Arrays im DATA-Step - wann und warum?

Ralf Minkenberg
Boehringer Ingelheim Pharma GmbH & Co. KG
Binger Str. 173
55216 Ingelheim
ralf.minkenberg@boehringer-ingelheim.com

Zusammenfassung

Obwohl sicherlich jeder Programmierer schon mit Arrays im DATA-Step Erfahrung gemacht hat, werden diese doch seltener angewandt als sinnvoll sein könnte. Die Syntax bei Definition und Verwendung von Arrays erscheint manchmal nicht klar - und schließlich geht es ja auch ohne Arrays! Es soll gezeigt werden, wie Arrays im DATA-Step definiert werden können; besonders der Unterschied zwischen expliziter Definition – in der Dokumentation zu SAS zu finden – und impliziter Definition – nur aufgrund Kompatibilität mit älteren SAS-Versionen verfügbar – wird in Beispielen erläutert. In diesen Beispielen wird auch die Verwendung von Arrays dargestellt. Weiterhin werden Anwendungen von speziellen Befehlen und Variablen gezeigt, die nur in Verbindung mit Arrays benutzt werden können.

Schlüsselwörter: Array, Data-Step

1 Das Array-Statement

Immer wenn in einem Datenschritt in SAS Befehle oder Berechnungen nicht nur für eine, sondern für mehrere Variablen auf die gleiche Weise durchgeführt werden sollen, bietet sich – anstatt für jede Variable die gleichen Statements zu wiederholen – die Benutzung von Arrays an. Jedoch scheint bei der Benutzung von Arrays oft Unsicherheit zu bestehen, wie genau diese benutzt werden können und was tatsächlich erlaubt und möglich ist. Sehr häufig werden dann doch die benötigten Codezeilen für jede Variable kopiert, was ein Programm oft unnötig lang und damit auch unübersichtlich machen kann. Im Folgenden soll ein Überblick über Syntax und Verwendung von Array-Befehlen im Data-Step gegeben werden, der hoffentlich motiviert, Arrays häufiger zu verwenden bzw. deren Benutzung besser zu verstehen.

1.1 Die Syntax des Array-Statements

Die allgemeine Syntax des Array-Befehls (siehe auch [1]) hat folgendes Aussehen:

```
ARRAY array-name {subscript} <$> <length> <array-elements>
<(initial-value-list)> ;
```

Die einzelnen Teile dieses Befehls werden nun etwas näher erläutert.

`ARRAY` ist der Befehl zur Initialisierung eines Arrays. Mit ihm wird eine Variablenmenge als Elemente eines Arrays zusammengefasst, damit auf einfache Art alle (oder bestimmte einzelne) Elemente dieser Menge angesprochen werden können. Die Variablen einer solchen Variablenmenge müssen alle den gleichen Typ (numeric bzw. character) haben. Es wird hier keine Datenstruktur definiert, es handelt sich nur um die Gruppierung von Variablen, um bestimmte Befehle auf die gesamte Variablenmenge anwenden zu können.

`array-name` gibt den Namen des Arrays an, welcher (nahezu) beliebig vom Benutzer gewählt werden kann. Der Name identifiziert das Array im Data-Step, in dem er definiert wird, und auch nur dort – außerhalb des Data-Steps, in dem ein Array definiert wurde, ist das Array (und damit der Arrayname) nicht bekannt und kann dort nicht verwendet werden. Der Name eines Arrays muss den gleichen Vorschriften genügen wie (z.B.) ein Variablenname. Weiterhin sollte es vermieden werden (wenn es auch nicht verboten ist), als Arrayname den Namen einer SAS-Funktion zu wählen. Eventuell können (wenn die Funktion im gleichen Data-Step benutzt werden soll) unerwünschte Effekte auftreten. Zwingend erforderlich ist, dass ein Arrayname sich von allen im Data-Step verwendeten Variablennamen unterscheidet. Der Name eines Arrays ist auf keinen Fall eine Variable.

