8.2 Operationen und Vergleiche für Zahlen

Bei den Operationen mit Zahlen als Operanden sind einerseits die Typen der Operanden zu beachten und andererseits der Typ des Ergebnisses der Operation. Bei der Division von zwei reellen Zahlen ist der Quotient stets vom Typ Float. Die beiden Zahlen iZahl1 und iZahl2 dagegen müssen bei der Operation *iZahl1 MOD iZahl2* ganze Zahlen sein.

8.2.1 Arithmetische Operatoren

Beachten Sie, dass es drei verschiedene Operationen für die Division von zwei Zahlen mit unterschiedlichen Operatoren (/, MOD ≡ %, DIV ≡ \) gibt.

Operation	Beschreibung
- Zahl	Berechnet die entgegengesetzte Zahl, wobei Null das Entgegengesetzte von sich selbst ist; das Vorzeichen wird invertiert.
Zahl1 + Zahl2	Es werden zwei Zahlen addiert.
Zahl1 - Zahl2	Es werden zwei Zahlen subtrahiert.
Zahl1 * Zahl2	Es werden zwei Zahlen multipliziert.
Basis ^ Exponent	Berechnet wird der Potenzwert zur Basis b mit dem Exponenten e für reelle Zahlen b und e. Ein Beispiel ist b^e = 4^3 = 64.
Zahl1 / Zahl2	Es werden zwei Zahlen dividiert. Der Quotient ist vom Typ *Float*. Ein *DivisionByZero-Fehler* tritt auf, wenn der Divisor Null ist.
iZahl1 MOD iZahl2 oder iZahl1 % iZahl2	Berechnet den Rest des Quotienten von zwei ganzen Zahlen. Das Ergebnis der Operation ist eine ganze Zahl. Ein *DivisionByZero-Fehler* tritt auf, wenn der Divisor Null ist.
iZahl1 DIV iZahl2 oder iZahl1 \ iZahl2	Berechnet den Quotienten von zwei ganzen Zahlen – nach unten abgerundet oder den ganzzahligen Teil des Quotienten *iZahl1 / iZahl2*. Das Ergebnis der Operation ist eine ganze Zahl. Ein *DivisionByZero-Fehler* tritt auf, wenn der Divisor Null ist.

Tabelle 8.2.1.1: Arithmetische Operatoren

In vielen Fällen werden nicht nur die vorgestellten Grund-Operationen eingesetzt, sondern komplexe mathematische Berechnungen ausgeführt. Bei komplexen arithmetischen Operationen unterscheidet man nach *Priorität* und *Assoziativität*, wenn mindestens 2 Operatoren in einer Berechnungsanweisung vorhanden sind. Priorität beschreibt die Rangfolge, mit der mathematische Terme miteinander verknüpft werden und Assoziativität die Priorität, wenn mindestens zwei gleichrangige Operatoren vorhanden sind. Im Kapitel 8.5 *'Rangfolge von Operationen*' erhalten Sie detaillierte Informationen zu *Assoziativität* und *Priorität* von Operationen. Interessant ist der Operator ^, der den Potenzwert zu einer Basis b und dem Exponenten e mit b,e ∈ ℝ berechnet:

```
2^3 =  8
2^(-3) =  0,125
2^3.44 =  10,8528346195814
(-2)^3 =  -8
-2^3 =  -8
(-2)^(-3) =  -0,125
-2^-3 =  -0,125
3.2^(-0.77) =  0,40835183987713
3.2^-0.77 =  0,40835183987713
SQR(5)^CBR(-22.7) =  0,10243569625776
1^0.88 =  1
0^3 =  0
3.4567^0 =  1
```

8.2.1.1 Operator MOD oder %

Für die Berechnung des Restes bei der Division von zwei ganzen Zahlen verwendet man den Operator MOD oder %. Bei der Ganzzahl-Division mit Rest gibt es unterschiedliche Auffassungen, wie man den Ganzzahl-Quotienten sowie den Rest bei unterschiedlichen Zahlenbereichen berechnet. Die Abbildung {Z,N} → R* ordnet jedem Paar ganzer Zahlen den Divisionsrest R* der Division Z/N zu. Da die

