7.0 Datentypen

Datentypen werden dann benötigt, wenn Sie Daten – als an Zeichen gebundene Informationen – in einer Variablen abspeichern wollen. Datentypen beschreiben:

- welche Art von Daten in einer Variablen gespeichert werden können,
- den Wertebereich, den ein bestimmter Datentyp abdeckt,
- die zulässigen Operationen für den ausgewählten Datentyp sowie
- die Repräsentation der Daten im Speicher, die darüber entscheidet, wie viel Speicher für eine Variable eines bestimmten Typs zur Verfügung gestellt wird.

In Gambas gibt es native Datentypen. Dazu zählen zum Beispiel Datentypen mit den Namen (Schlüsselwort) Integer, Byte, String oder Float. Eine Übersicht zu den nativen Datentypen finden Sie in der Tabelle 7.0.1.

Die Syntax einer Struktur (Struct) wurde in Gambas 3 eingeführt → Kapitel 7.2. Diese Strukturen erlauben Ihnen die Zusammenfassung logisch relevanter informatischer Einheiten (sowohl Datentypen als auch Klassen oder Objekte) zu einer neuen Einheit – einer Struktur. Diese Struktur besitzt jedoch keine Methoden oder Events.

Von besonderer Wichtigkeit sind in Gambas als objektorientierter Sprache die Klassen. Alles, was kein Datentyp oder Struktur ist oder ausführbaren Code kennzeichnet, ist eine Klasse. Sie zeichnen sich dadurch aus, dass sie eine logische Kapselung von Eigenschaften, Methoden und Events sind. Eine Instanz einer Klasse heißt Objekt.

Zusammenfassung

- Gambas besitzt native Datentypen, die ihrem Typ entsprechend ihre Wert-Information tragen. Es wird geprüft, ob der einer Variablen oder Konstanten zugewiesene Wert mit dem deklarierten Datentyp kollidiert.
- Die Strukturen sind eine Bündelung von verschiedenen, aber zusammengehörigen Wert-Informationen.
- Klassen besitzen zusätzlich zu den Informationen (Eigenschaften) auch Methoden, um diese zu modifizieren sowie Ereignisse (Events), um interaktiv zu agieren.

Somit nehmen Strukturen eine *vermittelnde* Stellung zwischen den nativen Datentypen und den abstrakten Klassen (konkret und abstrakt im Sinne ihrer Definition) ein.

Datentyp	Beschreibung	Default	Speichergröße
Boolean	TRUE oder FALSE	FALSE	1 Byte
Byte	0 … 255	0	1 Byte
Short	-32.768 … +32.767	0	2 Bytes
Integer	-2.147.483.648 … +2.147.483.647	0	4 Bytes
Long	-9.223.372.036.854.775.808 ... +9.223.372.036.854.775.807	0	8 Bytes
Single	Wie der *float*-Datentyp in C	0.0	4 Bytes
Float	Wie der *double*-Datentyp in C	0.0	8 Bytes
Date	Datum und Zeit, jeweils in einem Integer-Wert abgelegt	NULL	8 Bytes
String	Eine Zeichenkette mit variabler Länge	NULL	4 Bytes
Variant	Jeder Datentyp	NULL	12 Bytes
Object	Anonyme Referenz auf ein Objekt	NULL	4 Bytes
Pointer	Zeiger auf eine Speicheradresse	0	4 Bytes auf 32-Bit-Systemen und 8 Bytes auf 64-Bit-Systemen

Tabelle 7.0.1: Übersicht zu den nativen Datentypen in Gambas

7.0.1 Array-Datentyp

Die nativen Datentypen haben einen zugeordneten Array-Datentyp, dessen Name mit dem Namen des nativen Datentyps korrespondiert und dem eckige Klammern folgen:

```
Boolean[], Byte[], Short[], Integer[], Single[], Float[], String[], Datum[], Variant[], Pointer[], Object[]
```

Die Beschreibung von Arrays finden Sie im Kapitel 20.12.

7.0.2 Datentypen-Funktionen

Die folgenden Funktionen können Sie nutzen, um den *Daten-Typ* zu ermitteln. Als Argument wird den Datentypen-Funktionen ein *String* oder bei SizeOf() ein *Datentyp* oder bei TypeOf() ein *Ausdruck* übergeben. Der Funktionswert ist entweder True oder False oder ein Speicherwert oder ein Datentyp.

