14 Der strukturierte Datentyp »string« (Zeichenketten)

Gliederung

Für die Behandlung von Strings stehen in Turbo-Pascal mehrere Prozeduren und Funktionen zur Verfügung, die der Einfachheit halber in einem Demo-Programm gezeigt werden. Das Listing enthält auch simulierte Bildschirmausgaben.

```
program Pas14000; { Standard-Strings in Turbo-Pascal }

uses
  CRT;

  { 14.1: Deklaration eines Strings
    Beispiel:
    ...;
    var
      Bezeichner: string[50];
    ...;
    Der String kann in diesem Fall max. 50 Zeichen umfassen. Die
    aktuelle Länge des Strings kann zwischen 0 und 50 liegen. Im
    Speicher werden 50 + 1 = 51 Bytes reserviert. Die Zählung
    beginnt bei 0. Das Byte 0 enthält das Längenbyte, das die
    aktuelle String-Länge repräsentiert. Der ASCII-Code (in Pascal
    die Ordnungsnummer Ord() ) des Zeichens im Längenbyte ist die
    aktuelle Stringlänge.
     0                   1                   4                   5
     0 1 2 3 4 5 6 7 8 9 0 1  2      7 8 9 0 1 2 3 4 5 6 7 8 9 0
    ╔═══════════════════════════════( )═══════════════════════════╗
    ╟─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─┬─╢
    ╟─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─╢
    ╟─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─┼─╢
    ╚═══════════════════════════════( )═══════════════════════════╝
     └──── Das Längenbyte, Index 0

    Beispiel: Im Längenbyte steht das Zeichen '/'. Dann ist die
              aktuelle Länge des Strings: Ord('/')    ----> 47.
              Wäre die  aktuelle Länge 42 Zeichen, stünde im
              Längenbyte das Zeichen '*', da Ord('*') ----> 42.
  }
type
  Str24 = string[24];

var
  s1:   string[10]; { Dieser String kann bis 10 Zeichen lang werden  }
                    { Die max. Länge in Turbo-Pascal 255 Zeichen     }
                    { Die max. Stringlänge muß eine Konstante sein   }
  s2:   string[10];
  s3:   Str24;      { Eigener Datentyp }
  s4:   string[40];
  yStr: string;     { max. Länge 255 Zeichen, wenn keine Länge dekl. }

  Ch:   Char;
  i:    Byte;
  j:    Integer;
  y:    Real;
  Fehlercode: Integer;

{ Siehe auch Kap. 8 betreffend Funktionen "Ord" und "Chr" }
```

```
procedure KonventionelleStringUebergabe(s: Str24);
        { N i c h t: ...StringUebergabe(s: string[24]);         }
begin   { Siehe auch Kap. 14.14: Übergabe von "offenen Strings" }
  WriteLn(' Nr 14.14: ', s);
         {¦ Nr 14.14:  4.71099999999860E+0001 }
end;

begin
  ClrScr;
```

{ 14.2: Die Funktion "Concat" (Stringverknüpfung)

```
    Format: Concat(s1, s2, s3, ...)

            s1, s2, s3, ...: Stringausdrücke
    }

    s1 := 'Anton ';
    s2 := 'Huber, ';
    s3 := 'München';

    s4 := Concat(s1, s2, s3);

    WriteLn(' Nr 14.2a:  ', s4);
          { ¦ Nr 14.2a:  Anton Huber, München }
    { oder einfacher ... }

    s4 := s1 + s2 + s3;

    WriteLn(' Nr 14.2b:  ', s4);
          { ¦ Nr 14.2b:  Anton Huber, München }
```

{ 14.3: Die Funktion "Length" (aktuelle Länge des Strings)

```
    Format:  Length(s)

             s:  Stringausdruck

    Ergebnistyp: Integer, Bereich 0..255 oder Typ Byte
    }
    WriteLn(' Nr 14.3:   Die Länge von s1: ', Length(s1));
          { ¦ Nr 14.3:   Die Länge von s1: 6 }
```

