Abstract
Arrays are powerful, flexible and confusing. Using arrays in SAS code makes the code more compact and effective. By looking at how arrays work and why we use them, we can enjoy the advantages and avoid the angry bite of an abused array.

Introduction
To understand how to use arrays in the SAS system we must look at why we use arrays, in the SAS system, or in any other.

Arrays are a common feature to most languages. In fact, it's difficult to find a language that doesn't have arrays in one form or another. It seems that no matter what language or system you use, it has provisions for collecting variables and dealing with that collection. But it is also common to find that people who use the system don't use arrays in many situations that would make their jobs easier. Why is that?

There are three main reasons people avoid using arrays:

Arrays make them nervous.

Mistakes are easy to make.

Mistakes may be hard to find.

These reasons are not lightly disregarded. In fact, whenever someone pouring over the code is saying "It's right! It's right! So WHY doesn't it work?!" there's a pretty good chance there is an array involved.

Even with these difficulties, many people use arrays as part of their standard toolkit for solving problems.

Why use arrays?
Tools and strategies have been developed to deal with different problems. One of the most common strategies people are taught is how to deal very large, complex problems. Different terms are used: decompose, partition, factor. Often Julius Caesar is cited as an example of how to "divide and conquer". Divide and conquer is a useful strategy, but not all of the problems we face are, by nature, large and monolithic. Sometimes, the problem is the opposite. Sometimes, the solution demands that we deal with a large number of small parts. What do we do then? To quote another great thinker:

"Put all your eggs in one basket, and WATCH that basket"
-- Carl Sandburg

The solution for dealing with large numbers of small parts is to collect them. If we can organize them at the same time, so much the better.

Bringing it all together
To collect things, we should look at what we have to see the best way to bring them together. There are two logical ways to bind things: by type and by process.

If we looked at a list of names, we would recognize that it represented a collection of people. If we looked a list of business, it might take a little longer to recognize them, but we would again see that there was something common to all the items. Furthermore, with proper organization and order, these two collections provide information that is easily accessed and readily used. The Phone company has been doing this for years.
The second method for collecting things is not as obvious. If we spotted a list that showed: milk, bread, apples, light bulbs, stamps, and spark plugs; we'd have a hard time discovering what they all had in common. In fact, these items themselves are different. They have nothing in common. The reason they appear on the same list, is because they will all be involved in the same process. We need to remember to buy them the next time we go shopping. Collecting things by common process is less obvious than collecting things with common properties, but it's useful. People make shopping lists all the time.

The SAS system solution

The SAS system provides methods for collecting variables that match the two reasons for collecting them:

- Numbered variables are used to collect variables that represent the same TYPE of data.

  The ARRAY statement is used to collect variables that will share the same PROCESSING.

And, when variables have the same type and the same processing at the same time, the SAS system provides a short cut combination.

The SAS system imposes one restriction on all types of arrays: The elements in an array must all be numeric variables, or they must all be character variables.

Numbered Variables

Numbered variables are used when the reason we collect them is to have elements of the same type together. The two main benefits that numbered variables provide are speed and flexibility.

The speed comes mainly from avoiding a great deal of repetition in the coding. The amount of repetition save depends, of course, on the size of the group you're dealing with: Numbered variables buys you nothing for a group of two variables. For 30 variables, the savings become very obvious.

The way the SAS system handles numbered variables provides you with a degree of flexibility not found in other languages or systems. The usual constraints of uniform size and placement that other systems impose do not apply to numbered variables in the SAS system.

The most useful flexibility for numbered variables is that they can be used in DATA steps or in PROC steps. Many SAS statements work with numbered variables:

- KEEP, DROP, & VAR statements determine the variables used in a PROC or DATA step. With numbered variables, we can easily specify variable groups or subgroups we want to process.

- INPUT & PUT statement communication values between the SAS system and the operating system it runs under. These statements are limited to the DATA step, but they provide an additional benefit. Numbered variables permit using factored INFORMAT or FORMATS to assure that values are handled uniformly.

- FORMAT, INFORMAT, LABEL & LENGTH statements control the attributes, or characteristics of variables within the SAS system. The SAS system permits numbered variables to share common attributes (like LENGTH or FORMAT), or have them assigned independently for each variable in the set.

Numbered variables may be defined in whatever sequence makes sense. We may choose any numbering scheme (with constraints) that fits what the data represents. But, unlike other systems, once we've defined a set of variables, the definition and order are carried through to every DATA or PROC step that uses them. But, we're not completely stuck with the choices made. We may specify a subset group very easily, change the order in which we reference, or deal with individual variables from the numbered set.
Two DATA step features that work well with numbered variables are DATA step functions and macros. Both features provide powerful techniques for dealing with groups of variables, and they are even more powerful when you combine them with numbered variables. (See Numbered Variable Goodies)

The ARRAY Statement

The ARRAY statement appears only in the DATA step. Just as a shopping list gathers items for a single purpose, the ARRAY statement gathers variables within a DATA step to perform a uniform step of tests or operations. And just as the items on a shopping list get put to different uses after we buy them, the variables defined to an ARRAY statement are bound to their common purpose only within the DATA step, and not in following steps.