`{subscript}` definiert die Anzahl und Anordnung der Array-Elemente. Es ist möglich, sowohl die Anzahl festzulegen als auch die Anordnung („Reihenfolge“) zu beschreiben. Es kann auch die Dimension des Arrays bestimmt werden. Es kann hier nur eine Zahl, mehrere Zahlen oder ein Stern (*) verwendet werden. Immer muss das Subscript in Klammern eingeschlossen werden, wobei {}, [] als auch () erlaubt sind (am Anfang und Ende natürlich immer die gleiche Klammer). Beispiele für die verschiedenen Möglichkeiten:

```
{Dimension}         array simple {3};            array x{5,3};
{lower : upper}     array x {1:5, 1:3};
{*}                 array col {*};
```

`<array-elements>` enthält die Variablennamen, welche das definierte Array füllen sollen. In jedem Array müssen alle angegebenen Variablen entweder numeric oder character sein, ein Mix zwischen beiden Typen ist nicht erlaubt. Es gibt einige spezielle Namen, die bestimmte Gruppen von Variablen zusammenfassen:

all	alle zum Zeitpunkt der Initialisierung des Arrays bekannten Variablen (funktioniert nur, wenn alle Variablen entweder numeric oder alle Variablen character sind)
character	alle character-Variablen zum Zeitpunkt der Initialisierung des Arrays
numeric	alle numeric-Variablen zum Zeitpunkt der Initialisierung des Arrays
temporary	die im Array definierten Elemente sind nur temporär definiert und keinem Variablennamen zugeordnet

`<(initial-value-list)>` eine Liste von Anfangswerten, die den Array-Elementen zugewiesen werden. Wenn hier Werte für bereits im Data-Step existierende Variablen angegeben werden, werden die vorhandenen Werte kommentarlos überschrieben.

1.2 Beispiele

Die Definition und Initialisierung eines Arrays kann am besten durch einige Beispiele veranschaulicht werden. Zunächst wird hinter dem Arraynamen jeweils die Dimension des Arrays angegeben und die Variablennamen, die in dem Array angesprochen werden sollen, benannt. Es ist möglich, sowohl Variablen zu verwenden, die bereits im Data-Step bekannt sind, als auch solche, die noch nicht verwendet wurden, welche dann durch die Array-Initialisierung im Data-Step angelegt werden.

```
array rain {5} janr febr marr aprr mayr ;
array month {*} jan feb jul okt nov ;
```

In beiden Beispielen wird ein Array bestehend aus 5 Variablen definiert. Im ersten Fall wird mit `rain {3}` beispielsweise die Variable `marr` angesprochen.

Auf die explizite Angabe der Variablen kann auch verzichtet werden. In diesem Fall sind dann als Variablennamen die Zusammensetzung aus dem Array-Namen und einer Zahl definiert:

```
array meal {3} ;
```

Durch diesen Array-Befehl wird ein Array mit den Variablen `meal1`, `meal2` und `meal3` definiert.

Neben ein-dimensionalen Arrays können auch mehr-dimensionale Arrays definiert werden:

```
array score{5,3} score1-score15 ;
array test {3:4,3:7} test1-test10 ;
```

Im ersten Fall wird ein zwei-dimensionales Array mit 5 mal 3 Variablen definiert. Mit `score {3,2}` würde beispielsweise dann die Variable `score8` benannt. Im zweiten Fall wird ein zwei-dimensionales Array mit 2 mal 5 Variablen definiert, jedoch beginnen die Indizes in beiden Dimensionen nicht bei 1, sondern bei 3. Somit wäre mit `test {4,4}` z.B. die Variable `test7` gemeint.

In den folgenden drei Beispielen werden auch Anfangswerte bei der Initialisierung des Arrays definiert, die dann den genannten Variablen zugeordnet werden.

```
array test {4} t1-t4 (90 80 2*70) ;
array zeit {5} _temporary_ (0 3 7 14 35) ;
array x {10} a b c d e f g h i j (2*1:5) ;
```

Im ersten Array `test` sind die 4 Variablen `t1`, `t2`, `t3` und `t4` mit den Werten `90`, `80`, `70` bzw. `70` besetzt. Im dritten Array `x` sind die 10 Variablen zweimal nacheinander mit den Zahlen 1 bis 5 belegt, also ist der Wert von `x {7}` und somit der Variable `g` gleich `2`. Bei dem zweiten Array `zeit` werden 5 temporäre Array-Elemente `zeit {1}`, `zeit {2}`, … definiert mit vorgegebenen Zahlenwerten. Für diese Array-Elemente gibt es keine Variablennamen im Data-Step.

