INTRODUCTION

This is a tutorial introducing the reader to SAS DATA step arrays. But it is also a tutorial with other dimensions. For example, the vehicle for introducing arrays is a fairly typical happenstance: a simple information problem arises and is solved by a simple application. Subsequently a demand to enhance the solution necessitates change in the whole approach to the application. Part of that new solution path involves a careful look at what the data should look like, what is called in the jargon “Data Modeling.” Once that form is decided on, SAS DATA step arrays become the mechanics that help solve the problem. The paper concludes with yet another lesson: one should always be aware of different tools that can solve the same problem.

In the final step of making the desired report for the application, even though we could solve the needed data re-arrangement using arrays, we instead employ another handy tool, PROC TRANSPOSE, because it gets that job done in a simpler manner.

Arrays are a SAS Basic Language DATA-step feature useful for organizing, manipulating and performing calculations on groups of data element values. Multiple instances of data are handled by a single array name and one or more indices. From the early days of FORTRAN, arrays have been an important feature of procedural languages, later examples being PL/1, BASIC and PASCAL. In C, Java and JavaScript, the concept is still important. As with most other languages, an important thing to keep in mind is that arrays are a processing tool within the SAS DATA step and within Screen Control Language. In particular, they are not, in and of themselves, data on SAS Datasets, although in SCL they can be passed from one module to another using CALL DISPLAY.

Why use them? In a DATA-step or an SCL program, often one may find oneself in the situation of coding the same calculation on a series of SAS Variables, as for example these tax calculations below:

- `AutoSalesTax = SalesTaxRate * AutoSales`
- `LightTruckSalesTax = SalesTaxRate * LightTruckSales`
- `SUVSalesTax = SalesTaxRate * SUVSales`

For each of the original income items, we see what is essentially the same calculation being carried out: a fixed quantity (the tax rate) multiplied by an income figure to yield a tax amount. While repeating the calculation statement is not burdensome when there are just three such instances, it becomes more of a pain when there are fifty of them. The more wordy and laborious the program code is, the more the opportunity for variable misspellings and other silly errors, thus elongating the development process. After the program is done and running, such an approach can make maintenance changes similarly problematic. The use of arrays permits the collapse of such a series of statements into a single statement.

PAPER ORGANISATION

This paper will follow the development of an application, which starts simply as a quick and dirty request and which then expands in such a way as to demand a remedy to the growing volume of repetitious SAS Code. The solution path taken is the use of arrays, and both one- and two-dimensional arrays will be shown. In the course of the presentation, we will cover:

- How one defines arrays
- How one can use arrays to simply input from non-SAS files;
- How one can use arrays to organize and simplify calculations;
- How arrays can be useful (in place of lists) in certain functions.

The example will use both “temporary” and “permanent” arrays, along with the highly useful functions that reference array bounds. All discussion will relate to explicitly indexed arrays. SAS still supports a related concept, the “implicit” SAS array. Use of these is discouraged, however, and we will not cover this last topic.

INITIAL APPLICATION : A SIMPLE REPORT REQUEST

You are an analyst in the Corporate Planning unit of the Stag Insurance Company. One day, your boss, Mr. Fickle, drops by – you know, the fellow with the two pointy cones of black hair sticking up on either side of his head. He begins: “Heh, heh, heh” (He always begins his request with a little nervous nerve-wracking giggle.) “Heh, heh, heh. You know what would really hit the spot would be an historical summary of income taxes incurred on non-government taxable bond investments.”

It turns out that there are taxes incurred on interest and dividend income and on realized gains from investment turnover. From one of his worksheets, a friend in the Investment department cuts you a little flat file, called NESG9201.PRN, with the income organized by year and company subsidiary. This is shown at the top of the next page as Figure 1.0. In Figure 2.0 is shown SAS Code that uses a DATA step to produce the SAS Dataset of tax information and then uses PROC PRINT to create the report.
The solution to the problem is relatively straightforward, the only complication being the variability of the tax rate by year. Do note this calculation is separated into a subroutine and the routine’s reference using the LINK statement. The little report, shown in part as Figure 3.0 provides Mr. Fickle with the information he wanted.