{ 14.4: Die Funktion "Copy" (Teile eines Strings kopieren)

```
    Format:  Copy(s, position, anzahl)

             s:          Stringausdruck aus dem ein Teilstring
                         kopiert werden soll.
             position:   Integerausdruck. Ab dieser Position soll
                         kopiert werden.
             anzahl:     Integerausdruck. Anzahl der Zeichen, die
                         in die Kopie übernommen werden sollen.

    Beispiel: +-+
           123456789012  }
    s3 := 'abcxyzhiojkl';
    s2 := Copy(s3, 4, 3);
    WriteLn(' Nr 14.4:   ', s2);
          { ¦ Nr 14.4:   xyz  }
```

{ 14.5: Die Funktion "Pos" (Position eines Strings in einem anderen)

Format: Pos(*suchstring, zu_durchsuchender_string*)

Beide Strings können Ausdrücke sein. Geliefert wird die Position des ersten Auftretens (Datentyp Byte, 0..255). Wenn der Suchstring nicht enthalten ist, liefert "Pos" den Wert 0.

Beispiel:

```
       12345678901234567890 }
s3 := 'Anton Huber, München';
WriteLn(' Nr 14.5:   ', Pos('Huber', s3) );
      { ¦ Nr 14.5:   7 }
```

{ 14.6: Die Prozedur "Delete" (Zeichen im String löschen)

Format: Delete(*s, position, anzahl*)

s: Stringvariable
position: Integerausdruck. Ab dieser Position (inklusiv) wird gelöscht.
anzahl: Integerausdruck. Anzahl der zu löschenden Zeichen.

Wenn "*position*" größer ist als die aktuelle Länge von "*s*", wird nichts gelöscht. Wenn die Summe von "*position*" und "*anzahl*" größer ist als die aktuelle Länge von "*s*", werden alle ab "*position*" (inklusiv) stehenden Zeichen gelöscht.

Beispiel:

```
       12345678901234567890
             12345         }
s3 := 'Anton Huber, München';
Delete(s3, 7, 5);
WriteLn(' Nr 14.6:   ', s3);
      { ¦ Nr 14.6:   Anton , München    }
```

{ 14.7: Die Prozedur "Insert" (Zeichen in den String einfügen)

Format: Insert(*quellstring*, *zielstring*, *position*)

quellstring: Stringausdruck
zielstring: Stringvariable
position: Integerausdruck, aber Bereich 1..255

Der "*quellstring*" wird in den "*zielstring*" ab "*position*" eingefügt werden und zwar ab "*position*". Bereich 1..255, sonst run time error)

Wenn der entstehende Zielstring länger ist als die deklarierte Länge, dann werden die überflüssigen Zeichen abgeschnitten.

Wenn "*position*" länger ist als die Länge des ursprünglichen "*zielstrings*", dann wird der "*quellstring*" an den "*zielstring*" angehängt.

```
Beispiel:
       12345678901234567890 }
s2 := 'Huber, ';          { Quellstring }
s3 := 'Anton München';    { Zielstring  }

Insert(s2, s3, 7);
WriteLn(' Nr 14.7:   ', s3);
      { ¦ Nr 14.7:   Anton Huber, München }
```

{ 14.8: Die Prozedur "Str" (Umwandlung Zahl in Ziffernstring)

```
Format:   Str(n, s)

          n: Numerischer Ausdruck mit optionalen Formatierungs-
             parametern, siehe Beispiele
          s: Stringvariable
}

j := -4711;
Str(j, s1);
WriteLn(' Nr 14.8a:  ', s1);
      { ¦ Nr 14.8a:  -4711           }

Str(4711:8, s1);   { optional mit Formatierung, hier 8 }
WriteLn(' Nr 14.8b:  ', s1);
      { ¦ Nr 14.8b:      4711        }

y := 47.11;
Str(y, s3);
WriteLn(' Nr 14.8c: ', s3);
      { ¦ Nr 14.8c:  4.71099999999860E+0001 }  { Coprozessor! }

Str(y:7:3, s2);  { optional mit Dezimal-Formatierung }
WriteLn(' Nr 14.8d: ', s2);
      { ¦ Nr 14.8d:  47.110          }
```