1.3 Einige Anwendungen

Arrays erweisen sich immer dann als nützlich, wenn für mehrere Variablen in einem Data-Step gleiche Berechnungen oder Statements durchgeführt werden sollen. Im ersten, folgenden Beispiel soll für Variablen `days1`, `days2`, …, `days7` jeweils ein evtl. gespeicherter Wert von 99 auf 100 geändert werden. Gleichzeitig werden 7 neue Variablen `h1`, `h2`, …, `h7` definiert, in denen der Wert aus der entsprechenden `days`-Variable mal 24 gespeichert werden.

```
data bsp1;
  array days {7} ;  array hours {7} h1-h7 ;
  do i=1 to 7 ;
    if days {i} = 99 then days {i} = 100 ;
    hours {i} = days {i} * 24 ;
  end ;
run;
```

Obwohl siebenmal dieselben zwei Codezeilen ausgeführt werden, tauchen sie im Data-Step nur einmal (innerhalb einer `do`-Schleife) auf. Dies macht den entsprechenden Code kürzer und auch übersichtlicher.

```
data bsp2;
  array big {3,4} wt1-wt4 ht1-ht4 bmi1-bmi4 ;
  do i=1 to dim(big,2) ;
    big {3,i} = big{1,i} / big{2,i}**2 ;
  end ;
run;
```

In diesem Beispiel-Datensatz gibt es zu 4 verschiedenen Zeitpunkten jeweils ein Messwert für das Gewicht (`wt`) und die Größe (`ht`) einer Person. Zu jedem Zeitpunkt soll jetzt der Body-Mass-Index (`bmi`) bestimmt werden. Das zwei-dimensionale Array `big {3,4}` enthält alle involvierten 3 Variablen zu allen 4 Zeitpunkten. Mit dieser Array-Definition lassen sich die erforderlichen Berechnungen dann in einer Zeile durchführen. Dies erleichtert natürlich auch evtl. notwendige Formel-Korrekturen, da diese nur an einer einzigen Stelle durchzuführen sind. Im Beispiel ist auch zu erkennen, wie der `dim`-Befehl, der die Anzahl Elemente einer Dimension eines Arrays zurückgibt, bei mehrdimensionalen Arrays verwendet werden kann.

1.4 Änderung der Datenstruktur mit Arrays

Mit Hilfe von Arrays ist es auch leicht möglich, die Datenstruktur eines Datensatzes zu verändern. Sei beispielsweise ein horizontal strukturierter Datensatz mit jeweils 3 Körpergrößen und -gewichte gegeben, der in eine vertikale Struktur überführt werden soll, bei der für jeden Mess-Zeitpunkt eine Beobachtung angelegt sein soll:

PATNO	HTCM1	HTCM2	HTCM3	WTKG1	WTKG2	WTKG3
10001	122	124	124	43	44.5	45
10002	109	112	117	33.5	36	39
10003	115	115	119	40	39	46

Ein Programm zur Lösung der gestellten Aufgabe könnte so aussehen:

```
data vertical ;
  set horizontal ;
  array ahtcm {1:3} htcm1-htcm3 ;
  array awtkg {1:3} wtkg1-wtkg3 ;
  do timept=1 to 3 ;
    bmi = awtkg {timept} /
        (ahtcm {timept} / 100) ** 2 ;
    htcm = ahtcm {timept} ;
    wtkg = awtkg {timept} ;
    output ;
  end ;
  keep timept patno htcm wtkg bmi ;
  format bmi 4.1;
run ;
```

Der Ergebnis-Datensatz sähe dann so aus:

PATNO	TIMEPT	HTCM	HTKG	BMI
10001	1	122	43	28.9
10001	2	124	44.5	28.9
10001	3	124	45	29.3
10002	1	109	33.5	28.2
10002	2	112	36	28.7
…	…	…	…	…

2 Explizite und implizite Arrays

Alle bisher gezeigten Array-Definitionen und -Anwendungen benutzen sog. explizite Arrays, d.h. die Anzahl Elemente (in verschiedenen Dimensionen) ist explizit definiert. Aus älteren SAS-Versionen ist aus Kompatibilitätsgründen auch noch die sog. implizite Definition bzw. Initialisierung eines Arrays beibehalten worden. Implizite Arrays sollen in neueren SAS-Versionen nach Möglichkeit nicht mehr verwendet werden. Da diese jedoch weiterhin unterstützt werden und unter Umständen eine Erleichterung bei der Programmierung von Data-Steps bringen können, werden diese im Folgenden näher betrachtet.

