# Arrays

Natural unterstützt die Verarbeitung von sogenannten Arrays. Arrays sind mehrdimensionale Tabellen, d.h. zwei oder mehr logisch verwandte Elemente, die unter einem gemeinsamen Namen definiert werden.

Arrays können aus einzelnen Datenelementen mit mehreren Dimensionen bestehen oder aus hierarchischen Datenstrukturen, die sich wiederholende Strukturen oder individuelle Elemente aufweisen.

Dieses Kapitel behandelt folgende Themen:

- Arrays definieren
- Ausgangswerte für Arrays
- Ausgangswerte für eindimensionale Arrays zuweisen
- Ausgangswerte für zweidimensionale Arrays zuweisen
- Dreidimensionales Array
- Arrays als Teil einer größeren Datenstruktur
- Datenbank-Arrays
- Arithmetische Ausdrücke in Index-Notationen
- Arithmetische Funktionen bei Arrays

---

# Arrays definieren

Ein Natural-Array kann ein-, zwei- oder dreidimensional sein. Es kann eine unabhängige Variable, Teil einer größeren Datenstruktur oder Teil einer Datenbanksicht sein.

**Wichtig:**
Dynamische Variablen sind in einer Array-Definition nicht zulässig.

### Um ein eindimensionales Array zu definieren

- Geben Sie hinter Format und Länge einen Schrägstrich (/) und danach eine Index-Notation, d.h. die Anzahl der Ausprägungen des Arrays, an.

  Das folgende Array hat zum Beispiel drei Ausprägungen, wobei jede Ausprägung Format/Länge A10 hat:

```
DEFINE DATA LOCAL
1 #ARRAY (A10/1:3)
END-DEFINE
...
```

### Um ein zweidimensionales Array zu definieren

- Geben Sie für beide Dimensionen eine Index-Notation an:

  ```
  DEFINE DATA LOCAL
  1 #ARRAY (A10/1:3,1:4)
  END-DEFINE
  ...
  ```

  Ein zweidimensionales Array kann man sich als Tabelle vorstellen. Das im obigen Beispiel definierte Array wäre demnach eine Tabelle, die aus 3 Zeilen und 4 Spalten besteht:

| | | | |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

# Ausgangswerte für Arrays

Um einer oder mehreren Ausprägungen eines Arrays einen Ausgangswert zuzuweisen, verwenden Sie, ähnlich wie für "einfache" Variablen (siehe folgende Beispiele), eine `INIT`-Angabe.

# Ausgangswerte für eindimensionale Arrays zuweisen

Die folgenden Beispiele veranschaulichen, wie einem eindimensionalen Array Ausgangswerte zugewiesen werden:

- Um einer einzelnen Ausprägung einen Ausgangswert zuzuweisen, geben Sie an:

  ```
  1 #ARRAY (A1/1:3) INIT (2) <'A'>
  ```

  A wird der zweiten Ausprägung zugewiesen.

- Um allen Ausprägungen den gleichen Ausgangswert zuzuweisen, geben Sie an:

  ```
  1 #ARRAY (A1/1:3) INIT ALL <'A'>
  ```

  A wird jeder Ausprägung zugewiesen. Stattdessen könnten Sie auch angeben:

  ```
  1 #ARRAY (A1/1:3) INIT (*) <'A'>
  ```

- Um einem Bereich von mehreren Ausprägungen den gleichen Ausgangswert zuzuweisen, geben Sie an:

  ```
  1 #ARRAY (A1/1:3) INIT (2:3) <'A'>
  ```

  A wird der zweiten bis dritten Ausprägung zugewiesen.

- Um jeder Ausprägung einen anderen Ausgangswert zuzuweisen, geben Sie an:

  ```
  1 #ARRAY (A1/1:3) INIT <'A','B','C'>
  ```

  A wird der ersten Ausprägung zugewiesen, B der zweiten und C der dritten.