Figure 1.1 Investment Income and Gains Data

<table>
<thead>
<tr>
<th>Year</th>
<th>Subsidy</th>
<th>Income</th>
<th>Tax Income</th>
<th>Gains</th>
<th>Tax Gains</th>
<th>Total Tax</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td>Haverford Fire</td>
<td>145,243</td>
<td>66,812</td>
<td>2,341</td>
<td>655</td>
<td>67,467</td>
</tr>
<tr>
<td>1984</td>
<td>Haverford Fire</td>
<td>154,974</td>
<td>71,288</td>
<td>2,458</td>
<td>-688</td>
<td>70,600</td>
</tr>
<tr>
<td>1985</td>
<td>Haverford Fire</td>
<td>165,358</td>
<td>76,065</td>
<td>2,581</td>
<td>723</td>
<td>76,787</td>
</tr>
<tr>
<td>1986</td>
<td>Haverford Fire</td>
<td>176,437</td>
<td>77,288</td>
<td>-2,458</td>
<td>70,600</td>
<td></td>
</tr>
<tr>
<td>1987</td>
<td>Haverford Fire</td>
<td>188,258</td>
<td>82,964</td>
<td>3,459</td>
<td>969</td>
<td>83,933</td>
</tr>
<tr>
<td>1988</td>
<td>Canaan Indemnity</td>
<td>376,789</td>
<td>173,323</td>
<td>1,189</td>
<td>333</td>
<td>173,656</td>
</tr>
<tr>
<td>1989</td>
<td>Canaan Indemnity</td>
<td>402,034</td>
<td>184,936</td>
<td>1,248</td>
<td>349</td>
<td>185,285</td>
</tr>
</tbody>
</table>

Figure 1.2 : Initial SAS Reporting Program

```sas
/* ******************************************** */
/* ** Step 1.0 : Read in Data, Calculate Tax ** */
/* ******************************************** */
DATA NESG2003.BONDTAX;
Infile 'D:\NESG2003\INDATA\NESG9201.PRN';
Input @2 YEARCHAR $CHAR8. @ ;
If YEARCHAR = ' ' Then Delete;
Format INCOME   GAINS    TAXINCME TAXGAINS TAXTOTAL COMMA8. ;
Format CALYEAR 4. ;
Input @6   CALYEAR   4. @11  SUBSIDCO $CHAR17. @28   INCOME  GAINS  ;
Link GETRATE;
TAXINCME = INCTXRAT * INCOME;
TAXGAINS = GNSTXRAT * GAINS;
TAXTOTAL = TAXINCME + TAXGAINS;
Output;
Return;
/* ******************************************** */
/* ** TAX RATE SELECTION SUBROUTINE ** */
/* ******************************************** */
DATA NESG2003.BONDTAX;
Infile 'D:\NESG2003\INDATA\NESG9201.PRN';
Input @2 YEARCHAR $CHAR8. @ ;
If YEARCHAR = ' ' Then Delete;
Format INCOME   GAINS    TAXINCME TAXGAINS TAXTOTAL COMMA8. ;
Format CALYEAR 4. ;
Input @6   CALYEAR   4. @11  SUBSIDCO $CHAR17. @28   INCOME  GAINS  ;
Link GETRATE;
TAXINCME = INCTXRAT * INCOME;
TAXGAINS = GNSTXRAT * GAINS;
TAXTOTAL = TAXINCME + TAXGAINS;
Output;
Return;
When ( CALYEAR = 1987 ) Do;
   INCTXRAT = .40;
   GNSTXRAT = .28;
   END;
Otherwise Do;
   INCTXRAT = .34;
   GNSTXRAT = .28;
   END;
End;
Run;
/* ******************************************** */
/* ** Step 2.0 : PRINT THE REPORT ** */
/* ******************************************** */
Title   "STAG INSURANCE COMPANY";
Title2  "ORIGINAL SIMPLE INVESTMENT SUMMARY"
PROC PRINT;
Id SUBSIDCO Notsorted;
Var CALYEAR INCOME TAXINCME GAINS TAXGAINS TAXTOTAL;
RUN;
/* ******************************************** */
```

