

☐ Gr. 1, Dr. D. Auer☐ Gr. 2, Dr. G. Kronberger

Gr. 3, Dr. H. Gruber

Name Andreas RoitherAufwand in h 5 h

Punkte _____

Kurzzeichen Tutor / Übungsleiter _____ / _____

0. Zahlen zur Basis 10 und 16**(0 Punkte;-)**

Pascal erlaubt die Angabe von ganzen Zahlen (Literalen) zur Basis 10 (Dezimalzahlen) und zur Basis 16 (Hexadezimalzahlen), indem für Hex-Zahlen das \$-Zeichen vor einer Folge von Hex-Ziffern (0 bis 9 und A bis F) angegeben wird. Hier zwei *Beispiele*, die dasselbe Resultat liefern:

```
WriteLn(17); (* 17 = 1*10 + 7 *)  
WriteLn($11); (* $11 = 1*16 + 1 = 17 *)
```

Nicht schlecht, aber das reicht Ihnen nicht: Sie möchten mit Zahlen zu einer beliebigen ganzzahligen Basis ≥ 2 arbeiten können.

1. Zahlen mit Basen von 2 bis 36**(4 + 4 + 4 Punkte)**

Bis zur Basis 36 geht das noch relativ einfach, da man die Ziffernfolge einer solchen Zahl durch eine Zeichenkette (vom Datentyp *STRING*) repräsentieren kann, indem man (je nach Basis) für die Werte der "Ziffern" von 0 bis 9 die Dezimalziffern und darüber die 26 Buchstaben des Alphabets mit den Ziffernwerten A = 10 bis Z = 35 verwendet.

a) Sie entwickeln erst einmal eine Pascal-Funktion

```
FUNCTION ValueOf(digits: STRING; base: INTEGER): INTEGER;
```

die eine Ziffernfolge *digits* zur Basis *base* in einen *INTEGER*-Wert umrechnet. Natürlich prüfen Sie auch, ob die Basis passt und die verwendeten Ziffern zur Basis passen. Sollte das nicht der Fall sein, liefert Ihre Funktion den Fehlercode -1.

Beispiele:

```
ValueOf('10001', 2) = 17  
ValueOf('17', 10) = 17  
ValueOf('11', 16) = 17  
ValueOf('AB', 10) = -1
```

b) Für die Umwandlung eines positiven *INTEGER*-Werts *value* in eine Ziffernfolge zu einer beliebigen Basis *base* – also für die Umkehrung der Funktionalität aus a) – entwickeln Sie eine Funktion

```
FUNCTION DigitsOf(value: INTEGER; base: INTEGER): STRING;
```

die für negative Werte in *value* als Ergebnis 'ERROR' liefert.

c) Interessant werden Zahlen aber erst, wenn man mit ihnen rechnen kann. Sie entwickeln deshalb vier Funktionen mit den Namen *Sum*, *Diff*, *Prod* und *Quot* jeweils mit der Schnittstelle

```
FUNCTION ...(d1: STRING; b1: INTEGER;  
            d2: STRING; b2: INTEGER): INTEGER;
```

so, dass Sie die vier Grundrechenoperationen auf beliebigen Zahlen (auch unterschiedlicher Basis) ausführen können. Nehmen Sie an, dass die Werte mit denen Sie rechnen und die Ergebnisse, die sich durch die Operationen ergeben, positiv sind und mit *INTEGER* repräsentiert werden können (deren Wertebereich also zwischen 0 und 32767 liegt).

2. Erzeugen von Balkendiagrammen – revisited**(4 Punkte)**

Mehr Wissen (z. B. über Felder) ermöglicht flexiblere Programme: Erweitern Sie Ihr Pascal-Programm zur Erzeugung von Balkendiagrammen aus der Übung 3 so, dass nicht mehr genau sechs, sondern eine beliebige Anzahl (zwischen 2 und 40) von Zahlen (alle im Bereich von 1 bis 10) behandelt werden können.

3. The Missing ... – tj, diesmal nicht *Link* sondern *Element***(8 Punkte)**

Gegeben sei eine Folge ganzer Zahlen mit n Elementen. Die Zahlen in der Folge sind unsortiert und entstammen dem Wertebereich 1 bis $n + 1$. Bis auf eine Zahl kommen alle anderen Elemente aus dem Wertebereich genau einmal darin vor. Hier ein *Beispiel* für eine solche Folge für $n = 4$, also mit 4 Elementen: 3, 2, 4, 5. Offensichtlich fehlt hier das Element 1.

Gesucht ist eine Pascal-Funktion *MissingElement*, die das fehlende Element einer solchen Zahlenfolge liefert.