{ 14.9: Die Prozedur "Val" (Umwandlung Ziffernstring in Zahl)

```
Format:  Val(s, n, f)

         s: Stringausdruck
         n: Numerische Variable
         f: Integervariable für Fehlercode. Bei erfolgreicher
            Konvertierung hat der Fehlercode den Wert 0, sonst
            die Position im String s, bei der die Inter-
            pretation wegen numerisch unzulässiger Zeichen
            abgebrochen werden mußte.
}

s1 := '47.11';
Val(s1, y, Fehlercode);
if Fehlercode = 0
   then WriteLn(' Nr 14.9a: ', y)
             { ¦ Nr 14.9a:  4.71099999999860E+0001 }
                            { Coprozessor! }
   else WriteLn(' Nr 14.9a:  Fehler an Position ', Fehlercode);

s1 := '47,11';
Val(s1, y, Fehlercode);
```

```
  if Fehlercode = 0
     then WriteLn(y)
     else WriteLn(' Nr 14.9b:  Fehler bei "', s1, '" an Position ',
                                                    Fehlercode);
               { ¦ Nr 14.9b:  Fehler bei "47,11" an Position 3 }

  { Mit der Prozedur "Val" kann man in Verbindung mit einer
    repeat/until-Schleife Numerik-Eingaben absichern. Bekanntlich
    bricht das Programm ab, wenn eine Numerik-Eingabe unzulässige
    Zeichen enthält. Um dieses zu vermeiden, zieht man den ge-
    wünschten Wert zunächst als String ein, wandelt dieses String
    mit "Val" in einen numerischen Wert um. Der Vorgang wird in
    der Schleife solange wiederholt, bis der Fehlercode den
    Wert 0 erhält.
  }
  repeat
    WriteLn(' Nr 14.9c:  Demo Numerik-Absicherung:');
    Write(  '            Realtyp zuerst fehlerhaft, ' +
                         'dann richtig eingeben: ');
    ReadLn(yStr);   { String einziehen }
    Val(yStr, y, Fehlercode);
  until Fehlercode = 0;

  (*  Eine primitivere Möglichkeit der Numerikabsicherung bietet
      der Compilerschalter "{$I-}" in Verbindung mit der
      Standardfunktion "IOResult":
      ...;
      Write('     ... oder Numerikabsicherung mit "IOResult": ');
      {$I-}   { Automatische Input/Output-Kontrolle abschalten }
      repeat
        ReadLn(y);
      until IOResult = 0;
      {$I+}  { Automatische Input/Output-Kontrolle einschalten }
      ...;
  *)
```

{ 14.10: Der String als "array of Char" betrachtet

```
  }
  s1    := 'Huber';
  s1[3] := '?';
  WriteLn(' Nr 14.10a: ', s1);
        { ¦ Nr 14.10a: Hu?er   }

  Write(' Nr 14.10b: ');
      { ¦ Nr 14.10b: Hu?er  }
  for i := 1 to Length(s1) do
    Write(s1[i]);
  WriteLn;
```

{ 14.11: Die Prozedur "FillChar" (einen String bis zur deklarierten Länge mit gleichen Zeichen füllen)