Figure 1.3 – The Report (Sample Lines)

<table>
<thead>
<tr>
<th>Subsidy</th>
<th>Year</th>
<th>Income</th>
<th>Tax Income</th>
<th>Gains</th>
<th>Tax Gains</th>
<th>Total Tax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Haverford Fire</td>
<td>1983</td>
<td>145,243</td>
<td>66,812</td>
<td>2,341</td>
<td>655</td>
<td>67,467</td>
</tr>
<tr>
<td></td>
<td>1984</td>
<td>154,974</td>
<td>71,288</td>
<td>2,458</td>
<td>-688</td>
<td>70,600</td>
</tr>
<tr>
<td></td>
<td>1985</td>
<td>165,358</td>
<td>76,065</td>
<td>2,581</td>
<td>723</td>
<td>76,787</td>
</tr>
<tr>
<td></td>
<td>1986</td>
<td>176,437</td>
<td>77,288</td>
<td>-2,458</td>
<td>70,600</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1987</td>
<td>188,258</td>
<td>82,964</td>
<td>3,459</td>
<td>969</td>
<td>83,933</td>
</tr>
<tr>
<td>Canaan Indemnity</td>
<td>1983</td>
<td>376,789</td>
<td>173,323</td>
<td>1,189</td>
<td>333</td>
<td>173,656</td>
</tr>
<tr>
<td></td>
<td>1984</td>
<td>402,034</td>
<td>184,936</td>
<td>1,248</td>
<td>349</td>
<td>185,285</td>
</tr>
</tbody>
</table>

etc
THE APPLICATION GETS BIGGER: NON-ARRAY SOLUTION USED

As with so many little requests, successfully addressing an information need expands that need. Into your cubicle walks Mr. Fickle. “Heh, heh, heh. You know, I’ve been thinking about that little tax report,” says he. “What would really hit the spot would be a breakout of the taxes by type of bond, as well as providing the totals that you gave me.”

It turns out that there are five taxable bond categories in which Fickle is interested: Utilities, Transportation, Industrial, Financial Institutions and C.M.O.’s (collateralized Mortgage Obligations). You go back to your friend in Investment, and he provides you with a second file, NESG9202.PRN, containing the broken out information. A few records are displayed below, with the data lines wrapped. The order of the data is: five instances of income by bond type, followed by five instances of gains information by bond type.