Verwenden Sie folgende Deklarationen:

```
CONST
    max = 100;

TYPE
    IntArray = ARRAY [1 .. max] OF INTEGER;

FUNCTION MissingElement(a: IntArray; n: INTEGER): INTEGER;
```

Hinweis:

Natürlich kann man ein Hilfsfeld h mit max Elementen vom Datentyp *BOOLEAN* verwenden, wobei im ersten Schritt alle Elemente von 1 bis $n + 1$ mit *FALSE* initialisiert werden, im zweiten Schritt jedes Element in h , dessen Index in a vorkommt auf *TRUE* gesetzt wird und im dritten Schritt der Index jenes Elements gesucht wird, das noch den Wert *FALSE* enthält. – Es geht aber auch ohne Hilfsfeld und noch dazu viel schneller!

Übung 4

Aufgabe 1

Lösungsidee

Für das Umwandeln in andere Basen wird eine allgemeine Formel verwendet. Dazu wird ein String verwendet in dem alle erlaubten Zeichen sind, gleichzeitig wird die Position des Zeichens im String als Wertigkeit des Zeichens in der Basis 10 verwendet.

```

1 program base2to36;

3 (*String mit Characters die verwendet werden dürfen*)
const possibleCharacters : String = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';

5
6 (*Funktion für  $x^y$ *)
7 function powerFn (number, expo: Real): Real;
begin
9 powerFn := Exp(Expo*Ln(Number));
end;

11
12 (*Liefert Position des gesuchten Elementes in einem String zurück*)
13 function getPosofElement (e : char; s : string): Integer;
var i, count : Integer;
15 begin
    count := 0;

17
18     for i := 1 to Length(s) do
19         if e = s[i] then exit(count)
20         else count := count + 1;

21
22     getPosofElement := count;
23 end;

25 (*Ueberprueft ob ein Zeichen in einem String vorhanden ist*)
26 function stringContains(digits : string; c2 : char): Boolean;
var i : Integer;
27 begin
28     for i := 1 to Length(digits) do begin
29         if digits[i] = c2 then exit(True);
30     end;
31     stringContains := False;
32 end;

35 (*Gueltigkeitsueberpruefung ob für die jeweilige Basis die erlaubten Zeichen im
    String enthalten sind*)
function checkIfValid(digits: String; base: Integer; var count: Integer):
    Boolean;
37 var i : Integer;
begin
38     count := 0;
39     if (base >= 2) and (base <= 36) then
40     begin
41         for i := 1 + base to 36 do
42         begin
43             if stringContains(digits, possibleCharacters[i]) then exit(False);
44         end;
45         checkIfValid := True;

```

```

47   end
    else
49     checkIfValid := False;
end;

51  (*Liefert den Wert des String Inhaltes von beliebiger Basis in Basis 10 zurück
    *)
53  function ValueOf(digits: String; base: Integer): Integer;
var count, i, expo : Integer;
55  var value : Real;
begin
57    if checkIfValid(digits, base, count) then begin
        value := 0;
59        expo := 0;

61        for i:= Length(digits) downto 1 do begin
            value := value + getPosofElement(digits[i],possibleCharacters) * powerFn(
base,expo);
63            expo := expo + 1;
        end
65    end
    else
67        exit(-1);

69    ValueOf := Trunc(value);
end;

71  (*Liefert String mit Ziffernfolge für beliebige Basis*)
73  function DigitsOf(value: Integer; base: Integer): string;
var result : string;
75  begin
    result := '';
77    if value < 0 then DigitsOf := 'ERROR'
    else
79        begin
            while value > 0 do
81                begin
                    (*an den String wird ein Zeichen mit einer bestimmten Wertigkeit in der
Basis 10 angehängt*)
83                    if value mod base > 0 then result := Concat(possibleCharacters[(value mod
base)+1], result);

85                    value := value DIV base;
                end;
87            end;
            DigitsOf := result;
89        end;


91  function sum(d1: STRING; b1: INTEGER; d2: STRING; b2: INTEGER): INTEGER;
begin
93    sum := ValueOf(d1,b1) + ValueOf(d2,b2);
end;

95  function diff(d1: STRING; b1: INTEGER; d2: STRING; b2: INTEGER): INTEGER;
begin
97    diff := ValueOf(d1,b1) - ValueOf(d2,b2);
99  end;