- Um verschiedenen (aber nicht allen) Ausprägungen verschiedene Ausgangswerte zuzuweisen, geben Sie an:

  ```
  1 #ARRAY (A1/1:3) INIT (1) <'A'> (3) <'C'>
  ```

  A wird der ersten Ausprägung zugewiesen und C der dritten; der zweiten Ausprägung wird kein Wert zugewiesen.

  Stattdessen könnten Sie auch angeben:

  ```
  1 #ARRAY (A1/1:3) INIT <'A',,'C'>
  ```

- Wenn weniger Ausgangswerte angegeben werden als Ausprägungen vorhanden sind, bleiben die letzten Ausprägungen leer:

  ```
  1 #ARRAY (A1/1:3) INIT <'A','B'>
  ```

  A wird der ersten Ausprägung zugewiesen und B der zweiten; der dritten Ausprägung wird kein Wert zugewiesen

# Ausgangswerte für zweidimensionale Arrays zuweisen

Dieser Abschnitt zeigt, wie einem zweidimensionalen Array Ausgangswerte zugewiesen werden. Die folgenden Themen werden behandelt:

- Vorbemerkung
- Gleichen Wert zuweisen
- Unterschiedliche Werte zuweisen

## Vorbemerkung

Für die Beispiele gehen wir von einem zweidimensionalen Array aus, das drei Ausprägungen in der ersten Dimension (Zeilen) und vier Ausprägungen in der zweiten Dimension (Spalten) hat:

```
1 #ARRAY (A1/1:3,1:4)
```

### Vertikal: Erste Dimension (1:3), Horizontal: Zweite Dimension (1:4):

| | | | |
|---|---|---|---|
| (1,1) | (1,2) | (1,3) | (1,4) |
| (2,1) | (2,2) | (2,3) | (2,4) |
| (3,1) | (3,2) | (3,3) | (3,4) |

Die erste Gruppe von Beispielen veranschaulicht, wie den Ausprägungen eines zweidimensionalen Arrays der *gleiche* Ausgangswert zugewiesen wird; die zweite Gruppe von Beispielen veranschaulicht, wie *unterschiedliche* Ausgangswerte zugewiesen werden.

Beachten Sie bei den Beispielen insbesondere die Verwendung der Notationen * und V. Beide Notationen beziehen sich auf alle Ausprägungen der betreffenden Dimension: Mit * werden *alle* Ausprägungen der betreffenden Dimension mit dem gleichen Wert initialisiert, mit V werden alle Ausprägungen der betreffenden Dimension mit *unterschiedlichen* Werten initialisiert.

## Gleichen Wert zuweisen

- Um einer Ausprägung einen Ausgangswert zuzuweisen, geben Sie an:

```
1 #ARRAY (A1/1:3,1:4) INIT (2,3) <'A'>
```

| | | | |
|---|---|---|---|
| | | | |
| | | A | |
| | | | |

- Um einer Ausprägung der zweiten Dimension — in allen Ausprägungen der ersten Dimension — den gleichen Ausgangswert zuzuweisen, geben Sie an:

```
1 #ARRAY (A1/1:3,1:4) INIT (*,3) <'A'>
```

| | | | |
|---|---|---|---|
| | | A | |
| | | A | |
| | | A | |

- Um einem Bereich von Ausprägungen der ersten Dimension — in allen Ausprägungen der zweiten Dimension — den gleichen Ausgangswert zuzuweisen, geben Sie an:

```
1 #ARRAY (A1/1:3,1:4) INIT (2:3,*) <'A'>
```

| | | | |
|---|---|---|---|
| | | | |
| A | A | A | A |
| A | A | A | A |

- Um einem Bereich von Ausprägungen in beiden Dimensionen den gleichen Ausgangswert zuzuweisen, geben Sie an:

```
1 #ARRAY (A1/1:3,1:4) INIT (2:3,1:2) <'A'>
```

| | | | |
|---|---|---|---|
| | | | |
| A | A | | |
| A | A | | |

- Um allen Ausprägungen (in beiden Dimensionen) den gleichen Ausgangswert zuzuweisen, geben Sie an:

```
1 #ARRAY (A1/1:3,1:4) INIT ALL <'A'>
```

| | | | |
|---|---|---|---|
| A | A | A | A |
| A | A | A | A |
| A | A | A | A |

Stattdessen könnten Sie auch angeben:

```
1 #ARRAY (A1/1:3,1:4) INIT (*,*) <'A'>
```