<table>
<thead>
<tr>
<th>Year</th>
<th>Bond Type</th>
<th>Income</th>
<th>Gains</th>
<th>Total Income</th>
<th>Total Gains</th>
</tr>
</thead>
<tbody>
<tr>
<td>1983</td>
<td>HAVERFORD FIRE</td>
<td>25272</td>
<td>407</td>
<td>25779</td>
<td>348</td>
</tr>
<tr>
<td>1984</td>
<td>HAVERFORD FIRE</td>
<td>26966</td>
<td>-428</td>
<td>26538</td>
<td>-376</td>
</tr>
<tr>
<td>1985</td>
<td>HAVERFORD FIRE</td>
<td>28772</td>
<td>449</td>
<td>29221</td>
<td>415</td>
</tr>
<tr>
<td>1986</td>
<td>HAVERFORD FIRE</td>
<td>30700</td>
<td>364</td>
<td>31064</td>
<td>382</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Being a conservative soul, you set about solving the problem by expanding the program you wrote to address the first request, and the data step component of the code now expands to the following.

```
DATA NESG2003.BONTAX2;
  Infile 'D:\NESG2003\INDATA\NESG9202.PRN';
  Input @2 YEARCHAR $char8. @ ;
  If YEARCHAR = ' ' Then DELETE;
  Input @6   CALYEAR   4.  @11  SUBSIDCO $CHAR17. @28  INCOM_UTILS INCOM_TRANS INCOM_INDUS INCOM_FINCL INCOM_CMO;
  GAINS_UTILS GAINS_TRANS GAINS_INDUS GAINS_FINCL GAINS_CMO;
  Link GETRATE;
  * ....... Utility Bonds .................;
  INCTX_UTILS = INCTXRAT * INCOM_UTILS;
  GNSTX_UTILS = GNSTXRAT * GAINS_UTILS;
  TOTTX_UTILS = INCTX_UTILS + GNSTX_UTILS;
  * ....... Transportation Bonds ..........
  INCTX_TRANS = INCTXRAT * INCOM_TRANS;
  GNSTX_TRANS = GNSTXRAT * GAINS_TRANS;
  TOTTX_TRANS = INCTX_TRANS + GNSTX_TRANS;
  * ....... Industrial Bonds .............
  INCTX_INDUS = INCTXRAT * INCOM_INDUS;
  GNSTX_INDUS = GNSTXRAT * GAINS_INDUS;
  TOTTX_INDUS = INCTX_INDUS + GNSTX_INDUS;
  * ....... Finan Institution Bonds .......
  INCTX_FINCL = INCTXRAT * INCOM_FINCL;
  GNSTX_FINCL = GNSTXRAT * GAINS_FINCL;
  TOTTX_FINCL = INCTX_FINCL + GNSTX_FINCL;
  * ....... C.M.O.s ........................
  INCTX_CMO  = INCTXRAT * INCOM_CMO ;
  GNSTX_CMO  = GNSTXRAT * GAINS_CMO ;
  TOTTX_CMO  = INCTX_CMO + GNSTX_CMO ;
  Drop YEARCHAR ;
  Output;
  Return;
/* ******************************************** */
/* ** TAX RATE SELECTION SUBROUTINE ** */
/* ******************************************** */
GETRATE:
  Select;
  When  CALYEAR LT 1987 ) Do;
    INCTXRAT = .46;
    GNSTXRAT = .28;
  End;
  When( CALYEAR =  1987 ) Do;
    INCTXRAT = .40;
    GNSTXRAT = .28;
  End;
  Otherwise Do;
    INCTXRAT = .34;
    GNSTXRAT = .28;
  End;
  Return;
Run;
```

Clearly the corresponding PROC PRINT would expand considerably. We’ve not shown it here, but you can picture what it looks like. By now there are enough variables to eat up more than a reasonable line size, so that the report would “wrap” if we try to print everything in a single report. One alternative solution to that is to make three PROC PRINTS out of it: one for Income Taxes, one for Realised Gains and one for Total Taxes. Even though we took advantage of SAS version 8.0 variable lengths to make the additional variables clear in their meaning, the report looks rather ugly with those names as column labels. After a review of the first cut and his usual opening giggle, Mr. Fickle notes that the reports really don’t look pretty, and so we...
introduce an intermediate PROC Datasets execution to assign some SAS Variable Labels and formats to address Mr. fickle’s complaints. This step is shown below, along with one of the three PROC PRINT’s:

