15 Der strukturierte Datentyp »set« (Mengen)

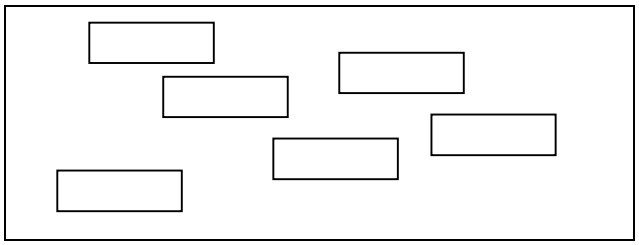
Gliederung

15.1 Allgemeines zu Mengen

Ein "set" ist eine Zusammenfassung von Elementen des gleichen Grundtyps, im Gegensatz zu Arrays aber ohne Rangordnung. Somit kann ein einzelnes Mengenelement auch nicht über einen Index angesprochen werden. Die Reihenfolge der Elemente in der Menge ist nicht definiert.

Ein Array:

Eine Menge:



Der Grundtyp der Menge muß ein Ordinaltyp (Integer, Char, Boolean, Aufzählungstyp oder Teilbereichstyp) sein. Die Deklaration einer Mengenvariablen lautet somit:

```
var
  Mengenvariable: set of Ordinaltyp;
```

Es empfiehlt sich, auch für Mengen mit "type" eigene Datentypen zu deklarieren.

Bei Turbo-Pascal liegt (wie bei vielen Pascal-Implementationen) aus technischen Gründen eine Einschränkung der zulässigen Ordinaltypen für Mengen vor: **Der Ordinalwert muß zwischen 0 und 255 liegen**. Somit ist "set of Integer" oder "set of ShortInt" oder "set of Word" in Turbo-Pascal nicht möglich, wohl aber "set of Byte" oder "set of Char".

Mengenvariablen können nicht direkt mit "Read" eingelesen oder "Write" ausgegeben werden. Für Zuweisungen dient der üblichen Zuweisungsoperator ":=".

Zum Aufbau einer Menge aus Elementen dient als Mengenbildner (engl. constructor) das eckige Klammernpaar. Eine Menge, die keine Elemente enthält, ist eine leere Menge. Sie wird durch die Zeichenfolge "[]" ausgedrückt.

Beispiel für Deklaration und Mengenbildung:

```
var
  M1,
  M2: set of Char;
  M3: set of Byte;
begin
  M1 := ['a', 'e', 'i', 'o', 'u'];
  M2 := ['a'..'z', 'A'..'Z', '0'..'9'];
  M3 := [2, 5, 9, 11..47];
  M1 := [];       { Leere Menge }
  M3 := [47..11]; { Ist auch eine leere Menge, da der untere Ordi-  }
       { nalwert des Bereiches größer ist als der obere Ordinalwert }
```

15.2 Mengen-Operatoren

Es stehen verschiedene Mengen-Operatoren zur Verfügung, mit denen z.B. geprüft werden kann, ob ein Element in einer Menge enthalten ist, oder mit denen zwei Mengen miteinander verknüpft werden können, um z.B. die Schnittmenge zu bestimmen.

Für die folgende Darstellung aller Mengenoperatoren seien *M1* und *M2* Mengen des gleichen Grundtyps und *E* sei ein Element:

Op	Beispiel	Ergebnis-Datentyp	Erklärungen
in	*E* **in** *M1*	`Boolean`	Operator "**in**" für Prüfung, ob Element *E* in der Menge *M1* enthalten ist.
=	*M1* **=** *M2*	`Boolean`	Operator "=" für Prüfung auf Gleichheit der Mengen *M1* und *M2*.
<>	*M1* **<>** *M2*	`Boolean`	Operator "<>" für Prüfung auf Ungleichheit.
<=	*M1* **<=** *M2*	`Boolean`	Operator "<=" für Prüfung auf Untermenge. Das Ergebnis ist dann True, wenn alle Elemente von *M1* in *M2* enthalten sind.
>=	*M1* **>=** *M2*	`Boolean`	Operator ">=" für Prüfung auf Obermenge. Das Ergebnis ist dann True, wenn alle Elemente von *M2* in *M1* enthalten sind.
+	*M1* **+** *M2*	**set**	Operator "+" für Bildung der Vereinigungsmenge. Wirkung ähnlich wie logisches "or".
-	*M1* **-** *M2*	**set**	Operator "-" für Bildung der Differenzmenge.
*****	*M1* ***** *M2*	**set**	Operator "*" für Bildung der Schnittmenge. Wirkung ähnlich wie logisches "and". Die Mengendisjunktion (kein gemeinsames Element) kann getestet werden, wenn man die Schnittmenge auf Gleichheit mit der leeren Menge prüft: `if (M1 * M2 = []) then ...` Ist die Schnittmenge leer, d.h. ergibt der Vergleich True, dann sind M1 und M2 disjunkt, haben also kein gemeinsames Element. Die Disjunktion ist nicht zu verwechseln mit der Ungleichheit, die dann schon vorliegt, wenn sich die beiden Mengen auch nur bei einem Element unterscheiden.

