Aufgabe 2 – Klassen für Zeichenketten-Operationen

Lösungsidee:

(a) Die Klasse StringBuilder enthält eine Datenkomponente buffer vom Typ STRING sowie die weiteren Methoden zum Anhängen verschiedener Typen an den String.

(b) Die abgeleitete Klasse TabStringBuilder erbt von der Klasse StringBuilder und überschreibt die Methoden, um Elemente spaltenweise mit Leerzeichen aufzurichten, indem Sie jedes Mal den hereinkommenden Typen so bearbeitet das er genau 1 spalte breit ist und den Rest mit Leerzeichen befüllt. Die Spaltenbreite wird beim Erstellen eines TabStringBuilder-Objekts festgelegt.

(c) Die Klasse StringJoiner verwendet eine Datenkomponente vom Typen StringBuilder zur Verkettung von Zeichenketten und Trennzeichen. Sie enthält einen Konstruktor Init, um ein StringJoiner-Objekt mit einem Trennzeichen delimiter zu initialisieren, die Methode Add, um eine Zeichenkette e mit dem Trennzeichen an den StringJoiner anzufügen, und die Methode AsString, um das Ergebnis als Zeichenkette zurückzugeben.

Zeitaufwand: ~1h

Code:

```
unit StringBuilderUnit;

interface

type
  StringBuilderPtr = ^StringBuilderObj;
  StringBuilderObj = object
  public
    constructor Init;
    destructor Done; virtual;

    procedure AppendStr(e: string); virtual;
    procedure AppendChar(e: char); virtual;
    procedure AppendInt(e: integer); virtual;
    procedure AppendBool(e: boolean); virtual;
    function AsString: string; virtual;
    function BufferLength: integer;
  private
```

```
    buffer: string;
  end;

function NewStringBuilder: StringBuilderPtr;

implementation

function NewStringBuilder: StringBuilderPtr;
var
  builder: StringBuilderPtr;
begin
  New(builder, init);
  NewStringBuilder := builder;
end;

constructor StringBuilderObj.Init;
begin
  buffer := '';
end;

destructor StringBuilderObj.Done;
begin
end;

procedure StringBuilderObj.AppendStr(e: string);
begin
  buffer := buffer + e;
end;

procedure StringBuilderObj.AppendChar(e: char);
begin
  buffer := buffer + e;
end;

procedure StringBuilderObj.AppendInt(e: integer);
```

```
var
  intStr: string;
begin
  Str(e, intStr);
  buffer := buffer + intStr;
end;

procedure StringBuilderObj.AppendBool(e: boolean);
begin
  if e then
    buffer := buffer + 'TRUE'
  else
    buffer := buffer + 'FALSE';
end;

function StringBuilderObj.AsString: string;
begin
  AsString := buffer;
end;

function StringBuilderObj.BufferLength: integer;
begin
  BufferLength := Length(buffer);
end;

end.
```

```
unit TabStringBuilderUnit;

interface

uses
  StringBuilderUnit;
```

```
type
  TabStringBuilderPtr = ^TabStringBuilderObj;
  TabStringBuilderObj = object(StringBuilderObj)
  public
    constructor Init(width: integer);
    destructor Done; virtual;
    procedure AppendStr(e: string); virtual;
    procedure AppendChar(e: char); virtual;
    procedure AppendInt(e: integer); virtual;
    procedure AppendBool(e: boolean); virtual;
  private
    columnWidth: integer;
    function AlignText(text: string): string;
  end;

function NewTabStringBuilder(width: integer):
TabStringBuilderPtr;

implementation

function NewTabStringBuilder(width: integer):
TabStringBuilderPtr;
var
  builder: TabStringBuilderPtr;
begin
  New(builder, Init(width));
  NewTabStringBuilder := builder;
end;

constructor TabStringBuilderObj.Init(width: integer);
begin
  inherited Init;
  columnWidth := width;
end;
```

```
destructor TabStringBuilderObj.Done;
begin
  inherited done;
end;

procedure TabStringBuilderObj.AppendStr(e: string);
begin
  inherited AppendStr(AlignText(e));
end;

procedure TabStringBuilderObj.AppendChar(e: char);
begin
  inherited AppendStr(AlignText(e));
end;

procedure TabStringBuilderObj.AppendInt(e: integer);
var
  intStr: string;
begin
  Str(e, intStr);
  inherited AppendStr(AlignText(intStr));
end;

procedure TabStringBuilderObj.AppendBool(e: boolean);
begin
  if e then
    inherited AppendStr(AlignText('TRUE'))
  else
    inherited AppendStr(AlignText('FALSE'));
end;

function TabStringBuilderObj.AlignText(text: string):
string;
var
  temp: string;
```

```
begin
  if Length(text) >= columnWidth then
    temp := Copy(text, 1, columnWidth)
  else
  begin
    temp := text;
    while Length(temp) < columnWidth do
      temp := Concat(temp, ' ');
  end;
  AlignText := temp;
end;

end.
```

```
unit StringJoinerUnit;

interface

uses
  StringBuilderUnit;

type
  StringJoinerPtr = ^StringJoinerObj;
  StringJoinerObj = object
  public
    constructor Init(delimiter: char);
    destructor Done; virtual;

    procedure Add(e: string);
    function AsString: string;
  private
    delimiter: char;
    count: integer;
    resultBuilder: StringBuilderPtr;
```

```
  end;

function NewStringJoiner(delimiter: char):
StringJoinerPtr;

implementation

function NewStringJoiner(delimiter: char):
StringJoinerPtr;
var
  joiner: StringJoinerPtr;
begin
  New(joiner, Init(delimiter));
  NewStringJoiner := joiner;
end;

constructor StringJoinerObj.Init(delimiter: char);
begin
  self.delimiter := delimiter;
  count := 0;
  resultBuilder := NewStringBuilder;
end;

destructor StringJoinerObj.Done;
begin
  Dispose(resultBuilder, Done);
end;

procedure StringJoinerObj.Add(e: string);
begin
  if count > 0 then
    resultBuilder^.AppendChar(delimiter);
  resultBuilder^.AppendStr(e);
  Inc(count);
end;
```

```
function StringJoinerObj.AsString: string;
begin
  AsString := resultBuilder^.AsString;
end;

end.
```