**Figure 2.3 Assigning Labels and Formats (To Make Report Clearer)**

```sas
PROC DATASETS Library=NESEG2003;
Modify BONDTAX2;
Format CALYEAR 4.;
****************************************;
Format INCOM_UTILS INCOM_TRANS INCOM_INDUS INCOM_FINCL INCOM_CMO comma9.;
Format GAINS_UTILS GAINS_TRANS GAINS_INDUS GAINS_FINCL GAINS_CMO comma9.;
Format INCTX_UTILS INCTX_TRANS INCTX_INDUS INCTX_FINCL INCTX_CMO comma9.;
Format GNSTX_UTILS GNSTX_TRANS GNSTX_INDUS GNSTX_FINCL GNSTX_CMO comma9.;
Format TOTTX_UTILS TOTTX_TRANS TOTTX_INDUS TOTTX_FINCL TOTTX_CMO comma9.;
Label  INCOM_UTILS = 'UTILITIES INCOME';
Label  INCOM_TRANS = 'TRANSPRTN INCOME';
Label  INCOM_INDUS = 'INDUSTR''L INCOME';
Label  INCOM_FINCL = 'FINANCIAL INCOME';
Label  INCOM_CMO   = 'C.M.O.-BD INCOME';
Label  INCOM_TOTAL = '**TOTAL** INCOME';
Label  GAINS_UTILS = 'UTILITIES REAL GAIN';
Label  GAINS_TRANS = 'TRANSPRTN REAL GAIN';
Label  GAINS_INDUS = 'INDUSTR''L REAL GAIN';
Label  GAINS_FINCL = 'FINANCIAL REAL GAIN';
Label  GAINS_CMO   = 'C.M.O.-BD REAL GAIN';
Label  GAINS_TOTAL = '**TOTAL** REAL GAIN';
Label  INCTX_UTILS = 'UTILITIES INCME TAX';
Label  INCTX_TRANS = 'TRANSPRTN INCME TAX';
Label  INCTX_INDUS = 'INDUSTR''L INCME TAX';
Label  INCTX_FINCL = 'FINANCIAL INCME TAX';
Label  INCTX_CMO   = 'C.M.O.-BD INCME TAX';
Label  INCTX_TOTAL = '**TOTAL** INCME TAX';
Label  GNSTX_UTILS = 'UTILITIES GAINS TAX';
Label  GNSTX_TRANS = 'TRANSPRTN GAINS TAX';
Label  GNSTX_INDUS = 'INDUSTR''L GAINS TAX';
Label  GNSTX_FINCL = 'FINANCIAL GAINS TAX';
Label  GNSTX_CMO   = 'C.M.O.-BD GAINS TAX';
Label  GNSTX_TOTAL = '**TOTAL** GAINS TAX';
Label  TOTTX_UTILS = 'UTILITIES TOTAL TAX';
Label  TOTTX_TRANS = 'TRANSPRTN TOTAL TAX';
Label  TOTTX_INDUS = 'INDUSTR''L TOTAL TAX';
Label  TOTTX_FINCL = 'FINANCIAL TOTAL TAX';
Label  TOTTX_CMO   = 'C.M.O.-BD TOTAL TAX';
Label  TOTTX_TOTAL = '**TOTAL** TOTAL TAX';
Run;
Quit;
:::;
/* ******************************************** */
/* ** Step 2.2 : PRINT THE REPORT            ** */
/* ******************************************** */
PROC PRINT Label Data=NESEG2003.BONDTAX2;
Title4 'REALIZED GAINS';
Id  SUBSIDCO;
By  SUBSIDCO Notsorted;
Var CALYEAR GNSTX_UTILS GNSTX_TRANS GNSTX_INDUS GNSTX_FINCL GNSTX_CMO GNSTX_TOTAL;
Run;
```

To use the variable labels, all one has to do is add the option LABEL to the PROC PRINT statement, as shown on the example producing the Realised Gains tax info.

It is time to step back to assess what we’ve done. The solution certainly lacks the compact feel of the original SAS program. The variates (“analysis variables”) have expanded from eight to 30. The DATA step has expanded from 32 lines (excluding comments) to 55 lines. The newly added PROC DATASETS execution adds another 47 lines. There are now three PROC PRINTS instead of one. But at least the job is done. Or is it?

**THE APPLICATION GETS BIGGER STILL : WATERSHED**

In walks Mr. Fickle and says he: “Heh, heh, heh.. You know, I’ve been thinking about that little tax report. What would really hit the spot would be to add to your report some other sources of Investment return -- the tax free bonds and the stocks.”

It turns out that there are two categories within each of these two types of investments, depending on the purchase date of the stocks or bonds – before or after the 1986 Tax Reform act. This raises the number of investment vehicles from five to nine. Not including the item listing on the DATA-step input statement, this raises the number of analysis variables to 52, 20 lines will be added to the PROC DATASETS, and the DATA step code will now double, in part due to the number of variables and in part due to more complicated tax rate considerations. Ye gads!

**RETHINKING THE APPLICATION : DATA CONSIDERATIONS**

Your friend in the Investment area agrees to cut you another version of the source data, NESG92BF, with the additional income and gains information for the additional investments. These will be additional columns on the file, which (except for the expansion) looks like the one shown in Figure 2.1.