Abbildung eindeutig ist, handelt es sich um eine Funktion, die den Funktionsnamen *modulo* oder abgekürzt mod oder % trägt. Die Funktion *modulo* wird in Gambas als Operator MOD in mathematischen Berechnungen eingesetzt. Nach Definition gilt: z MOD n = z - |_z/n_| * n mit dem Symbol |_z/n_| als *Abrundungsfunktion oder Gauss-Klammer*. Für eine reelle Zahl q ist |_q_| die größte ganze Zahl, die kleiner oder gleich q ist. Diese Vorbemerkungen sind notwendig, weil Gambas diesem sogenannten mathematischen Ansatz NICHT folgt, wie Sie der Hilfe zu Gambas entnehmen können. Wenn Sie jedoch den mathematischen Ansatz bevorzugen, dann hilft Ihnen die folgende Funktion:

```
Public Function myMOD(iA As Integer, iM As Integer) As Integer
  Return iA - (Int(iA / iM)) * iM
End ' myMOD(iA As Integer, iM As Integer)
```

Im folgenden Abschnitt erfahren Sie, wie Sie den Unterschied zwischen dem mathematischen Ansatz und der Definition von MOD in der Sprache Gambas ermitteln können:

```
Print "11 MOD 4 = ";; 11 Mod 4
Print "11 myMOD 4 = ";; myMOD(11, 4)
Print "11 MOD -4 = ";; 11 Mod -4
Print "11 myMOD -4 = ";; myMOD(11, -4)
Print "-11 MOD 4 = ";; -11 Mod 4
Print "-11 myMOD 4 = ";; myMOD(-11, 4)
Print "-11 MOD -4 = ";; -11 Mod -4
Print "-11 myMOD -4 = ";; myMOD(-11, -4)
```

```
11 MOD 4 =  3
11 myMOD 4 =  3

11 MOD -4 =  3
11 myMOD -4 =  -1

-11 MOD 4 =  -3
-11 myMOD 4 =  1

-11 MOD -4 =  -3
-11 myMOD -4 =  -3
```

8.2.1.2 Operator DIV oder \

Obgleich in der Definition zum Operator DIV "Calculates the quotient of two Integer numbers, rounding down. Berechnet den Quotient aus zwei ganzen Zahlen und rundet diesen nach unten ab" steht, gibt es bei Berechnungen mit dem Operator DIV Fehler, weil die o.a. Definition nicht konsequent umgesetzt wird. Durch den Einsatz der Funktion *myDIV(..)* können Sie Fehler vermeiden:

```
Public Function myDIV(iA As Integer, iM As Integer) As Integer
  Return Int(iA / iM)
End ' myDIV(iA As Integer, iM As Integer)
```

```
20:7 =  2,85714285714286
20 DIV 7 =  2
20 myDIV 7 =  2

20:(-7) =  -2,85714285714286
20 DIV -7 =  -2
20 myDIV -7 =  -3

(-20):7 =  -2,85714285714286
-20 DIV 7 =  -2
-20 myDIV 7 =  -3

(-20):(-7) =  2,85714285714286
-20 DIV -7 =  2
-20 myDIV -7 =  2
```

Der folgende Quelltext zeigt den Einsatz der Operatoren DIV und MOD:

```
[1]   Public Sub btnM2H_Click()
[2]     Dim iSummeMinuten, iStunden, iRestMinuten As Integer
[3]     Dim skommentar As String
[4]
```

```
[5]      iSummeMinuten = 444
[6]      iStunden = iSummeMinuten Div 60
[7]      iRestMinuten = iSummeMinuten Mod 60
[8]
[9]      If iRestMinuten Mod 2 = 0 Then
[10]        sKommentar = "Restminuten = " & Str(iRestMinuten) & " ---> gerade Zahl"
[11]     Else
[12]        sKommentar = "Restminuten = " & Str(iRestMinuten) & " ---> ungerade Zahl"
[13]     Endif
[14]     Print "Stunden = ";; iStunden
[15]     Print "RestMinuten = ";; iRestMinuten
[16]     Print sKommentar
[17]
[18]  End ' btnM2H_Click()
```

Besonders der Vergleich in der Zeile 9 wird oft benutzt, um festzustellen, ob eine ganze Zahl gerade (zahl mod 2 = 0) oder ungerade ist. Davon wird auch im nächsten Quelltextausschnitt Gebrauch gemacht, damit die Zeilen in einer GridView unterschiedlich eingefärbt werden, um eine bessere Lesbarkeit zu erzielen. Achtung: Für die 1. angezeigte Zeile gilt Row = 0 für die GridView:

```
[1]   Public Sub TableView1_Data(Row As Integer, Column As Integer)
[2]     TableView1.Data.Text = aMatrix[Row][Column]
[3]     If Row MOD 2 = 0 Then
[4]        TableView1.Data.Background = Color.RGB(224, 224, 224) ' hellgrau
[5]     Endif ' MOD 2 = 0?
[6]   End ' TableView1_Data(..)
```