## Unterschiedliche Werte zuweisen

- ```
  1 #ARRAY (A1/1:3,1:4) INIT (V,2) <'A','B','C'>
  ```

| | | | |
|---|---|---|---|
| | A | | |
| | B | | |
| | C | | |

- ```
  1 #ARRAY (A1/1:3,1:4) INIT (V,2:3) <'A','B','C'>
  ```

| | | | |
|---|---|---|---|
| | A | A | |
| | B | B | |
| | C | C | |

- ```
  1 #ARRAY (A1/1:3,1:4) INIT (V,*) <'A','B','C'>
  ```

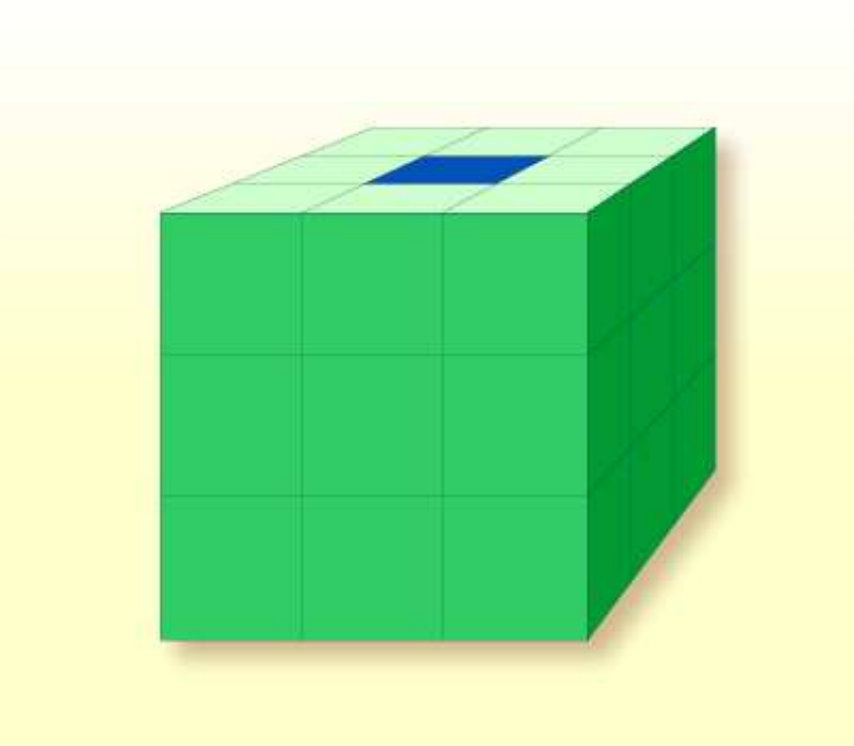
| | | | |
|---|---|---|---|
| A | A | A | A |
| B | B | B | B |
| C | C | C | C |

- ```
  1 #ARRAY (A1/1:3,1:4) INIT (V,*) <'A',,'C'>
  ```

| | | | |
|---|---|---|---|
| A | A | A | A |
| | | | |
| C | C | C | C |

- ```
  1 #ARRAY (A1/1:3,1:4) INIT (V,*) <'A','B'>
  ```

| | | | |
|---|---|---|---|
| A | A | A | A |
| B | B | B | B |
| | | | |
  ```

- ```
  1 #ARRAY (A1/1:3,1:4) INIT (V,1) <'A','B','C'> (V,3) <'D','E','F'>
  ```

| | | | |
|---|---|---|---|
| A | | D | |
| B | | E | |
| C | | F | |