It is probably not the best idea to further expand the application the way you just did in the second iteration. What you need to do is stand back and apply a little “data modeling” to eliminate the proliferation of SAS Variables that is complicating the program. If you examine the second iteration of this exercise, you should see in the data two **explicit keys** on the records: the subsidiary (SUBSIDCO) and the Calendar Year (CALYEAR). But in that same data, there is also an **implicit hidden key**: the
**type of investment**, of which there were five on the second iteration and of which there will be nine on the newly requested version. Taking this perspective, the actual number of analysis variables is really only five in number:

- Investment Income
- Realised Capital Gains
- Taxes on Investment Income
- Taxes on Realised Gains
- Total Federal Taxes

We are going to reconfigure the file to make the **investment type** an **explicit key**. This will thereby cut the file down so that it has the following structure:

<table>
<thead>
<tr>
<th>Category</th>
<th>Varname</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys</td>
<td>CALYEAR</td>
<td>4.</td>
<td>Calendar Year</td>
</tr>
<tr>
<td></td>
<td>SUBSIDCO</td>
<td>char17.</td>
<td>Subsidiary Company Name</td>
</tr>
<tr>
<td></td>
<td>ITYPE</td>
<td>Z2.</td>
<td>Investment Vehicle ID Code</td>
</tr>
<tr>
<td></td>
<td>INVSNAME</td>
<td>char17.</td>
<td>Investment Vehicle Description</td>
</tr>
<tr>
<td>Variates</td>
<td>INVINCOM</td>
<td>comma9.</td>
<td>Investment Income (Interest and Dividends)</td>
</tr>
<tr>
<td></td>
<td>REALGAIN</td>
<td>comma9.</td>
<td>Realised Capital Gains</td>
</tr>
<tr>
<td></td>
<td>INCOMTAX</td>
<td>comma9.</td>
<td>Taxes On Investment Income</td>
</tr>
<tr>
<td></td>
<td>GNSTAX</td>
<td>comma9.</td>
<td>Taxes on Realised Capital Gains</td>
</tr>
<tr>
<td></td>
<td>TOTALTAX</td>
<td>comma9.</td>
<td>Total Federal Taxes</td>
</tr>
</tbody>
</table>

Except for the two additional keys, this looks suspiciously like the simple SAS Dataset of the first iteration. The Investment ID code ITYPE and the corresponding set of Descriptions will carry the following values.

<table>
<thead>
<tr>
<th>ITYPE</th>
<th>INVSNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Utility Bonds</td>
</tr>
<tr>
<td>2</td>
<td>Transport’n Bonds</td>
</tr>
<tr>
<td>3</td>
<td>Industrial Bonds</td>
</tr>
<tr>
<td>4</td>
<td>Finan-Inst Bonds</td>
</tr>
<tr>
<td>5</td>
<td>C.M.O. Bonds</td>
</tr>
<tr>
<td>6</td>
<td>Pre-TRA Muni-Bonds</td>
</tr>
<tr>
<td>7</td>
<td>Proration Muni-Bonds</td>
</tr>
<tr>
<td>8</td>
<td>Pre-TRA Stocks</td>
</tr>
<tr>
<td>9</td>
<td>Proration Stocks</td>
</tr>
<tr>
<td>10</td>
<td>Total All Investments</td>
</tr>
</tbody>
</table>

The sharp eye will notice that having both a code value (ITYPE) and also a description (INVSNAME) is a redundancy: the INVSNAME is really an "alternate Key", and technically it is not needed on the record to identify the data line. However, as you will see, the description variable INVSNAME will prove very useful when it comes time to make the report.

In any case, the revised DATA Statement needed to produce this new version of the output is going to be:

```
DATA NESG2003.BONDTAX4
  (KEEP = SUBSIDCO CALYEAR ITYPE INVSNAME INVINCOM REALGAIN INCOMTAX GNSTAX TOTALTAX )
```

Each input data record will cause ten output records of the simplified type described in Figure 3.1. If you want to get a heads-up on what’s coming, take a peek at the code in Figure 4.0, at the end of this paper.