Test:

```
program TestStringBuilder;

uses
  StringBuilderUnit, TabStringBuilderUnit;

procedure ExecuteStringBuilderTests(builder:
StringBuilderPtr);
begin
  // Append different types of values to the
StringBuilder
  builder^.AppendStr('Hello ');
  builder^.AppendChar('W');
  builder^.AppendChar('o');
  builder^.AppendChar('r');
  builder^.AppendChar('l');
  builder^.AppendChar('d');
  builder^.AppendInt(2023);
  builder^.AppendBool(true);
  builder^.AppendStr('123456789');

  // Get the resulting string from the StringBuilder
  Writeln('StringBuilder content: ', builder^.AsString);
end;
```

```
var
  myBuilder: StringBuilderPtr;
  myTabBuilder: TabStringBuilderPtr;

begin
  myBuilder := NewStringBuilder;
  myTabBuilder := NewTabStringBuilder(8);

  Writeln('Testing StringBuilder:');
  ExecuteStringBuilderTests(myBuilder);

  Dispose(myBuilder, Done);

  writeln; writeln;
  Writeln('Testing TabStringBuilder:');
  ExecuteStringBuilderTests(myTabBuilder);

  Dispose(myTabBuilder, Done);
  writeln; writeln;
end.
```

```
Testing StringBuilder:
StringBuilder content: Hello World2023TRUE123456789


Testing TabStringBuilder:
StringBuilder content: Hello   W       o       r       l       d       2023    TRUE    12345678


Heap dump by heaptrc unit of C:\_data\fh-repos\2023SS_ADF\UE8\hu2\TestStringBuilder.exe
2 memory blocks allocated : 524/528
2 memory blocks freed     : 524/528
0 unfreed memory blocks : 0
True heap size : 98304 (112 used in System startup)
True free heap : 98192
```

```
program TestStringJoiner;

uses
  StringJoinerUnit;

procedure ExeuteStringJoinerTests;
```

```
var
  joiner: StringJoinerPtr;
begin
  // Create a StringJoiner with delimiter ","
  joiner := NewStringJoiner(',');

  // Add some strings
  joiner^.Add('Hello');
  joiner^.Add('World');
  joiner^.Add('!');
  joiner^.Add('How');
  joiner^.Add('are');
  joiner^.Add('');
  joiner^.Add('you');
  joiner^.Add('today');
  joiner^.Add('?');

  // Get and print the result
  Writeln('Result: ', joiner^.AsString);

  // Clean up memory
  Dispose(joiner, done);
end;

begin
  ExeuteStringJoinerTests;
end.
```

```
Result: Hello,World,!,How,are,,you,today,?
Heap dump by heaptrc unit of C:\_data\fh-repos\2023SS_ADF\UE8\hu2\TestStringJoiner.exe
2 memory blocks allocated : 272/280
2 memory blocks freed     : 272/280
0 unfreed memory blocks : 0
True heap size : 131072 (112 used in System startup)
True free heap : 130960
```