```

```
101 function prod(d1: STRING; b1: INTEGER; d2: STRING; b2: INTEGER): INTEGER;
    begin
103     prod := ValueOf(d1,b1) * ValueOf(d2,b2);
    end;
105 function quot(d1: STRING; b1: INTEGER; d2: STRING; b2: INTEGER): INTEGER;
    begin
107     quot := ValueOf(d1,b1) div ValueOf(d2,b2);
    end;
109
    begin
111     WriteLn('— base2to36 —');
        WriteLn('ValueOf');
113     WriteLn('23672 Basis 10');
        WriteLn(ValueOf('23672',10));
115     WriteLn('4DF8 Basis 17');
        WriteLn(ValueOf('4DF8',17));
117     WriteLn('1LH5 Basis 23');
        WriteLn(ValueOf('1LH5',23));
119
        WriteLn('DigitsOf');
121     WriteLn('23672 Basis 28');
        WriteLn(DigitsOf(23672,28));
123
        WriteLn('diff of 1234 in Basis 12');
125     WriteLn(diff('1234',12,'1234',12));
127
        WriteLn('prod of 1234 in Basis 12');
        WriteLn(prod('1234',12,'1234',12));
129
        WriteLn('Sum of 1234 in Basis 12');
131     WriteLn(sum('1234',12,'1234',12));
133
        WriteLn('quot of 1234 in Basis 12');
        WriteLn(quot('1234',12,'1234',12));
135 end.
```

base2to36.pas



```
C:\WINDOWS\system32\cmd.exe

C:\Users\andir\Google Drive\Hagenberg\1. Semester\ADE\Uebung\Uebung04>base2to36.exe
-- base2to36 --
ValueOf
23672 Basis 10
23672
4DF8 Basis 17
23672
1LH5 Basis 23
23672
DigitsOf
23672 Basis 28
125C
diff of 1234 in Basis 12
0
prod of 1234 in Basis 12
-32704
Sum of 1234 in Basis 12
4112
quot of 1234 in Basis 12
1

C:\Users\andir\Google Drive\Hagenberg\1. Semester\ADE\Uebung\Uebung04>
```

Abbildung 1: Testfälle Basis 2 bis 36

Testfälle

Zum Testen werden verschiedene Basen verwendet, dabei soll immer dieselbe Zahl ausgegeben werden. Anschließend wird noch ein Zahl in der 10 Basis in der 28er Basis ausgegeben. Zum Schluss werden Diff, Sum, Prod und quot ausgegeben (mit 1234 in Basis 12), wobei bei Prod eine negative Zahl ausgegeben wird weil der Bereich überschritten wird. Dasselbe kann auch bei Sum passieren wenn zu hohe Zahlen verwendet werden.

Aufgabe 2

Lösungsidee

Beim Balkendiagramm redone wird ein String verwendet an dem immer die jeweilige Zeile für einen Politiker angehängt und anschließend ausgegeben wird.

```

1 program balkendiagramm_redone;

3 (*Integer auf oder abrunden*)
function round(num: Integer) : Integer;
5 var temp, temp2 : Integer;
begin
7   temp := abs(num);

9   if num mod 10 >= 5 then temp2 := 1 else temp2 := 0;

11  round := (temp div 10) + temp2;
end;

13 (*Zeile fuer einen Politiker mit String*)
15 function printGraph(a, b, pol_count : Integer) : String;
var i: Integer;
17 var s: String;
begin
19   s := '';
   str(pol_count, s);
21   s := Concat(s, ':');

23   for i := 1 to 10 - round(a) do s := Concat(s, ' ');
   for i := 1 to round(a) do s := Concat(s, 'X');

25   s := Concat(s, '|');

27   for i := 1 to round(b) do s := Concat(s, 'X');
   for i := round(b) to 10 do s := Concat(s, ' ');

31   (*Anfügen eines NewLine Characters an den String*)
   s := Concat(s, chr(10));

33   printGraph := s;
35 end;

37 var anzahl, i, neg, pos : Integer;
var result : string;
39 begin
   result := '';
41   anzahl := 0;

43   WriteLn('— Balkendiagramm Redone —');
   Write('Geben Sie die Anzahl der Politiker ein: ');
45   Read(anzahl);

47   if( anzahl >= 2 ) and (anzahl <= 40) then
begin
49     (*Für beliebige Anzahl Politiker wird eingelesen und die Werte, falls gü
ltig,
       verarbeitet und in einem String gespeichert*)
       for i := 1 to anzahl do

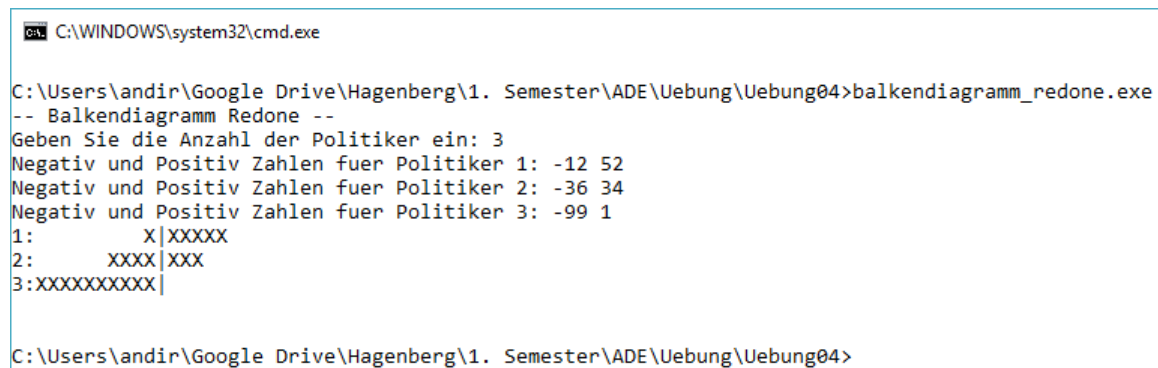
```

```

begin
53   Write('Negativ und Positiv Zahlen fuer Politiker ',i,': ');
      ReadLn(neg, pos);
55   if abs(neg) + pos <= 100 then
      result := Concat(result, printGraph(abs(neg),pos,i))
57   else WriteLn('Summe groesser 100 fuer Politiker ',i);
      end;
59   WriteLn(result);
      end
61   else
      WriteLn('Keine gueltige Angabe');
63 end.

