia                      }
  { ¦                                   }
  { ¦Vater:  Josef Meier                }
  { ¦Mutter: Maria Meier                }
  { ¦Wunsch der Kinder: Einen Fernseher }
  { ¦Kind 1: Kurt                       }
  { ¦Kind 2: Helga                      }
  repeat
  until KeyPressed;
end.
```