Format: FillChar(*v, anzahl, zeichen*)

v: Variable, hier Stringvariable
anzahl: Word-Ausdruck. Anzahl der Zeichen (Bytes)

```
          zeichen: Char-Ausdruck.

  Beispiel:

  Der String "s1" wurde eingangs mit max. 10 Zeichen ver-
  einbart. Er soll ganz mit dem Zeichen '%' (Ordnungs-Nr 37)
  gefüllt werden.

  Die deklarierte Maximallänge incl. Längenbyte kann mit der
  Funktion "SizeOf()" auch abgefragt werden. Die Funktion
  "SizeOf()" liefert den Speicherbedarf des Arguments in Bytes
  und hat den Datentyp Word. Das Argument kann jede beliebige
  Variablen-Bezeichner oder ein Datentyp-Bezeichner sein.
  }

  FillChar(s1, 10 + 1, '%');     { 10 Zeichen + 1 Längenbyte }
  s1[0] := Chr(10);   { Das Längenbyte im Byte Nr. 0 belegen }
  WriteLn(' Nr 14.11a: ', s1, ' ', Length(s1));
        { ¦ Nr 14.11a: %%%%%%%%%% 10  }

  { oder .... }

  FillChar(s1, SizeOf(s1), '%');
  s1[0] := Chr(SizeOf(s1) - 1);
                               { Das Längenbyte im Byte Nr. 0 }

  WriteLn(' Nr 14.11b: ', s1, ' ', Length(s1));
        { ¦ Nr 14.11b: %%%%%%%%%% 10  }

  { oder .... }

  s1 := '';   { Leerstring, Länge 0 }
  for i := 1 to SizeOf(s1) - 1 do
    s1 := s1 + '%';

  WriteLn(' Nr 14.11c: ', s1, ' ', Length(s1));
        { ¦ Nr 14.11c: %%%%%%%%%% 10  }
```

{ 14.12: Vergleichsoperatoren für Strings

```
  Auf Strings können die üblichen sechs Vergleichsoperatoren

                =   >    <    >=   <=    <>

  angewendet werden.

  Beim Stringvergleich werden die Zeichen beider Strings von
  links beginnend verglichen. Die Ordnungsnummer (ASCII-Nr)
  wird zum Vergleich der Zeichen herangezogen. Wenn erstmals
  ein Unterschied auftaucht, wird der String, der das Zeichen
  mit der höheren Ordnungsnummer besitzt, als der "größere"
  String gewertet und der Vergleich beendet; die Länge der
  Strings ist nicht maßgebend.
  }

  s1 := 'AntoNius';
  s2 := 'Anton';

  if s1 > s2
     then WriteLn(' Nr 14.12:  String #1 größer')
     else WriteLn(' Nr 14.12:  String #2 größer');
               { ¦ Nr 14.12:  String #2 größer }
  { Ordnungsnummer von 'N':  78 }
  { Ordnungsnummer von 'n': 110 }
```

```
  { 14.13: String und Character
    Lediglich ein String-Element kann auf einen Character zuge-
    wiesen werden, nicht aber ein String mit der Länge 1.
    }
    s1 := 'Anton Huber';
    Ch   := s1[7];   { "Ch" eingangs als "Char"-Typ deklariert }
    WriteLn(' Nr 14.13:  ', Ch);
          { ¦ Nr 14.13:  H  }

  { 14.14: Konventionelle Übergabe von Strings an Routinen
    Bei der konventionellen Übergabe eines Strings als Parameter
    an eine Prozedur oder Funktion muß bei der Deklaration des
    formalen Parameters in der Routine ein (vorher mit "type"
    deklarierter) eigener String-Datentyp verwendet werden.
    Es können dann auch nur passende Strings übergeben werden,
    was eine Einschränkung bedeutet, wenn man die gleiche Routine
    mit verschieden deklarierten Strings benutzen möchte.
    Siehe auch Kap. 14.14: Übergabe von "offenen Strings".
    }
    KonventionelleStringUebergabe(s3);
  { ------------------------------------------------------------ }
  repeat
  until ReadKey <> '';
end.
```

14.15 Übergabe von offenen String-Parametern an Routinen

Ab Turbo-Pascal 7.0 verschieden lang deklarierte Strings an die gleiche Prozedur oder Funktion als Parameter übergeben werden. Dazu stehen bei der Deklaration des formalen String-Paramters zwei Möglichkeiten zur Verfügung:

- Verwendung des neuen Datentypbezeichners "OpenString" (kein reserviertes Wort).
- Mit dem reservierten Wort "string" in der Compilerschalter-Einstellung "{$P+}" oder der entsprechenden Menüeinstellung "Option/Compiler../Offene Arraygrenzen".