2.1 Unterschiede in der Definition

Die bereits vorgestellte explizite Definition eines Arrays und dessen Verwendung könnte z.B. so aussehen:

```
array item {1:6} xvar  yvar  zvar  xxvar  yyvar  zzvar ;
do i=1 to 6 ;   item {i} = item {i} / 100 ;   end ;
```

Es wird ein Array `item` definiert, bestehend aus 6 Variablen `xvar`, `yvar`, … In einer Schleife wird dann für jede Variable des Arrays eine Berechnung ausgeführt.

Ein entsprechendes implizites Array kann mit folgendem Code definiert und benutzt werden:

```
array item  xvar  yvar  zvar  xxvar  yyvar  zzvar ;
do over item ;   item = item / 100 ;   end ;
```

Bei der Definition eines impliziten Arrays wird nach dem array-Befehl und dem Namen des Arrays direkt die Variablenmenge des Arrays angegeben, es gibt keine Angabe von Dimension bzw. Variablenanzahl. Dies ermöglicht bei der Benutzung eines solchen Arrays die Verwendung einer do over-Schleife, die über alle Variablen des Arrays ausgeführt wird. Bei der Angabe der Array-Elemente innerhalb einer solchen Schleife darf daher auch kein Index verwendet werden, sondern es wird nur der Arrayname selber angegeben. Hierbei ist sicher zu beachten, dass diese Syntax zu Schwierigkeiten führen kann, einen Variablen- von einem Arraynamen zu unterscheiden.

Es ist auch möglich, implizite und explizite Arrays neben- und miteinander zu verwenden. In folgendem Beispiel wird ein implizites Array `item` und ein explizites Array `results` definiert:

```
array item  xvar  yvar  zvar ;
array results(1:3) ;
do over item ; if item > 3.5 then results {_i_} = 1 ; end ;
```

Die interne Data-Step-Variable `_i_` kann verwendet werden, um bestimmte einzelne Variablen eines impliziten Arrays zu identifizieren. Der Wert von `_i_` entspricht also immer der Position der gerade angesprochenen Variable innerhalb des impliziten Arrays. Im Beispiel können derart die Variablen des expliziten Arrays `results` abhängig von Werten der Variablen des impliziten Arrays `item` gefüllt werden.

2.2 Ein Beispiel mit implizitem Array

In diesem Beispiel sei ein Datensatz mit 6 Variablen (val1-val6) gegeben. Es soll für jede Beobachtung sowohl die Anzahl an Variablen mit nicht-fehlenden Werten als auch

die Variablennummer des letzten nicht-fehlenden Werts bestimmt werden (jeweils aus den 6 Variablen). Falls in den Variablen negative Werte vorhanden sind, müssen diese auch als fehlende Werte angesehen werden.

```
data test (keep=id n_of_v last_of_v);
  set original ;
  array values val: ;
  n_of_v = 0 ; last_of_v = 0 ;
  do over values ;
    if values < 0 then values = . ;
    else n_of_v + 1 ;
    if values ^= . then last_of_v = _i_ ;
  end ;
run ;
```

Um die Vorgehensweise dieses Data-Steps zu verstehen, betrachten wir folgende Beispieldaten (`original`):

ID	VAL1	VAL2	VAL3	VAL4	VAL5	VAL6
1	4	-1	3	3	.	.
2	-1	2	6	5	-1	2
3	0	1	3	2	5	.

Nach Anwendung obigen Codes ergibt sich im Ergebnis-Datensatz (`test`) folgendes Bild:

ID	N_OF_V	LAST_OF_V
1	3	4
2	4	6
3	5	5

3 Fehlermeldungen im Zusammenhang mit Arrays

Bei der Benutzung von Arrays im Data-Step können eine Vielzahl an Meldungen im LOG ausgegeben werden. Bei der folgenden Übersicht ist jeweils der Platzhalter ____ in einem konkreten Fall mit einem passenden Namen belegt (Name eines Arrays, einer Variable, …).