**INTRODUCING SOME ONE DIMENSIONAL PERMANENT ARRAYS**

We will simplify the calculations in the DATA-step by using arrays to overlay the respective income, gains and taxes by investment type. We will need at least five of these arrays, and each is defined using the ARRAY statement:

```
Array INCOME (9) T1-T9: /* Interest Income */
Array GAINS  (9) S1-S9: /* Realised Gains. */
Array INCTAX (9) U1-U9: /* Tax on Interest */
Array GNSTAX (9) V1-V9: /* Tax on Gains... */
Array TOTTAX (9) W1-W9: /* Tax: Total..... */
```

Here is a very strong suggestion: **place these statements up front in the DATA step** -- as you would with LENGTH and FORMAT Statements -- before the procedural code.

In each case the name of the array appears after the keyword ARRAY, and this name, in turn, is followed by the dimension of the array in parentheses brackets or braces. For a permanent array, the dimension is followed by the SAS Variables the Array
elements overlay. The data above is all numeric, but, as we shall see later on, you can also define arrays of character elements. Each item overlaid corresponds to a pigeon-hole in the array. So for example

- \text{INCOME}(1) \text{ is the same as } T1
- \text{INCOME}(2) \text{ is the same as } T2
- \text{INCOME}(3) \text{ is the same as } T3

and so on. The variables overlaid do not have to be in the form of an indexed list like these are. We could have defined an array overlaying the five income variables in the second iteration of our application as follows:

```
Array INCOME_X(5) INCOM_UTILS INCOM_TRANS INCOM_INDUS INCOM_FINCL INCOM_CMO;
```

In this case

- \text{INCOME_X}(1) \text{ is the same as } INCOM_UTILS
- \text{INCOME_X}(2) \text{ is the same as } INCOM_TRANS
- \text{INCOME_X}(3) \text{ is the same as } INCOM_INDUS

Since the number of elements dictates the size of the array here, we can forego coding “9” as the dimension in the arrays on the preceding page, since it is redundant in the specification. For one-dimensional permanent arrays like these, you may dummy code the array’s dimension with an asterisk as shown below

```
Array INCOME (*) T1-T9; /* Interest Income */
Array GAINS  (*) S1-S9; /* Realised Gains. */
Array INCTAX (*) U1-U9; /* Tax on Interest */
Array GNSTAX (*) V1-V9; /* Tax on Gains... */
Array TOTTAX (*) W1-W9; /* Tax: Total..... */
```

As a matter of programming philosophy, the less you hardcode, the less you have to change in the program when the Mr. Fickle’s of the world change their minds. And they will as surely as the tax season cometh.

Had I wanted to initialize the array elements to zero, I could do that in the array statement by following the variable list with initial values as per

```
Array INCOME (*) T1-T9 (0 0 0 0 0 0 0 0 0); /* Interest Income */;
```

Initialising the array like this has a curious side effect: it is equivalent to putting a RETAIN statement on the underlying variables, in this case T1-T9.

**USING ONE DIMENSIONAL ARRAYS TO SIMPLIFY THE INPUT**

Having introduced these arrays, the input statement now becomes

```
Input @6 CALYEAR
@11 SUBSIDCO
@28 INCOME(*)
   GAINS(*);
```

This is equivalent to the following:

```
Input @6 CALYEAR
@11 SUBSIDCO
@28 T1-T9
   S1-S9;
```

Both are pretty compact, but the first does have one advantage: having set up the array, you won’t have to change the input statement if the number of investments again increases and your friend in Investment supplies the same type of file. And clearly, had the INCOME array been something more akin to INCOME_X, which overlaid a list of SAS Variables, the use of the array rather than the list does make for a very much more concise INPUT expression.