Die Art der Übergabe, mit Wert oder mit Adresse, ist davon nicht betroffen.

Das folgende Demo-Programm zeigt die Vorgehensweise.

```
{$P+  Compilerschalter für offene Arrays und Strings }
{ Nicht notwendig bei Verwendung des neuen Bezeichners "OpenString" }
program Pas14151;   { "Pas14151.PAS": Offene Strings }

uses
  CRT;

var
  s3: string[3];
  s6: string[6];

procedure OffeneStrings(var s: OpenString);
begin   { Hier Übergabe mit Adresse "var" }
  Write(s);
  s := 'Meier';
end;

begin
  ClrScr;
  s3 := 'ABC';     OffeneStrings(s3);  WriteLn(' ', s3, Length(s3));
  s6 := 'ABCDEF';  OffeneStrings(s6);  WriteLn(' ', s6, Length(s6));
  { ¦ABC Mei3      }
  { ¦ABCDEF Meier5 }
  repeat
  until KeyPressed;
end.
```

14.16 Lange Strings bis 64 KByte: Ein Überblick

Die bisher behandelten (Standard-) Pascal-Strings hatten eine maximale Länge von 255 Byte; limitiert durch das vorangestellte Längenbyte, das zugleich das Stringzeichen mit dem Index 0 darstellt. Das erste "echte" Zeichen des Strings hatte somit den Index 1, was sehr anschaulich ist.

Ab Version 7.0 werden von Turbo-Pascal zusätzlich Strings bis zu einer Länge von 64 KByte (65535 Byte) unterstützt. Diese Strings werden - wie in C - nullterminiert, d.h. sie haben kein vorausgestelltes Längenbyte (es müßten ansonsten 2 Byte) sein, sondern werden am aktuellen Ende mit dem Nullbyte (Chr(0), #0) begrenzt, das automatisch hinzugefügt wird. Die Begrenzung auf 64 KByte ist nur durch das Betriebssystem MS-DOS bzw. durch die alten 16-bit-Prozessoren gegeben. Das auf Pascal aufsetzende und nur unter Windows auf Rechnern mit Prozessoren ab i80486 laufende grafische Entwicklungssystem Borland Delphi ist auch die 64-KByte-Begrenzung nicht mehr gegeben; dort können die Strings im Rahmen des verfügbaren Speichers beliebig lang sein ("... die ganze Bibel auf einen String ...").

Zurück zu den 64-KByte-Strings:

Für die Behandlung von langen Strings wird die Unit "STRINGS" benötigt. Weiter wird die Aktivierung der "**Erweiterten Syntax**" vorausgesetzt (entweder über Menü "Option/Compiler..." oder mit Compilerschalter "{$X+}" einzustellen). Damit *kann* bei allen Funktionen (ausgenommen Funktionen aus der Unit "SYSTEM"), also nicht nur bei den Langstring-Funktionen, auf den Rückgabewert verzichtet werden, d.h. im Bedarfsfall können Funktionen wie Prozeduren verwendet werden.

Der String wird als "array[0..n] of Char" betrachtet. Somit hat das erste Zeichen des Strings den Index 0, was gewöhnungsbedürftig ist. Ansonsten werden die Strings nicht statisch (wie alle bisher behandelten Variablen) behandelt, sondern dynamisch mittels Zeigern.

Die Unit "STRINGS" enthält mit "PChar" einen neuen Datentyp, der einen Zeiger (Pointer auf Character) darstellt:

```
type
  PChar: ^Char
```

Zeiger (Pointer) werden erst im Kap. 19 behandelt.

Die Unit "STRINGS" enthält darüber hinaus 21 Funktionen zur Behandlung von langen Strings.