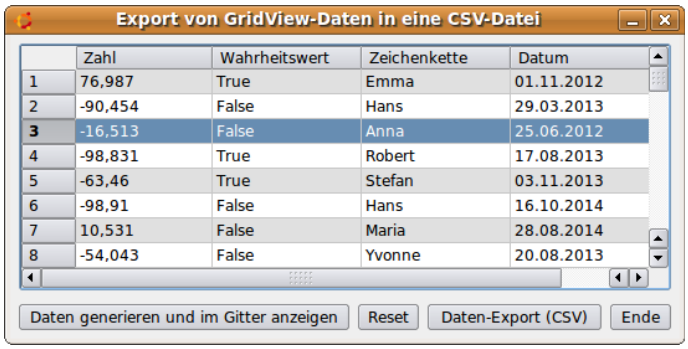

Abbildung 8.2.1.2.1 Alternative Färbung der Zeilen in einer Gitteransicht

8.2.2 Vergleichsoperatoren für Zahlen

Für den Vergleich von zwei Zahlen stehen Ihnen 6 Vergleichsoperatoren zur Verfügung:

Vergleich	Beschreibung
Zahl1 = Zahl2 ?	Es wird TRUE zurückgegeben, wenn die zwei Zahlen gleich sind.
Zahl1 <> Zahl2 ?	Es wird TRUE zurückgegeben, wenn die zwei Zahlen nicht gleich sind.
Zahl1 < Zahl2 ?	Es wird TRUE zurückgegeben, wenn die Zahl1 kleiner ist als Zahl2.
Zahl1 > Zahl2 ?	Es wird TRUE zurückgegeben, wenn die Zahl1 größer ist als Zahl2.
Zahl1 <= Zahl2 ?	Es wird TRUE zurückgegeben, wenn die Zahl1 kleiner oder gleich Zahl2 ist.
Zahl1 >= Zahl2 ?	Es wird TRUE zurückgegeben, wenn die Zahl1 größer oder gleich Zahl2 ist.

Tabelle 8.2.2.1: Vergleichsoperatoren für Zahlen

Wenn das Ergebnis eines Vergleichs von 2 Zahlen *nicht* einer boolschen Variablen sondern einer Integer-Variablen zugewiesen wird, dann ist das Ergebnis entweder -1 (True) oder 0 (False).

Mit besonderer Sorgfalt sind Vergleiche zu überdenken, in denen die *Gleichheit zweier reeller Zahlen* festgestellt werden soll *oder* geprüft wird, ob eine Variable vom Typ *Float* den Wert Null hat. Gute Erfahrungen konnten mit folgenden Anweisungen zum Beispiel bei der Berechnung von Wertetabellen gewonnen werden. Zuerst wird geprüft, ob der Funktionswert y = f(x) zu einem Argument x berechnet

werden kann. Dann wird das Argument auf Null gesetzt, wenn das Argument hinreichend klein ist. Abschließend wird der Funktionswert y = f(x) geprüft. In ähnlicher Weise können Sie auch bei den Werten für x und y vorgehen, um zum Beispiel Überläufe zu erkennen oder um (hinreichend) große Werte für das Argument oder den Funktionswert auf aufgaben-adäquate Werte zu verringern:

```
[1]    IF aValuePair.Valid = True THEN
[2]       IF Abs(x) < 1E-8 THEN aValuePair.x = 0
[3]       IF Abs(aValuePair.y) < 1E-6 THEN aValuePair.y = 0
[4]    ENDIF
```

Das ist der Inhalt der Klasse *ValuePair.class*:

```
' Gambas class file

' Diese Klasse ist eine Datenstruktur ohne eigene Methoden.
' Sie repräsentiert u.a. ein xy-Wertepaar. Die Valid-Variable muss
' von außen gesetzt werden und gibt an, ob das xy-Wertepaar gültig ist.
' Tag ist eine universell verwendbare Variable. Sie dient der Parser-Klasse zum Speichern
' der Fehlermeldung bei ungültigen Objekten.

Public x As Float
Public y As Float

Public Valid As Boolean
Public Tag As Variant
```