

Arrays

und Ihre Anwendung in eigerScript®

Autor: Adrian Weitnauer¹
Erstellt: 21.03.2011

■ Inhalt

Inhalt	1
Begriff und Deklaration	2
Deklaration durch Angabe der Anzahl Speicherzellen.....	2
Deklaration durch Angaben der Indexgrenzen	3
Mehrdimensionale Arrays.....	3
Speicherbedarf.....	4
Einschränkungen	4
Zugriff auf Arrays	5
Schreibender Zugriff	5
Lesender Zugriff.....	5
Abschliessende Bemerkungen.....	5

¹ Adrian Weitnauer ist Inhaber des Ingenieurbüros weitnauer MESSTECHNIK, CH-8752 Näfels, und Entwickler der Entwicklungsumgebung eigerStudio im Auftrag der S-TEC electronics AG



■ Begriff und Deklaration

Arrays sind mehrdimensionale Speicherfelder, die eine Anzahl des gleichen Variablentyps umfassen. eigerScript unterstützt Arrays aller numerischen Datentypen.

Die Deklaration einer Variablen in eigerScript ist bekanntlich wie folgt:

```
INTEGER a = 0
```

Arrays werden deklariert, indem entweder die Grenzen jeder Dimension oder die Anzahl Speicherzellen jeder Dimension angegeben werden. Die Deklarationsformen können beliebig gemischt werden.

Deklaration durch Angabe der Anzahl Speicherzellen

Direkt auf die Angabe des Variablentyps folgt in eckigen Klammern ein konstanter, ganzzahliger Ausdruck, der die Anzahl der Speicherzellen definiert. Anschliessend folgt der Variablenname (Insofern ist die Deklaration recht ähnlich zum bekannten STRING-Typ):

```
INTEGER [64] arrayA
```

Im Beispiel wird so ein eindimensionaler Vektor mit 64 Zellen vom Typ INTEGER im Arbeitsspeicher angelegt. Adressiert werden die Zellen beginnend von Null bis zur Anzahl minus 1: Also 0 bis 63 im Beispiel.

Allgemein:

```
VARIABLENTYP [n] variablenName
```

Der Vektor enthält n Zellen, deren Adressen von 0 bis n-1 laufen.

Deklaration durch Angaben der Indexgrenzen

Ähnlich wie oben folgt auch hier auf die Angabe des Variablentyps in eckigen Klammern die Angabe der Arraygrenzen. Dies sind nun zwei konstante ganzzahlige Ausdrücke, die durch zwei Punkte getrennt werden:

```
INTEGER [0..127] arrayB
```

Im Beispiel wird ein eindimensionaler Vektor mit 128 Zellen vom Typ INTEGER angelegt, dessen Zellen von 0 bis 127 adressiert werden.

Allgemein:

```
VARIABLENTYP [m..n] variablenName
```

Der Vektor enthält $n-m+1$ Zellen, deren Adressen von m bis n laufen. Selbstverständlich dürfen die Grenzen auch negative Zahlen enthalten, solange n grösser ist als m .

Mehrdimensionale Arrays

Arrays können beinahe beliebig viele Dimensionen enthalten. Allerdings gilt es zu bedenken, dass die Grösse solcher Speicherstrukturen schnell ansteigt und deshalb durch den verfügbaren Arbeitsspeicher begrenzt wird.

```
INTEGER [0..123, 0 .. 65] arrayC
```

Das obenstehende Beispiel erzeugt ein zweidimensionales Array vom Typ INTEGER mit den Indizes 0 bis 123 und 0 bis 65, enthält also $124 * 66 = 8184$ Felder.

```
INTEGER [-20..-5, 25..125] arrayD
```

Das obenstehende Beispiel erzeugt ein zweidimensionales Array vom Typ INTEGER mit den Indizes -20 bis -5 und 25 bis 125, enthält also $-5-(-20)+1=16$ mal $125-25+1=101$, also $16 * 101 = 1616$ Felder.

```
SINGLE [4, 4, 4] quader
```

Das obenstehende Beispiel erzeugt ein dreidimensionales Array vom Typ SINGLE, das insgesamt $5 * 5 * 5 = 125$ Felder enthält.

```
SINGLE [4, 4, 4, 4] hyperQuader = 0
```

Dieses letzte Beispiel erzeugt ein vierdimensionales Array vom Typ SINGLE, das insgesamt $5 * 5 * 5 * 5 = 625$ Felder umfasst. Da eine Variable vom Typ SINGLE 4 Bytes enthält, werden mit dieser Deklaration 2500 Bytes Nutzdatenspeicher angefordert.

Speicherbedarf

Der Speicherbedarf von Arrays lässt sich ermitteln durch Berechnung der Nutzdaten wie oben dargestellt: Die Anzahl Speicherzellen aller Dimensionen werden miteinander multipliziert, was die totale Anzahl Speicherzellen ergibt. Dieses Ergebnis wird nun mit dem Speicherbedarf des Variablentyps multipliziert.

Jedes Array besitzt einen Header (versteckter „Vorspann“ im Speicher), der für gewisse Betriebssystemoperationen Informationen bereithält. Die Headergrösse umfasst 15 Bytes plus 8 Bytes für jede Dimension.

Einschränkungen

Im Gegensatz zu elementaren Variablen können Arrays nicht initialisiert werden. Ausserdem kennt eigerScript zurzeit keine Array-Konstanten. Weiter kennt eigerScript keine abgekürzten Zuweisungsbefehle; somit müssen Zuweisungen streng zellenbasiert erfolgen.

Zum Beispiel muss zur Zuweisung eines Arrays zu einem anderen jede Quellzelle einzeln zu einer Zielzelle zugewiesen werden, was am besten mit einer FOR-Schleife realisiert wird.

Zugriff auf Arrays

Grundsätzlich können Arrays in beliebigen numerischen Ausdrücken verwendet werden. Die Adressierung geschieht dabei wie bei der Deklaration mit eckigen Klammern.

Die virtuelle Maschine stellt sicher, dass Zugriffsverletzungen zur Laufzeit aufgefangen werden und die Maschine kontrolliert zurückgesetzt wird.

Schreibender Zugriff

Das folgende Beispiel zeigt, wie in zwei geschachtelten Schleifen ein Array gefüllt wird.

```
FOR i := 0 TO 3
  FOR j := 0 TO 4
    arrayC[i,j] := (i+1) * (j+1)
  NEXT
NEXT
```

Lesender Zugriff

Der lesende Zugriff erfolgt entsprechend. Das nächste Beispiel zeigt ein vierdimensionales Zahlenarray, dessen Inhalte zu einer Gesamtsumme aufaddiert werden.

```
summe := 0
FOR i := 0 TO 4
  FOR j := 0 TO 4
    FOR k := 0 TO 4
      FOR l := 0 TO 4
        summe := summe + hyper[i, j, k, l]
      NEXT
    NEXT
  NEXT
NEXT
```

Abschliessende Bemerkungen

Arrays werden ab der Version eigerStudio 1.0 unterstützt. Ein laufender Ausbau ist noch im Gange. Vorerst muss mit der notwendigen Sorgfalt programmiert werden, weil insbesondere die Typenprüfung in der Analyse von mathematischen Ausdrücken (Stackmaschine) noch nicht abschliessend freigegeben wurde.

Auf der Download-Seite von www.eigergraphics.com finden Sie übrigens stets die neuste Version dieses Dokuments.