Programmieren 1

Wintersemester 2016/2017

Marcus Riemer, B.Sc.

Basierend auf den Unterlagen „Programmstrukturen 1“
von Prof. Dr. Andreas Häuslein

9 writeln und concat

Ausgabe mit `writeln` ist flexibler als `concat`

- `writeln` erlaubt die Angabe von gemischten Datentypen
- `concat` erlaubt lediglich zusammenfügen von String-Typen

```
program out_writeln;
var
   someBool   : Boolean;
   someInt    : Integer;
   someReal   : Double;
   someString : String;
begin
   // Werte zunächst einlesen, dann
   writeln(someBool, someInt,
           someReal, someString);
end.
```

```
program out_concat;
var
   // Siehe links, zusätzlich:
   outputBuffer : String;
   convStr      : String;
begin
   // Werte zunächst einlesen, dann
   str(someBool, convStr);
   outputBuffer := convStr;
   str(someInt, convStr);
   outputBuffer := concat(outputBuffer, convStr);
   str(someReal, convStr);
   outputBuffer := concat(outputBuffer, convStr);
   outputBuffer := concat(outputBuffer, someString);

   writeln(outputBuffer);
end.
```

9 Pufferung von Ausgaben

Typischer Einsatzzweck von Strings: Ausgaben verzögern

- Wiederholte Aufrufe von `writeln` oder ähnlichen Funktionen sind vergleichsweise teuer.
- Ausgaben können nicht rückgängig gemacht werden, im Fehlerfall ist dann möglicherweise eine teilweise Ausgabe zu sehen.
- Also: Ausgaben in eine Puffervariable speichern, statt Sie mit `write` direkt auszugeben

9 Pufferung von Ausgaben

```
program comma_add;

var
   userInput, outputBuffer : string;
   currString : string;
   sum, currInt, commaPos, valCode : Integer;

begin
   outputBuffer := '';
   sum := 0;

   write('Geben Sie eine komma-getrennte Liste von ganzen Zahlen ein: ');
   readln(userInput);

   // Keine Zwischen-Ausgaben nach dieser Zeile, nur Fehlermeldungen
   commaPos := pos(',', userInput);
   while(commaPos > 0) do
```

9 Pufferung von Ausgaben

```
// Keine Zwischen-Ausgaben nach dieser Zeile, nur Fehlermeldungen
commaPos := pos(',', userInput);
while(commaPos > 0) do
begin
   currString := copy(userInput, 1, commaPos - 1);
   // writeln('commaPos = ', commaPos);
   // writeln('currString = ', currString);
   outputBuffer := concat(outputBuffer, currString, ' + ');
   val(currString, currInt, valCode);
   if (valCode > 0) then
   begin
      writeln('Fehlerhaftes Zeichen an Stelle ', valCode, ' in "', currString, '"');
      commaPos := 0;
   end
   else
   begin
      sum := sum + currInt;
      delete(userInput, 1, commaPos);
      commaPos := pos(',', userInput);
   end
end;
```

Debug-Hilfe

9 Pufferung von Ausgaben

```
  delete(outputBuffer, length(outputBuffer) - 2, 2);
  writeln(outputBuffer, '= ', sum);
end.
```

- Schleife wird durch direkte Veränderung der Variable `commaPos` abgebrochen
- Ungültige Werte können erst nach einem kompletten Durchlauf der Schleife ausgeschlossen werden.
  - Daher Pufferung der Ausgabe in der Variable `outputBuffer`
  - Zuviel hinzugefügtes ‘ + ‘ kann vor der Ausgabe am Ende gelöscht werden.
- **Noch (mindestens) ein Fehler im Programm:**
  - Der letzte Wert wird nicht eingelesen.

9 Konvertierungsfunktionen

- Bei `val` und `str` handelt es sich um Prozeduren, daher immer der „Umweg“ über eine Variable erforderlich.
- Speziellere Funktionen verfügbar
  - `IntToStr  ( Value : Integer )  : String;`
  - `StrToInt  ( Value : String )  : Integer;`
  - `FloatToStr ( Value : Extended ) : String;`
  - `StrToFloat ( Value : String )  : Extended;`
  - `BoolToStr  ( Value : Boolean )  : String;`
  - `StrToBool  ( Value : String )  : Boolean;`
- Wesentliche Nachteile
  - Anpassung von Aufrufen nötig, wenn sich ein Datentyp verändert.
  - Verhalten beim Wandeln von unpassenden Werten wie `readln`.
    - Zur Umwandlung also lieber weiterhin `val` verwenden.
  - Nicht Teil des Delphi-Standards, dennoch sehr weit verbreitet
    - Einbindung des Moduls `System.SysUtils` erforderlich