Die String-Funktionen der Unit "STRINGS"

Wenn das Ergebnis dieser Funktionen ein String ist (genauer String-Zeiger), ist er vom Typ "PChar". Die Strings *s*, *s1* und *s2* seien ebenfalls vom Typ "PChar", die – wenn nicht später ausdrücklich von Variablen die Rede ist – auch String-Konstanten sein können.

---

function StrCat(*s1, s2*: PChar): PChar;
Concate. Hängt einen Kopie des Quellstrings *s2* an den Zielstringvariable *s1* an.

function StrComp(*s1, s2*: PChar): Integer;
Compare. Vergleicht zwei Strings *s1* und *s2* und liefert bei Gleichheit den Wert 0, bei *s1* > *s2* einen Wert größer 0 (Differenz der Ascii-Codes beim ersten verschieden Zeichen) und bei *s1* < *s2* einen Wert kleiner 0 (ebenfalls Ascii-Code-Differenz). Siehe auch Funktion StrIComp.

function StrCopy(*s1, s2*: PChar): PChar;
Kopiert den Quellstring *s2* in die Zielstringvariable *s1*.

function StrDipose(*s*: PChar);
Entfernt den String vom Heap.

function StrECopy(*s1, s2*: PChar): PChar;
Kopiert den Quellstring *s2* in die Zielstringvariable *s1* und liefert einen Zeiger auf das Ende von *s1*.

function StrEnd(*s*: PChar): PChar;
Liefert einen Zeiger auf das Null-Byte, das die Stringvariable *s* terminiert.

function StrIComp(*s1, s2*: PChar): Integer;
Vergleicht zwei Strings ohne Beachtung von Groß-/Kleinschreibung (I = Ignore). Sonst wie Funktion StrComp.

function StrLCat(*s1, s2*: PChar; *lMax:* Word): PChar;
Hängt maximal (*lMax* - StrLen(*s1*)) Zeichen von *s2* an die Stringvariable *s1* an.

function StrLComp(*s1, s2*: PChar; *l*: Word): Integer;
Vergleicht die beiden Strings bis zur vorgegebenen Anzahl *l* der Zeichen. Sonst wie StrComp.

function StrLCopy(*s1, s2*: PChar; *lMax*: Word): PChar;
Kopiert maximal *lMax* Zeichen von *s2* in die Stringvariable *s1*.

function StrLen(*s*: PChar): Word;
Liefert die Anzahl der Zeichen von *s.*

function StrLICopy(*s1, s2*: PChar; *lMax*: Word): Integer;
Vergleicht die beiden Strings bis zu *lMax*-Zeichen ohne Berücksichtigung von Groß-/Kleinschreibung. Sonst wie StrComp.

function StrLower(*s*: PChar): PChar;
Konvertiert den String *s* in Kleinbuchstaben. Siehe auch StrUpper.

function StrMove(*s1, s2*: PChar; *n*: Word): PChar;
Kopiert n-Zeichen von *s2* in den Anfang der Stringvariablen *s1*.

function StrNew(*s*: PChar): PChar;
Legt eine Kopie von *s* auf dem Heap an. Siehe auch Dispose.

function StrPas(*s*: PChar): string;
Konvertiert einen nullterminierten String in einen Pascal-String. Siehe auch StrPCopy.

function StrPCopy (*s1*: PChar, s2: string);
Konvertiert den Pascal-String *s2* in die (nullterminierte) Stringvariable *s1*. Siehe auch StrPas.

function StrPos(*s1, s2*: PChar): PChar;
Liefert einen Zeiger auf das erste Vorkommen von *s2* in *s1*. Wenn *s2* in *s1* nicht vorkommt, wir "nil" (not in list) geliefert.

function StrRScan(*s*: PChar; *Ch*: Char): PChar;
Liefert einen Zeiger auf das **letzte** Vorkommen des Zeichens Ch in s. Wenn Ch nicht darin vorkommt, wird "nil" geliefert. Das terminierende Null-Byte wird als Zeichen des Strings betrachtet.

function StrScan(*s*: PChar; *Ch*: Char): PChar;
Liefert einen Zeiger auf das **erste** Vorkommen des Zeichens Ch in *s*. Wenn Ch nicht darin vorkommt, wird "nil" geliefert. Das terminierende Null-Byte wird als Zeichen des Strings betrachtet.

function StrUpper(*s*: PChar): PChar;
Konvertiert den String *s* in Großbuchstaben. Siehe auch StrLower.