- ```
  1 #ARRAY (A1/1:3,1:4) INIT (3,V) <'A','B','C','D'>
  ```

| | | | |
|---|---|---|---|
| | | | |
| | | | |
| A | B | C | D |

- ```
  1 #ARRAY (A1/1:3,1:4) INIT (*,V) <'A','B','C','D'>
  ```

| | | | |
|---|---|---|---|
| A | B | C | D |
| A | B | C | D |
| A | B | C | D |

- ```
  1 #ARRAY (A1/1:3,1:4) INIT (2,1) <'A'> (*,2) <'B'> (3,3) <'C'> (3,4) <'D'>
  ```

| | | | |
|---|---|---|---|
| | B | | |
| A | B | | |
| | B | C | D |

- ```
  1 #ARRAY (A1/1:3,1:4) INIT (2,1) <'A'> (V,2) <'B',C',D'> (3,3) <'E'> (3,4) <'F'>
  ```

| | | | |
|---|---|---|---|
| | B | | |
| A | C | | |
| | D | E | F |

# Dreidimensionales Array

Ein dreidimensionales Array könnte man sich folgendermaßen vorstellen:
  ```



Das oben dargestellte Array müsste wie folgt definiert werden (wobei gleichzeitig dem dunkel markierten Feld `#FIELD2`, das die Position Zeile 1, Spalte 2, Ebene 2 hat, ein Ausgangswert zugewiesen wird):

```
DEFINE DATA LOCAL
1 #ARRAY2
  2 #ROW (1:4)
    3 #COLUMN (1:3)
      4 #PLANE (1:3)
        5 #FIELD2 (P3) INIT (1,2,2) <100>
END-DEFINE
...
```

Wenn man das gleiche Array im Data-Area-Editor als Local Data Area definiert, sieht dies so aus:

```
I T L Name                            F Leng Index/Init/EM/Name/Comment
- - - ------------------------------- - ---- --------------------------------
    1 #ARRAY2
    2 #ROW                                   (1:4)
    3 #COLUMN                                (1:3)
    4 #PLANE                                 (1:3)
I   5 #FIELD2                         P    3
```

# Arrays als Teil einer größeren Datenstruktur

Die Mehrdimensionalität von Arrays ermöglicht es, Datenstrukturen analog zu COBOL- oder PL1-Strukturen zu definieren.

**Beispiel:**

```
DEFINE DATA LOCAL
1 #AREA
  2 #FIELD1 (A10)
  2 #GROUP1 (1:10)
    3 #FIELD2 (P2)
    3 #FIELD3 (N1/1:4)
END-DEFINE
...
```

Im obigen Beispiel hat der Datenbereich #AREA insgesamt eine Größe von:

10 + (10 * (2 + (1 * 4))) Bytes = 70 Bytes

#FIELD1 ist alphanumerisch und 10 Bytes lang. #GROUP1 ist ein Unterbereich von #AREA, hat 10 Ausprägungen und besteht aus zwei Feldern: #FIELD2 und #FIELD3. #FIELD2 ist gepackt numerisch und 2 Bytes lang; #FIELD3 ist das zweite Feld von #GROUP1, ist numerisch, 1 Byte lang und hat 4 Ausprägungen.

Wollen Sie eine bestimmte Ausprägung von #FIELD3 referenzieren, sind hierzu zwei Angaben erforderlich: erstens die der betreffenden Ausprägung von #GROUP1 und zweitens die der betreffenden Ausprägung von #FIELD3. Falls #FIELD3 beispielsweise an anderer Stelle im Programm in einem ADD-Statement referenziert würde, sähe dies folgendermaßen aus:

```
ADD 2 TO #FIELD3 (3,2)
```

# Datenbank-Arrays

Adabas unterstützt Array-Strukturen innerhalb einer Datenbank in Form von multiplen Feldern und Periodengruppen. Diese sind im Abschnitt *Datenbank-Arrays* beschrieben.