9 Konvertierungsfunktionen

```
program out_concat;
var
   // Siehe links, zusätzlich:
   outputBuffer : String;
   convStr      : String;
begin
   // Werte zunächst einlesen, dann
   str(someBool, convStr);
   outputBuffer := convStr;
   str(someInt, convStr);
   outputBuffer := concat(outputBuffer, convStr);
   str(someReal, convStr);
   outputBuffer := concat(outputBuffer, convStr);
   outputBuffer := concat(outputBuffer, someString);

   writeln(outputBuffer);
end.
```

```
program out_concat_func;

uses
  System.SysUtils;

var
   someBool   : Boolean;
   someInt    : Integer;
   someReal   : Double;
   someString : String;

   outputBuffer : String;
begin
   // Werte zunächst einlesen, dann
   outputBuffer := concat(
      BoolToStr(someBool),
      IntToStr(someInt),
      FloatToStr(someReal),
      someString
   );

   writeln(outputBuffer);
end.
```

9 Formatierung

Mit einem Doppelpunkt hinter einem aktuellen Parameter kann bei den Prozeduren `writeln` und `str` die Breite der Ausgabe gesteuert werden. Für Fließkommazahlen können mit einem weiteren Doppelpunkt die Anzahl der Nachkommastellen angegeben werden.

```
program string_format;
uses Math;
var  i :  Integer;
begin
   writeln('Dieser Werte beruehren sich rechts':50);
   writeln('Einfache Zahlen':50);
   for i := 0 to 9 do
      writeln(i:50);
   writeln('Zehnerpotenzen (normale Schreibweise)':50);
   for i := 0 to 9 do
      writeln(power(10, i):50);
   writeln('Zehnerpotenzen (eine Nachkommastelle)':50);
   for i := 0 to 9 do
      writeln(power(10, i):50:1);
   writeln('Zehnerpotenzen (keine Nachkommastelle)':50);
   for i := 0 to 9 do
      writeln(power(10, i):50:0);
   writeln('Zehnerpotenzen (als Ganzzahl)':50);
   for i := 0 to 9 do
      writeln(trunc(power(10, i)):50);
end.
```

9 Aufgabe „Komponentenweise Addition”

Bei dieser Aufgabe sollen mindestens vier Eingaben mit nur zwei `readln`-Aufrufen entgegengenommen werden. Diese vier Eingabewerte werden dann komponentenweise addiert und formatiert ausgegeben.

- Lest vom Benutzer zwei Strings mit jeweils zwei durch einen Doppelpunkt als Trennzeichen getrennten Werte ein.
- Trennt den String am Trennzeichen auf und addiert den ersten Wert der ersten Eingabe zum ersten Wert der zweiten Eingabe. Für den zweiten Wert soll genauso verfahren werden.
  - `10:22<Enter>3:16` berechnet also 10 + 3 und 22 + 16
- Gebt diese Werte nach dem aus der Schule bekannten Turm-Schema aus:

```
10   22
 +    +
 3   16
 =    =
13   38
```

9 Absturzfreies einlesen mit der `val`-Prozedur

Problem: Unpassende Eingaben werden von der `readln`-Prozedur mit einem (momentan) nicht behandelbaren Fehler quittiert.

```
program input_readln;
{$APPTYPE CONSOLE}
{$R+,Q+,X-}
var
   targetInput : byte;
begin
   write('Geben sie einen ganzzahligen Wert ein:');
   readln(targetInput);
end.
```

```
File   Edit   View   Bookmarks   Settings   Help
marcus@mri-tp:~/f/w/p/programme >>> ./9-input-readln
Geben sie einen ganzzahligen Wert ein: a
Runtime error 106 at $0000000000400219
  $0000000000400219
  $000000000040018F

marcus@mri-tp:~/f/w/p/programme >>>
```

Lösung: Wert zunächst in einen String einlesen, und diesen dann sorgfältig prüfen.

9 Absturzfreies einlesen mit der `val`-Prozedur

`Val(S, V, Code)`
Prozedur, die den String `S` in die Zahl `V` umwandelt, falls `S` ein zulässiges* Zahlenformat darstellt. Ansonsten wird in `Code` (Integer) die Position des ersten unzulässigen Zeichens zurückgeliefert

*Zulässigkeit hängt vom Typ von `V` ab

```
program input_val;
{$APPTYPE CONSOLE}
{$R+,Q+,X-}
var
   userInput : string;
   targetInput : byte;
   valCode : integer;
begin
   repeat
      write('Geben sie einen ganzzahligen Wert ein:');
      readln(userInput);
      val(userInput, targetInput, valCode);
      if (valCode > 0) then
         writeln('Falsches Zeichen an Stelle ', valCode);
   until valCode = 0;
end.
```

```
File  Edit  View  Bookmarks  Settings  Help
marcus@mri-tp:~/f/w/p/programme >>> ./9-input-val
Geben sie einen ganzzahligen Wert ein: a
Falsches Zeichen an Stelle 1
Geben sie einen ganzzahligen Wert ein: -12
Falsches Zeichen an Stelle 2
Geben sie einen ganzzahligen Wert ein: 31,3
Falsches Zeichen an Stelle 3
Geben sie einen ganzzahligen Wert ein: 31.3
Falsches Zeichen an Stelle 3
Geben sie einen ganzzahligen Wert ein: 31:3
Falsches Zeichen an Stelle 3
Geben sie einen ganzzahligen Wert ein: 12
marcus@mri-tp:~/f/w/p/programme >>>
```

