Abgabetermin: Mi in der KW 45

☐ **Gr. 1,** Dr. D. Auer
☐ **Gr. 2,** Dr. G. Kronberger **Name** _____ **Aufwand in h** _____
☐ **Gr. 3,** Dr. H. Gruber

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* ... – tja, 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!