```

balkendiagramm_redone.pas



```

C:\WINDOWS\system32\cmd.exe

C:\Users\andir\Google Drive\Hagenberg\1. Semester\ADE\Uebung\Uebung04>balkendiagramm_redone.exe
-- Balkendiagramm Redone --
Geben Sie die Anzahl der Politiker ein: 3
Negativ und Positiv Zahlen fuer Politiker 1: -12 52
Negativ und Positiv Zahlen fuer Politiker 2: -36 34
Negativ und Positiv Zahlen fuer Politiker 3: -99 1
1:      X|XXXXX
2:     XXXX|XXX
3:XXXXXXXXXX|

C:\Users\andir\Google Drive\Hagenberg\1. Semester\ADE\Uebung\Uebung04>

```

Abbildung 2: Testfälle Balkendiagramm

Testfälle

Es werden drei Politiker eingegeben mit unterschiedlichen Zahlen.

Aufgabe 3

Lösungsidee

Zum finden der fehlenden Zahl wird eine Folge von Zahlen aufsummiert und dann von einer Check Summe abgezogen. Der Rest ist das fehlende Element.

```

1 program missingelement;

3 const
  max = 6;

5
6
7 type
  IntArray = array [1 .. max] OF integer;

9 function MissingElement(a: IntArray; n: integer): integer;
var i, array_sum, check_sum : integer;
11 begin
    check_sum := 0;
13    array_sum := 0;

15    (*Summe aus array bilden mit einer check summe*)
    for i := 1 to Length(a) do
17        begin
            check_sum := check_sum + i;
19            array_sum := array_sum + a[i];
        end;

21    (*check_sum - array_sum ergibt das fehlende Element*)
23    check_sum := check_sum + i+1;
    MissingElement := (check_sum - array_sum);
25 end;

27 var test_array : IntArray = (5,4,2,1,3,6);
    var test_array2: IntArray = (1,3,5,4,6,7);
29 var i : Integer;
    begin
31        WriteLn('— MissingElement —');
        Write('Array: ');
33        for i := 1 to Length(test_array) do
            Write(test_array[i], ' ');

35        WriteLn(#13#10, 'MissingElement: ', MissingElement(test_array, max));

37        Write('Array: ');
39        for i := 1 to Length(test_array2) do
            Write(test_array2[i], ' ');
41        WriteLn(#13#10, 'MissingElement: ', MissingElement(test_array2, max));
    end.

```

missingelement.pas



```
C:\WINDOWS\system32\cmd.exe

C:\Users\andir\Google Drive\Hagenberg\1. Semester\ADE\Uebung\Uebung04>missingelement.exe
-- MissingElement --
Array: 5 4 2 1 3 6
MissingElement: 7
Array: 1 3 5 4 6 7
MissingElement: 2

C:\Users\andir\Google Drive\Hagenberg\1. Semester\ADE\Uebung\Uebung04>
```

Abbildung 3: Testfall missingelement

Testfall

Ein Array mit einer Test Größe von 7 wird mit Zahlen in beliebiger Reihenfolge gefüllt und das fehlende Element wird ausgegeben. Dasselbe wird mit einem zweiten Array, mit einem anderen fehlenden Element, gemacht.