3.1 Fehlermeldungen (ERROR)

```
ERROR: Undeclared array referenced: _____.
ERROR: Variable _____ has not been declared as an array.
```

Es wird ein Array-Element angesprochen (z.B. test{i}), ohne dass ein entsprechendes Array definiert wurde.

```
ERROR: Array subscript out of range at line ___ column ___.
```

Ein verwendeter Index nimmt einen Wert an, der außerhalb der bei der Initialisierung angegebenen erlaubten Werten liegt.
(Beispiel: `array klein{1:4}; do i=1 to 5; klein{i} = 0; end;`)

```
ERROR: Attempt to initialize variable _____ in numeric array _____
       with character constant.
```

Bei der Angabe von Anfangselementen wird versucht, ein character-Wert in einem Array mit numeric-Werten anzugeben.

```
ERROR: Too few array subscripts specified for array _____.
ERROR: Too many array subscripts specified for array _____.
```

Bei der Benutzung von Arrayelementen im Data-Step sind zu wenige/viele Indizes angegeben worden, d.h. die Dimensionsanzahl ist inkorrekt.

```
ERROR: Too many variables defined for the dimension(s) specified for
       the array _____.
ERROR: Too few variables defined for the dimension(s) specified for
       the array _____.
```

Es werden zu viele/wenige Variablen bei der Definition eines Arrays angegeben.

```
ERROR: Illegal reference to the array _____.
```

Diese Meldung wird z.B. ausgegeben, wenn ein explizites Array ohne Index benutzt wird.

```
ERROR: The array _____ has been defined with zero elements.
ERROR: Invalid dimension specification for array _____.  The upper
       bound of an array dimension is smaller than its corresponding
       lower bound.
```

Fehler beim Definieren der Anzahl der Array-Elemente bzw. Dimensionen.

3.2 Weitere Meldungen (WARNING, NOTE)

Neben Fehlermeldungen, deren Ursache natürlich immer untersucht und beseitigt werden muss, werden hier noch ein paar weitere Meldungen im LOG im Zusammenhang mit Arrays vorgestellt. Auch hier ist der Platzhalter ____ entsprechend sinnvoll zu ersetzen.

```
WARNING: Partial value initialization of the array _____.
WARNING: Too many values for initialization of the array _____.
         Excess values are ignored.
```

Bei der Angabe von Anfangswerten für ein Array sind nicht alle bzw. zu viele Werte angegeben worden.

```
NOTE: The array _____ has the same name as a SAS-supplied or
      user-defined function. Parentheses following this name are
      treated as array references and not function references.
```

Diese Meldung wird ausgegeben, wenn der Name eines definierten Arrays gleich dem Namen einer SAS-Funktion ist. Im gesamten Data-Step wird dann bei Verwendung dieses Namens nicht die Funktion aufgerufen, sondern dies immer als Array-Verwendung interpretiert.

4 Zusammenfassung

Es ist kein Problem, jeden Data-Step für jedes Problem ohne die Benutzung von Arrays zu verwirklichen. Jedoch helfen Arrays in vielen Situationen, einfacheren, transparenteren und nachvollziehbareren Code zu erhalten.
Arrays sind innerhalb eines Data-Steps vor allem dann hilfreich, wenn zusammenhängende oder sich wiederholende Variablen auftauchen, für die mehrere gleiche Befehle oder Berechnungen durchgeführt werden müssen. Auch Restrukturierungen großer Datenmengen lassen sich übersichtlich programmieren. Bei Änderungen der Anzahl dieser Variablen ist ein Programm schnell und einfach anzupassen. Komplexe Beziehungen zwischen Variablen (die evtl. auch nicht aus den Variablennamen ersichtlich sind) können mit Hilfe von Arrays transparent und nachvollziehbar abgebildet werden. Bei der Notwendigkeit eine Vielzahl an Konstanten zu benutzen, sollten Arrays auf jeden Fall in Betracht gezogen werden.

Literatur

[1] SAS Institute Inc. 2013. SAS® 9.4 Statements: Reference, Second Edition. Cary, NC: SAS Institute Inc.