Das folgende Beispiel zeigt einen DEFINE DATA-View, der ein multiples Feld enthält, zunächst programmintern und dann in einer programmexternen Local Data Area definiert:

```
DEFINE DATA LOCAL
1 EMPLOYEES-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 ADDRESS-LINE (1:10) /* <--- MULTIPLE-VALUE FIELD
END-DEFINE
...
```

Dieselbe View in einer programmexternen Local Data Area:

```
I T L Name                            F Leng Index/Init/EM/Name/Comment
- - - --------------------------------- - ---- --------------------------------
  V 1 EMPLOYEES-VIEW                          EMPLOYEES
    2 NAME                            A   20
  M 2 ADDRESS-LINE                    A   20 (1:10) /* MU-FIELD
```

# Arithmetische Ausdrücke in Index-Notationen

Zur Bestimmung eines Bereiches von Ausprägungen in einem Array können auch einfache arithmetische Ausdrücke verwendet werden.

Beispiele:

| | |
|---|---|
| **MA (I:I+5)** | 6 Werte des Feldes MA werden referenziert, beginnend mit Wert I und endend mit Wert I + 5. |
| **MA (I+2:J-3)** | Die Werte des Feldes MA von I + 2 bis J – 3 werden referenziert. |

In derartigen Index-Angaben dürfen keine anderen Rechenzeichen als + und – verwendet werden.

# Arithmetische Funktionen bei Arrays

Arithmetische Funktionen lassen sich innerhalb von Arrays auf Tabellenebene, auf Zeilen-/Spaltenebene und auf Feldebene einsetzen.

Allerdings sind mit Array-Variablen nur einfache arithmetische Funktionen erlaubt, die höchstens ein oder zwei Operanden enthalten sowie möglicherweise eine dritte Variable als Ergebnisfeld.

Werden Indexbereiche definiert, so sind nur die Operatoren + und – zulässig.

## Beispiele für Array-Arithmetik:

Die folgenden Beispiele gehen von den folgenden Felddefinitionen aus:

```
DEFINE DATA LOCAL
01 #A (N5/1:10,1:10)
01 #B (N5/1:10,1:10)
01 #C (N5)
END-DEFINE
...
```

1. 

   ```
   ADD #A(*,*) TO #B(*,*)
   ```

   Der Ergebnisoperand, Array #B, enthält die elementweise Addition des Arrays #A und des ursprünglichen Werts von Array #B.

2. 

   ```
   ADD 4 TO #A(*,2)
   ```

   Die zweite Spalte des Arrays #A wird durch den ursprünglichen Wert plus 4 ersetzt.

3. 

   ```
   ADD 2 TO #A(2,*)
   ```

   Die zweite Zeile des Arrays #A wird durch den ursprünglichen Wert plus 2 ersetzt.

4. 

   ```
   ADD #A(2,*) TO #B(4,*)
   ```

   Der Wert der zweiten Zeile des Arrays #A wird zur vierten Zeile des Arrays #B addiert.

5.

```
ADD #A(2,*) TO #B(*,2)
```

Diese Operation ist nicht erlaubt und würde von Natural als Syntaxfehler zurückgewiesen. Es ist nicht gestattet, bei arithmetischen Funktionen Zeilen mit Spalten zu vermischen.

6.

```
ADD #A(2,*) TO #C
```

Alle Werte der zweiten Zeile des Arrays #A werden zu dem Skalarwert #C addiert.

7.

```
ADD #A(2,5:7) TO #C
```

Die Werte der 5. bis 7. Spalte der zweiten Zeile des Arrays #A werden zum Skalarwert #C addiert.