Einlesen & Wandeln

Fehlermeldung & Erneute Eingabe

9 Einlesen mehrerer Werte mit Strings

`Pos(Substring, S)`
Funktion, die im String `S` nach einem Vorkommen des Teilstrings `Substring` sucht und als Ergebnis die Position des ersten Zeichens von `Substring` in `S` liefert. Ist `Substring` nicht in `S` enthalten, ist das Ergebnis 0

```
program input_val;
{$APPTYPE CONSOLE}
{$R+,Q+,X-}
var
   userInput : string;
const
   TRENNZEICHEN = '$';
begin
   write('Zwei durch ', TRENNZEICHEN, ' separierte Zahlen ein?');
   readln(userInput);

   writeln('DEBUG 1: ', TRENNZEICHEN, ' an Stelle ',
           pos(TRENNZEICHEN, userInput));
end.
```

9 Einlesen mehrerer Werte mit Strings

`Copy (S, Index, Count)`
Funktion, die beginnend an der Position `Index` genau `Count` Zeichen aus dem String `S` kopiert und den Teilstring als Ergebnis liefert

```
var
   userInput : string;

const
   TRENNZEICHEN = '$';

begin
   write('Zwei durch ', TRENNZEICHEN, ' separierte Zahlen? ');
   readln(userInput);

   writeln('DEBUG 1: ', TRENNZEICHEN, ' an Stelle ',
           pos(TRENNZEICHEN, userInput));
   writeln('DEBUG 2: ', copy(userInput,
                             1,
                             pos(TRENNZEICHEN, userInput) - 1));
   writeln('DEBUG 3: ', copy(userInput,
                             pos(TRENNZEICHEN, userInput) + 1,
                             length(userInput)));
end.
```

Trennzeichen

Wert davor

Wert danach

9 Einlesen mehrerer Werte mit Strings

`Copy (S, Index, Count)`
Funktion, die beginnend an der Position `Index` genau `Count` Zeichen aus dem String `S` kopiert und den Teilstring als Ergebnis liefert

```
var
   userInput : string;
   dividerPos : integer;

const
   TRENNZEICHEN = '$';

begin
   write('Geben sie zwei durch ', TRENNZEICHEN, ' separierte Zahlen ein: ');
   readln(userInput);

   dividerPos := pos(TRENNZEICHEN, userInput);

   writeln('DEBUG 1: ', TRENNZEICHEN, ' an Stelle ', dividerPos);
   writeln('DEBUG 2: ', copy(userInput, 1, dividerPos - 1));
   writeln('DEBUG 3: ', copy(userInput, dividerPos + 1, length(userInput)));
end.
```

Trennzeichen

Wert davor

Wert danach

9 Einlesen mehrerer Werte mit Strings

```
var
   userInput : string;
   dividerPos : integer;
   targetVal1, targetVal2 : integer;
   valCode1, valCode2 : integer;

const
   TRENNZEICHEN = '$';

begin
   write('Geben sie zwei durch ', TRENNZEICHEN, ' separierte Zahlen ein: ');
   readln(userInput);

   dividerPos := pos(TRENNZEICHEN, userInput);

   val(copy(userInput, 1, dividerPos - 1), targetVal1, valCode1);
   val(copy(userInput, dividerPos + 1, length(userInput)), targetVal2, valCode2);
end.
```

valCode wird nicht beachtet

Trennzeichen

Wert davor

Wert danach

9 Aufgabe “Textersetzung”

- Lesen Sie vom Benutzer zunächst einen beliebigen String ein und speichern Sie diesen in der Variable `benutzerText`.
- Lesen Sie dann zwei beliebige reelle Zahlen vom Benutzer ein, die durch ein Semikolon getrennt werden sollen.
- Geben sie diese beiden Zahlen mit jeweils zwei Nachkommastellen aus.
- Nehmen Sie dann die folgenden, immer komplexer werdenden Ersetzungen vor:
  - Geben Sie an Stelle der Zeichenkette `<1>` im benutzerText die erste vom Benutzer eingegebene Zahl aus. Für diese Aufgabe benötigen Sie in der Ausgabe lediglich die Funktionen `pos` und `copy`.

9 Aufgabe “Textersetzung”

- Entfernen Sie alle doppelten Leerzeichen aus dem `benutzerText`. Sie benötigen dafür eine Schleife sowie die Routinen `pos` und `delete`.
- Ersetzen Sie mittels der Routinen `delete` und `insert` das erste Vorkommen der Zeichenkette `<1>` und das erste Vorkommen der Zeichenkette `<2>` mit der jeweils ersten bzw. zweiten reellen Zahl. Neben den erwähnten Routinen benötigen Sie mindestens eine weitere Variable, jedoch keine Schleife.
- Ersetzen Sie jedes Vorkommen der Zeichenkette `<1>` bzw. `<2>` in diesem String durch die erste bzw. zweite reelle Zahl. Für diese Aufgabe benötigen Sie eine Schleife, `pos`, `delete` sowie `insert`.