Das folgende Kurz-Demo-Programm soll die grundsätzliche Handhabung der langen Strings an ausgewählten Beispielen demonstrieren. Für weitergehende Anwendung ist die integrierte Hilfestellung heranzuziehen.

```
{$X+ Erweiterte Syntax für lange Strings notwendig.                  }
{    Mit der erweiterten Syntax  k ö n n e n  (müssen aber nicht)    }
{    Funktionen, soweit sie nicht in der Standard-Unit "SYSTEM"      }
{    deklariert sind, wie Prozeduren verwendet werden. Es wird dann  }
{    der Rückgabewert ignoriert. }
program Pas14161;  { "Pas14161.PAS", Demo "lange Strings" bis 64 KB  }
                   { mit Null-Terminierung, ab Turbo-Pascal 7.0      }
                   { Dr. K. Haller, FH München, DR,        26240597  }
  { Achtung: Der vordefinierte Zeiger "PChar" (type PChar = ^Char)   }
  {          erlaubt den Zugriff auf lange Strings, setzt aber       }
  {          voraus, daß die Option "Erweiterte Syntax", "$X+",      }
  {          eingestellt ist.                                        }
uses
  CRT, STRINGS; { Unit "Strings" mit Typ "PChar" und 21 Funktionen   }
                { für lange Strings. Sie beginnen alle mit "Str..."  }
var
  i, l:  Word;
  s:     array[0..6000] of Char;
  s1,
  s2:    PChar;

procedure Tastendruck;
begin
  ReadKey;  { So nur mit erweiterter Syntax möglich, sonst z.B.  }
  ClrScr;   {              repeat until ReadKey <> ''            }
end;        { o.ä., was auch bei erweiterter Syntax möglich ist. }

begin
  TextBackGround(Blue);
  TextColor(Yellow);
  ClrScr;

  WriteLn('Demo 1:');
  s1    := ' 2345678901234567890xxxxxxx Huber Anton ÄÖÜ äöü ß xxxx';
  s1[0] := '1';
  s2    := 'yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy';
  WriteLn(s1, StrLen(s1));
  WriteLn(s2, StrLen(s2));
  Tastendruck;

  WriteLn('Demo 2');
  StrCopy(s, s1);
  for i := 1 to 9 do
    StrCat(s, s2);
  s[9] := '?';
  WriteLn(s, StrLen(s));
  Tastendruck;

  WriteLn('Demo 3');
  StrUpper(s);
  WriteLn(s, StrLen(s));
  Tastendruck;
```

```
  WriteLn('Demo 4');
  StrLower(s);
  WriteLn(s, StrLen(s));
  Tastendruck;

  WriteLn('Demo 5');
  for i := 1 to StrLen(s) do
    s[i-1] := '.';
  WriteLn(s, StrLen(s));
  Tastendruck;

  WriteLn('Demo 6');           { Liefert: 4 = Ord('e') - Ord('a')  }
  WriteLn(StrComp('Meier', 'Maier' ));  { "StrComp" liefert Ascii- }
  Tastendruck;                 { Differenz beim ersten Unterschied }

  WriteLn('Demo 7');           { Liefert: 0 }
  WriteLn(StrComp('Huber Anton', 'Huber' + ' ' + 'Anton' ));
  Tastendruck;

  WriteLn('Demo 8');
  WriteLn(StrMove(s, 'Huber Toni', 10));
  Tastendruck;
end.
```