Introduction

When you need to perform the same action for a series of SCL variables, you can often simplify your programs by defining temporary groups of nonwindow (or program) variables called arrays. The differences between ARRAY statement execution in SCL and ARRAY statement execution in the DATA step are described in Chapter 7, “Using Other SAS Software Products,” on page 77.

Grouping Variables into Arrays

Listing variables in an ARRAY statement provides a way to refer to variables that are elements of the array. For example:

```
array month[5] $;
```

This statement produces character variables MONTH[1] through MONTH[5]. You can also use an ARRAY statement to list individual array elements. For example:

```
array month[5] $ jan feb mar apr may;
```

This form enables you to assign the names JAN, FEB, and so on to the array elements so that you can reference them in the SCL program.

You can also use the DECLARE statement to declare an array. This example declares an array named MONTH, which contains five character variables that are each up to three characters in length:

```
declare char(3) month[5];
```

This statement also produces variables MONTH[1] through MONTH[5]. In contrast to the ARRAY statement, the DECLARE statement cannot be used to list individual array elements. The following statement declares an array named MONTH plus five more character variables named JAN, FEB, MAR, APR, and MAY:
Arrays that are created with the DECLARE statement are temporary arrays. That is, they default to the _TEMPORARY_ option. For more information about temporary arrays, see “Using Temporary Arrays to Conserve Memory” on page 44.

To reference array elements, you can use the form array-name[position], where position is the index position of the variable in the array. This form of array reference is called subscripting. Subscripting is the only type of reference that you can use with arrays that were created with the DECLARE statement. For example, FACTOR[4] is the only way to reference the fourth element of array FACTOR if it is created with the statement

```
declare num Factor[5];
```

There are other ways to reference array elements. If an array is declared with an ARRAY statement that does not assign names to array elements (for example, `array month[5] $;`), then you can also reference the array elements as MONTH1, MONTH2, and so on. If an array is declared with an ARRAY statement that assigns names to the array elements (for example, `array month[3] $ jan feb mar;`), then you can also reference its elements by the assigned name. In this example, you can assign values to or retrieve values from the second element in array MONTH by using the array reference FEB or MONTH[2].

### Initializing Array Elements

You can define initial values for the elements of an array by listing the initial values in parentheses following the list of element names. Commas are optional between variable values. By default, all elements in a numeric array are initialized to numeric missing values if the array elements did not previously exist. For example, the following ARRAY statement creates a two-item array named COUNT:

```
array count[2] (1 2);
```

The ARRAY statement creates the elements and assigns them the initial values shown here:

```
count[1] = 1
count[2] = 2
```

You can also initialize array elements with the DECLARE statement. For example, the following program declares an array named MONTH, which contains five elements that can each contain three characters, and it assigns initial values to the array elements:

```
declare char(3) month[5] = ('jan' 'feb' 'mar'
   'apr' 'may');
```

```
INIT:
   put month;
return;
```

The example produces the following output:

```
month[1] = 'jan'
month[2] = 'feb'
month[3] = 'mar'
month[4] = 'apr'
month[5] = 'may'
```
Repeating an Action for Variables in an Array

To perform an action on variables in an array, you can use an iterative DO statement, using the index variable for the array subscript. A DO block is especially convenient when arrays contain many elements. For example, you could use a program like the following to sum the values of the array variables and to display the total in the SUM field:

```
array month[5] jan feb mar apr may (1,2,3,4,5);
INIT:
  do i=1 to 5;
    sum+month[i];
  end;
  put month;
  put sum=;
return;
```

The example produces the following output:

```
month[1] = 1
month[2] = 2
month[3] = 3
month[4] = 4
month[5] = 5
sum=15
```

The preceding DO block has the same effect as any one of the following assignment statements:

```
sum1=jan+feb+mar+apr+may;
sum2=sum(of month[*]);
sum3=sum(of jan--may);
put sum1= sum2= sum3= ;
```

This example produces the following output:

```
sum1=15 sum2=15 sum3=15
```

Grouping Variables That Have Sequential Names

If an application program or window has a series of variables whose names end in sequential numbers (for example, SCORE1, SCORE2, SCORE3, and so on), SCL enables you to group these variables into an array. For example, the following ARRAY statement groups the variables SCORE1, SCORE2, SCORE3, and SCORE4 into the array SCORE:

```
array score[4];
```

If the variables do not already exist as window variables, then SCL defines new, nonwindow, numeric variables with those names.
Using Temporary Arrays to Conserve Memory

If you want to use an array in an SCL program but do not need to refer to array elements by name, then you can add the _TEMPORARY_ argument to your ARRAY statement:

```
array total[4] _temporary_;
```

When you use the _TEMPORARY_ argument, you must use subscripting to refer to the array elements. For example, you must use TOTAL[2] to refer to the second element in the array TOTAL, defined above. You cannot use the variable name TOTAL2 as an alternative reference for the array element TOTAL[2]. Using the _TEMPORARY_ argument conserves memory. By default, SCL allocates memory for both the name of the array and the names of the individual array elements. However, when you use the _TEMPORARY_ argument, SCL allocates memory only for the array name. For large arrays, this can result in significant memory savings.

Note: Do not use the _TEMPORARY_ option if you plan to use the SET routine to read values from a SAS table directly into array elements. You must use the GETVAR or GETVARC function to read values from a SAS table into the elements of a temporary array.

Assigning the Same Value to Multiple Elements

You can use repetition factors to initialize arrays that are declared with the ARRAY and DECLARE statements. Repetition factors specify how many times the values are assigned in the array. They have the following form:

```
5 *( 2 3 4 )
```

In this example, 5 is the repetition factor and (2 3 4) is the list of initial values for the array elements. If the list consists of only a single item, then you can omit the parentheses.

For example, the following ARRAY statement and DECLARE statement both use repetition factors to initialize the values of the array REPEAT:

```
array repeat[17] (0,3*1,4*(2,3,4),0);
declare num repeat[17]=(0,3*1,4*(2,3,4),0);
```

This example repeats the value 1 three times and the sequence 2, 3, 4 four times. The following values are assigned to the elements of the array REPEAT:

0, 1, 1, 1, 2, 3, 4, 2, 3, 4, 2, 3, 4, 2, 3, 4, 0

Initializing Multidimensional Arrays

To initialize a multidimensional array, use the ARRAY or DECLARE statement to list values for the first row of the array, followed by values for the second row, and so on. The following examples both initialize a two-dimensional array named COUNT with two rows and three columns:

```
array count[2,3] (1 2 3 4 5 6);
decl num count[2,3]=(1 2 3 4 5 6);
```
Figure 4.1 on page 45 shows the values of the elements of this array.

**Figure 4.1**  Elements of the COUNT Array

<table>
<thead>
<tr>
<th>Rows</th>
<th></th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>column 1</td>
<td>column 2</td>
</tr>
<tr>
<td>row 1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>row 2</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

For more information about arrays, see “ARRAY” on page 248 and “DECLARE” on page 321.