Das folgende Demo-Programm demonstriert die Behandlung von Mengen:

```
{$R- Bereichsüberprüfung (Range) ausschalten   }
{    Für diese Demo ausnahmsweise zweckmäßiger }
program Pas15021;  { Strukturierter Datentyp "set" }

uses
  CRT;

const
  Farbe1 = White;
  Farbe2 = LightGreen;

type
```

```
  MengentypKleinbuchstaben = set of 'a'..'z';
                    { Teilbereich von "Char" }
  MengentypFarben = set of (Cyan, Magenta, Gelb, Black, Lila, Pink);
                           { Aufzählungstyp }
var
  Buchstaben,
  Vokale,
  Konsonanten: MengentypKleinbuchstaben;
  Farben:      MengentypFarben;

  GeradeZiffern,
  UngeradeZiffern,
  Ziffern:         set of 0..9; { Direkte Deklaration einer Menge }
                { "set", hier Teilbereich von "Byte" }

  i, x, Ze:        Byte;
  Zeichen:         Char;


function ZiffernzeichenZuInteger(Ch: Char): Byte;
begin
  ZiffernzeichenZuInteger := Ord(Ch) - Ord('0');
end;

begin
  TextBackground(Blue); TextColor(Farbe1); ClrScr;

  WriteLn('------ Demo Datentyp "set" -------', #13#10);

  { Mengen bilden mit "[ ]" und Mengenvariablen initialisieren:   }
  Farben := [Lila, Pink]; { Keine weitere Behandlung dieser Menge }
                          { mit Aufzählungstyp                    }

  Ziffern := [1..3];      { oder:  Ziffern := [1, 2, 3]; }


  { Mengenoperator "in", Byte-Menge: }
  TextColor(Farbe1);
  Ze := WhereY;
  repeat
    GotoXY(1, Ze); ClrEol; GotoXY(1, Ze);
    Write('Eingabe Ziffer 1 bis 3:          ');
    x := ZiffernzeichenZuInteger(ReadKey);
    WriteLn(x);
  until x in Ziffern;      { Mengenoperator "in" }

  { oder: }
  TextColor(Farbe2);
  Ze := WhereY;
  repeat
    GotoXY(1, Ze); ClrEol; GotoXY(1, Ze);
    Write('Oder so: Eingabe Ziffer 1 bis 3: ');
    x := ZiffernzeichenZuInteger(ReadKey);
    WriteLn(x);
  until x in [1..3];

  TextColor(Farbe1);
```

```
Ze := WhereY;
Ziffern := [2..5, 9, 7];     { Mengenvariable anders belegen  }
             { Die alten Werte sind, wie bei einer "normalen" }
             { Variablen auch, nicht mehr existent }
repeat
  GotoXY(1, Ze); ClrEol; GotoXY(1, Ze);
  Write('Eingabe Ziffer 2..5, 7, 9:       ');
  x := ZiffernzeichenZuInteger(ReadKey);
  WriteLn(x);
until x in Ziffern;

{ Mengenoperator "in", Char-Menge: }
TextColor(Farbe2);
Ze := WhereY;
Buchstaben := ['h', 'i', 'l', 'f', 'e'];   { Initialisierung }
repeat
  GotoXY(1, Ze); ClrEol; GotoXY(1, Ze);
         { Großbuchstaben werden in diesem Beispiel nicht    }
         { akzeptiert, da "Buchstaben" in der Variablen-     }
         { deklaration als "set of 'a'..'z'" aufgeführt ist  }
  Write('Eingabe ein Zeichen aus "hilfe": ');
  Zeichen := ReadKey;
  WriteLn(Zeichen);
until Zeichen in Buchstaben;

{ Differenzmenge mit Operator "-": }
Buchstaben  := ['a'..'z'];
Vokale      := ['a', 'e', 'i', 'o', 'u'];

Konsonanten := Buchstaben - Vokale; { Differenzmengenoperator "-" }

{ Mengenelement hinzufügen (Vereinigungsmenge)              }
{ oder entfernen (Differenzmenge):                          }
Konsonanten := Konsonanten + ['a'];  { Logisch unsinnig     }
Konsonanten := Konsonanten - ['a'];  { Wieder in Ordnung    }
{ oder mit "Include" bzw. "Exclude" (siehe auch Kap. 15.3): }
Include(Konsonanten, 'a'); { Logisch unsinnig  }
Exclude(Konsonanten, 'a'); { Wieder in Ordnung }

TextColor(Farbe1);
Ze := WhereY;
repeat
  GotoXY(1, Ze); ClrEol; GotoXY(1, Ze);
  Write('Geben Sie einen Konsonanten ein: ');
  Zeichen := ReadKey;
  WriteLn(Zeichen);
until Zeichen in Konsonanten;

{ Vereinigungsmenge mit Operator "+": }
GeradeZiffern   := [0, 2, 4, 6, 8];
UngeradeZiffern := [1, 3, 5, 7, 9];
Ziffern         := GeradeZiffern + UngeradeZiffern;

{ Nochmal: Mengenoperator "in": }
x := 2;
TextColor(Farbe2);
WriteLn(x, ' in geraden Ziffern:   ', x in GeradeZiffern   );
```

```
                                                    { ¦... TRUE  }
  WriteLn(x, ' in ungeraden Ziffern: ', x in UngeradeZiffern );
                                                    { ¦... FALSE }
  { Vergleichsoperatoren für Mengen: }
  TextColor(Farbe1);
  WriteLn;
  WriteLn('=  ', GeradeZiffern =  [0, 2, 4, 6, 8]  ); { ¦=  TRUE   }
  WriteLn('<> ', GeradeZiffern <> [0, 2, 4, 6, 8]  ); { ¦<> FALSE  }
  WriteLn('<= ', GeradeZiffern <= Ziffern          ); { ¦<= TRUE   }
  WriteLn('>= ', GeradeZiffern >= Ziffern          ); { ¦>= FALSE  }

  { Nochmals Vereinigungs-/Differenzmenge und Vergleiche: }
  TextColor(Farbe2);
  WriteLn('+  ', (GeradeZiffern + UngeradeZiffern) = Ziffern);
                                                    { ¦+  TRUE   }
  WriteLn('-  ', (Ziffern - GeradeZiffern) = UngeradeZiffern);
                                                    { ¦-  TRUE   }
  { Test ob beide Mengen disjunkt: }
  WriteLn('*  ', (GeradeZiffern * UngeradeZiffern) = []);
                                                    { ¦*  TRUE   }
  WriteLn;

  { Mengenelemente abfragen: }
  TextColor(Farbe1);
  Ziffern := [];   { leere Menge }
  for i := 1 to 4 do
    begin
      Write(i, ': Geben Sie ein Ziffernzeichen ein: ');
      x := ZiffernzeichenZuInteger(ReadKey);
      WriteLn(x);
      Ziffern := Ziffern + [x];
      { Vereinigungsmenge, Operator "+" }
    end;

  for i := 0 to 9 do
    begin
      Write(i, ':');
      if i in Ziffern
         then begin
                TextColor(Yellow);
                Write('J  ');    { "J" für "ja"   }
                TextColor(Farbe1);
              end
         else Write('n  ');      { "n" für "nein" }
    end;

  { Wenn vorher eingegeben wird: "5", "3", "5" und "7", }
  { dann erhält man folgende Ausgabe:                    }
  { ¦0:n  1:n  2:n  3:J  4:n  5:J  6:n  7:J  8:n  9:n    }

  TextColor(Farbe2);
  Write(#13#10, 'Ende mit Taste "Esc" ... ');
  repeat
  until ReadKey = #27;
end.
```