We have great flexibility in deciding how to group a set of variables (See ARRAY Statement Goodies.)

Once we've defined an array, we reference array elements using the array name and a subscript value. The subscript value can be any form of a numeric expression that results in value valid for the array. The value must be an integer, and must be between one and number of elements in the array. This flexibility is what gives the ARRAY statement great power for good or evil.

Consider the alternatives

Arrays, in the form of numbered variables or in the form of an ARRAY statement make coding easier, when they are properly selected. Other techniques for collecting variable values may provide better results.

Whether we choose numbered variables or the ARRAY statement, we must still make a decision about how big the collection is. To enlarge the collection at a later time requires recoding. Instead of storing the collection within each observation, use a collection of observations to hold each set.

Instead of defining an observation as

```
CUSTOMER SALES1-SALES12
```

Include a SASdate value for each month, and use 12 observations:

```
CUSTOMER MONTH SALES
```

Defining the collection in this fashion allows you to determine the length of sales history by controlling the contents of the data set. If you wish to go from 12 months to 24 months, you can do so by simply including the observations for 24 months. There's no need to recode. Choosing this form instead of the array also lets you use time-series or analysis PROC's on the sets of months.

For other more complex ways of relating collections you may want to consider SAS/JML or using embedded pointer variables with direct access SAS data sets. Each of these choices has its own advantages and complications.

SUMMARY

There are two basic reasons for collecting things: because they are the same type of object, or because they will undergo the same processing.

The SAS system provides methods that match these reasons. Numbered variables collect variables of the same type. The ARRAY statement collects variables to used in the same tests or operations. Numbered variables define a collection that may be passed between DATA steps and between PROC steps; while variables are collected by ARRAY statements only in the DATA step that defines them.

Both numbered variables and the ARRAY statement provide flexible and compact ways to deal with collections of variables. Both methods also have their GOODIES and GOTCHA'S when you use them.
Numbered Variable GOODIES

- Numbered variables can be used in DATA steps or PROC steps.

Statements where numbered variables can be used:

```sas
INPUT AMT1-AMT30;
```

Instead of:

```sas
INPut AMT1 AMT2 AMT3 AMT4 AMT5 AMT6 AMT7 AMT8 AMT9 AMT10 AMT11 AMT12 AMT13 AMT14 AMT15 AMT16 AMT17 AMT18 AMT19 AMT20 AMT21 AMT22 AMT23 AMT24 AMT25 AMT26 AMT27 AMT28 AMT29 AMT30;
```

- Numbered variables can be used in whatever order is convenient:

```sas
INPUT AMT30-AMT1;
```

- Numbered variables provide speed and flexibility in describing or referencing groups of variables.

You can code:

```sas
NUMBER AMT1-AMT10;
```

Instead of:

```sas
INPut AMT1 AMT2 AMT3 AMT4 AMT5 AMT6 AMT7 AMT8 AMT9 AMT10 AMT11 AMT12 AMT13 AMT14 AMT15 AMT16 AMT17 AMT18 AMT19 AMT20 AMT21 AMT22 AMT23 AMT24 AMT25 AMT26 AMT27 AMT28 AMT29 AMT30;
```

- Numbered variables can be used in a set of numbered variables

```sas
PROC PRINT; VAR ACCT1-ACCT5 /* top 5 */;
```

- Numbered variables provide a compact, accurate way to assign uniform attributes to a set of variables:

```sas
LENGTH CUST1-CUST25 $ 20;
FORMAT REV1-REV12 DOLLAR11.2;
```

- Coding numbered variables with SAS functions can save large amounts of coding.