Funktion	Beschreibung
IsBoolean*(string)*	Ist True, wenn string="True" oder string="False" ist (Vergleich nicht case-sensitiv)
IsDate*(string)*	Ist True, wenn *string* sicher als Datum interpretiert werden kann
IsFloat*(string)*	Ist True, wenn *string* sicher als Float-Zahl interpretiert werden kann
IsInteger*(string)*	Ist True, wenn *string* sicher als ganze Zahl interpretiert werden kann
IsLong*(string)*	Ist True, wenn *string* sicher als Long-Integer-Zahl interpretiert werden kann
IsNumber*(string)*	Ist True, wenn *string* sicher als Zahl interpretiert werden kann
IsNull*(string)*	Ist True, wenn *string* sicher NULL ist

Tabelle 7.0.2.1: Übersicht zu den Datentyp-Funktionen 1

Funktion	Beschreibung
SizeOf*(datentyp)*	Gibt den Speicher an, der von einem bestimmten Datentyp verwendet wird.
TypeOf*(ausdruck)*	Gibt den Datentyp eines Ausdrucks zurück.

Tabelle 7.0.2.2: Übersicht zu den Datentyp-Funktionen 2

Der Funktionswert von *SizeOf(datentyp)* oder *TypeOf(ausdruck)* ist einer der folgenden Konstanten:

Datentyp/Ausdruck	Rückgabe-Wert (numerisch)	Rückgabe-Wert (Konstante)
Boolean	1	gb.Boolean
Byte	2	gb.Byte
Short	3	gb.Short
Integer	4	gb.Integer
Long	5	gb.Long
Single	6	gb.Single
Float	7	gb.Float
Date	8	gb.Date
String	9	gb.String
Pointer	11	gb.Pointer
Variant	12	gb.Variant
Function	13	gb.Function
Class	14	gb.Class
Object	16	gb.Object

Tabelle 7.0.2.3: Übersicht zu den Funktionswerten von TypeOf() und SizeOf()

Hinweise:

- Beachten Sie, dass TypeOf(*NULL*) den numerischen Wert 15 oder *gb.Null* zurück gibt.
- Die Funktion IsNull(string) liefert für folgende Argumente den Funktionswert True:
  - string ist eine NULL-Konstante
  - string ist eine NULL-Objektreferenz
  - Die Länge von string ist 0
  - string repräsentiert ein Null-Datum
  - string repräsentiert eine nicht initialisierte Variable vom Typ Variant

7.0.3 Konvertierung von Datentypen

Gambas stellt Funktionen bereit, um Datentypen zu konvertieren. Informationen zu diesen Funktionen erhalten Sie im Kapitel *9.7 Konvertierungsfunktionen*.

Gambas nimmt auch automatisch Konvertierungen von Datentypen vor. Zum Beispiel ist "2" ein String, kann aber auch als ganze Zahl 2 (Integer) aufgefasst und so nach einer Konvertierung "on the fly" verwendet werden, wie die folgenden Beispiele zeigen:

```
Public Sub OnTheFly_Click()
  Print "2" + 2 ' Integer-Zahl
  Print "Kann '2' als Integer-Zahl interpretiert werden? ---> ";; IsInteger("2")
  Print "Type_1 = ";; TypeOf("2" + 2) ' 7 => Float
' String + Integer
' String -> Integer => 2 + 2 = 4
  Print "2" & 2 ' Zeichenkette
  Print "Type_2 = ";; TypeOf("2" & 2) ' 9 => String
' String & Integer
' Integer -> String => "2" & "2" = "22"
  Print 2.1 * ("2" & 2) ' Gleitkomma-Zahl
  Print "Type_3 =  ";; TypeOf(2.1 * ("2" & 2)) ' 7 => Float
' Float * (String & Integer)
' Integer -> String => "2" & "2" = "22"
' Float * String => 2.1 * "22"
' String -> Float => 2.1 * 22.0 = 46.2
End ' OnTheFly()
```

Das sind die Ausgaben in der Konsole:

```
4
Kann '2' als Integer-Zahl interpretiert werden? --->  True
Type_1 =  7
22
Type_2 =  9
46,2
Type_3 =   7
```