15.3 Mengen-Prozeduren

Ab Turbo-Pascal 7.0 stehen die Mengen-Prozeduren

```
Exclude(M, E)
Include(M, E)
```

zur Verfügung. Sie entfernen ein Element *E* aus der Menge *M* bzw. fügen ein Element *E* der Menge *M* hinzu. *M* muß eine Mengenvariable sein. Beide Prozeduren werden im folgenden Programm demonstriert. Das Entfernen kann aber auch mit dem Mengenoperator "-", das Hinzufügen auch mit dem Mengenoperator "+" bewerkstelligt werden.

```
program Pas15031;   { Ab Turbo-Pascal 7.0 stehen die Mengen-
  Prozeduren "Exclude(M, E)" und "Include(M, E)" zur Verfügung.
  Sie entfernen ein Element E aus der Menge M bzw. fügen ein
  Element E der Menge M hinzu. M muß eine Mengenvariable sein.
  Die gleichen Aufgaben können auch mit den Mengenoperatoren
  "-" und "+" gelöst werden. }

uses
  CRT;

var
  M:       set of 'A'..'H';
  Zeichen: Char;

procedure TestenUndSchreiben;
var
  ElementVorhanden: Boolean;
begin
  ElementVorhanden := Zeichen in M;
  Write(Zeichen, '  ');
  if ElementVorhanden
     then TextColor(White)
     else TextColor(Yellow);
  WriteLn(ElementVorhanden);
  TextColor(White);
end;


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

  M := ['A'..'H'];

  GotoXY(5, 5);
  WriteLn('Menge M vorher'); WriteLn;
  for Zeichen := 'A' to 'M' do
    begin
      GotoXY(5, WhereY);
      TestenUndSchreiben;
    end;
```

```
  GotoXY(30, 5);
  WriteLn('Demo "Exclude(M, E)"'); WriteLn;
  { Entfernen mit Mengenprozedur "Exclude": }
  Exclude(M, 'E');
  { Entfernen mit Mengenoperator "-":       }
  M := M - ['F'];
  for Zeichen := 'A' to 'M' do
    begin
      GotoXY(30, WhereY);
      TestenUndSchreiben;
    end;

  GotoXY(55, 5);
  WriteLn('Demo "Include(M, E)"'); WriteLn;
  { Einfügen mit Mengenprozedur "Include": }
  Include(M, 'E');
  { Einfügen mit Mengenoperator "+":       }
  M := M + ['F'];
  for Zeichen := 'A' to 'M' do
    begin
      GotoXY(55, WhereY);
      TestenUndSchreiben;
    end;

  repeat
  until KeyPressed;
end.
```