Instead of:

```sas
ARRAY HI{*} T1-T12;
DO I=1 TO DIM(HI);
IMPTOT = HI(I);
IF HI(I) NE . THEN
HI(I) = 1;
END;
AVG = IMPTOT / NT;
```

You can code:

```sas
AVG = MEAN(OF T1-T12);
```

As an exercise consider the code to replace:

```sas
/* Select salesmen with less */
/* than 4 months missing sales */
DATA;
SET SALESMAN;
IF NMISS(SALE1-SALE12) < 4;
```

Data step functions you can use with numbered variables:

- CSS CV KURTOSIS
- MAX MEAN MIN
- N NMISS RANGE
- SKEWNESS STD STDERR
- SUM US$ VAR

- Numbered variables provide a handy hook for macros:

```sas
%MACRO NAM2(OLD,NEW,DIM);
%DO I = 1 %TO &DIM;
&OLD&I = &NEW&I
%END;
%MEMO NAM2;
DATA ;
MERGE NOW LASTYR(
RENAME=(%NAM2(REV,OLDREV,12)))
BY CHAR;
RUN;
```

1233
ARRAY Statement GOODIES

- Syntax:

  ARRAY ARRAYNAME {*} VARLIST;

Where:

  ARRAYNAME is any valid variable name not already used to define a regular variable

  {*} designates the dimension for the array

  VARLIST identifies the elements referenced by ARRAYNAME

- Parentheses () may be used instead of braces {} when you're running SAS code on EBCDIC (IBM mainframe) systems.

  ARRAY ETC(S) UTIL RENT SUPPLIES;

- Any variables may be included in the element list as long as all variables are character or all variables are numeric.

  Individual variables may be used

    ARRAY REQO{*} NAME ADDRESS CITY STATE;

  Numbered variables may be used

    ARRAY JDATES{*} REA01'READ12 POST1·POST12;

  Individual variable names and numbered variables can be combined

    ARRAY NAME{*} SALESMAN CUSTMER1-CUSTMER8;

  Ranges may be used

    ARRAY NOMISS _NUMERIC_;

- Syntax Shortcut: The element list may be eliminated when it refers only to a set of numbered variables.

  You can write "ARRAY SALES(12);"

  Instead of "ARRAY SALES SALES1-SALES12;"

  The dimension may be specified only using a number in this case.

  Using the DIM function instead of a number lets you use an array even when the number of elements is not known

    /* Replace missing values with 0 */
    DATA NOMISS;
    SET RAW;
    ARRAY ALLNUM{*} _NUMERIC_;
    DO I = 1 TO DIM(ALLNUM);
    IF ALLNUM(I) = . THEN
      ALLNUM(I) = 0;
    END;
    RUN;

- The ARRAY statement allows you to collect variables that won't be used together any other place.

  /* DATA step code segment using array to verify all required variables have a value */
  ARRAY REQO{*} NAME STREET CITY STATE CODE;
  DO I = 1 TO DIM(REQO);
  IF REQO(I) = , THEN 00
    ENO;
  PUT 'VARIABLE MISSING VALUE'
    RECD{I }:
  DELETE;
  END;
  END;

- Using the same subscript in a statement or expression relates a group of arrays

  /* Data step code segment to compute */
  /* monthly gross profits for year */
  ARRAY SALES(12);
  ARRAY COST(12);
  ARRAY GROSS(12);
  DO I = 1 TO DIM(SALES);
  GROSS{I} = SALES{I} - COST{I};
  END;
  END;

- ARRAY statements let you use variables in a different order than the one they were defined in

  /* Data step code segment to compute */
  /* monthly gross profits for winter */
  /* season */
  ARRAY SALES(4) SALES12 SALES1-SALES3;
  ARRAY COST(4) COST12 COST11-COST3;
  ARRAY WGROSS(4);
  DO I = 1 TO 4;
  WGROSS{I} = SALES{I} - COST{I};
  END;

1234
Numbered variable GOTCHA's

- The 8 character length for SAS variable names must include the number.

If you were using ACCOUNT1-ACCOUNT8, changing it to ACCOUNT1-ACCOUNT15 would result in errors.

ARRAY Statement GOTCHA's

- If you use braces {} on an EBCDIC (IBM mainframe) system, they usually won't show on printed listings.

- If you use parentheses, using a DATA step function name as the name for an array disables the function.

- Using a number instead of the DIM function can create long-term maintenance problems.

  \[ \text{DO } I = 1 \text{ TO } 8; \text{ will only pull the first 8 elements, even if the list is expanded during later revisions.} \]

  \[ \text{DO } I = 1\text{ TO }\text{DIM(arrvar)}; \text{ will always process the whole array.} \]

- Incorrect setting of the subscript can produce the most subtle, nasty type of errors. BE CAREFUL.

  \[ \text{DO } I = 1 \text{ TO }\text{DIM(arrvar)}; \]

  /* The error in this code might let it run correctly, but it's likely to produce strange erratic results */

  /* What's the problem? I won't tell */

References

Version 5 Edition
pages 18, 42-50, 72-73, 116-117, 144, 149, 191-192, 230-231, 1204

SAS is the registered trademark of SAS Institute, Cary NC USA

SAS/IML™ is the trademark of SAS Institute, Cary NC USA

Author Contact

Bob Langill
Pacific Power
906 PSB
920 SW 6th
Portland OR, 97204