Similar to the INPUT example above, you can also use the arrays to simplify output. Here you might want to include a specific format, and below is an example of how such an output statement might look:

```
Put @6 CALYEAR 4.
@11 SUBSIDCO $char17.
@28 INCOME(*) (comma8. +1)
   GAINS(*) (comma8. +1);
```

In this statement, the elements of the array are written out in comma edited format, and “+1” is a one position skip between the output fields that ensures at least one blank separates the numbers.
USING ONE DIMENSIONAL ARRAYS AND DO-LOOPS TO SIMPLIFY CALCULATIONS

We can use the arrays we have defined to simplify the tax calculations as follows, using an iterative DO-loop:

\[
\text{Do } JJ = 1 \text{ to } 9 \\
\quad \text{INCTAX}(JJ) = \text{INCRATE} \times \text{INCOME}(JJ); \\
\quad \text{GNSTAX}(JJ) = \text{GNSRATE} \times \text{GAINS}(JJ); \\
\quad \text{TOTTAX}(JJ) = \text{INCTAX}(JJ) + \text{GNSTAX}(JJ) \\
\text{End;}
\]

If you are unfamiliar with DO-loops, note the use of the DO-loop counter JJ. The coded statements between the DO and END statements will execute 9 times, JJ increasing by 1 each time. For example, in the third iteration, the following statements will be executed:

\[
\text{INCTAX}(3) = \text{INCRATE} \times \text{INCOME}(3); \\
\text{GNSTAX}(3) = \text{GNSRATE} \times \text{GAINS}(3) \\
\quad \text{TOTTAX}(3) = \text{INCTAX}(3) + \text{GNSTAX}(3)
\]

and this is equivalent to:

\[
U3 = \text{INCRATE} \times T3; \\
V3 = \text{GNSRATE} \times S3; \\
W3 = U3 + V3;
\]

The upper bound of the DO-loop counter redundantly specifies the dimension of the array. It is logical to ask if we can avoid this redundancy. The answer is "yes". SAS provides a handy function called DIM( ) which uses an array name as an argument and returns the dimension of the array. We can substitute a use of this for the hard-coded upper bound of the loop.

\[
\text{Do } JJ = 1 \text{ to Dim(INCOME)} \\
\quad \text{INCTAX}(JJ) = \text{INCRATE} \times \text{INCOME}(JJ); \\
\quad \text{GNSTAX}(JJ) = \text{GNSRATE} \times \text{GAINS}(JJ); \\
\quad \text{TOTTAX}(JJ) = \text{INCTAX}(JJ) + \text{GNSTAX}(JJ) \\
\text{End;}
\]

In this case it does not matter which array we use as the DIM function argument, since all have the same dimension of 9.

Unfortunately, we are not quite done here. Although the above is syntactically correct, our solution here is not quite correct from the business perspective. The problem is that the income tax rate is going to vary, not only with the year (which it did before), but also with the type of investment. We shall have to revisit these calculations before we are done.

USING PERMANENT ARRAYS AND FUNCTIONS TO SIMPLIFY CALCULATIONS

We are going to want some totals across the investment categories. These will be

\[
\begin{align*}
\text{INCOMSUM} & \quad \text{Total Interest and Dividend Income} \\
\text{GAINSSUM} & \quad \text{Total Realised Capital Gains} \\
\text{INCTXSUM} & \quad \text{Total Tax on Investment Income} \\
\text{GNSTXSUM} & \quad \text{Total Tax on Realised Gains} \\
\text{TOTTXSUM} & \quad \text{Total Federal Investment Taxes}
\end{align*}
\]

Here is one solution, using a DO-loop to iteratively add the contributions to the totals:

\[
\text{INCOMSUM} = 0; \\
\text{GAINSSUM} = 0; \\
\text{INCTXSUM} = 0; \\
\text{GNSTXSUM} = 0; \\
\text{TOTTXSUM} = 0; \\
\text{Do } JJ = 1 \text{ to Dim(INCOME)} \\
\quad \text{INCOMSUM} = \text{INCOMSUM} + \text{INCOME}(JJ); \\
\quad \text{GAINSSUM} = \text{GAINSSUM} + \text{GAINS}(JJ); \\
\quad \text{INCTXSUM} = \text{INCTXSUM} + \text{INCTAX}(JJ); \\
\quad \text{GNSTXSUM} = \text{GNSTXSUM} + \text{GNSTAX}(JJ); \\
\quad \text{TOTTXSUM} = \text{TOTTXSUM} + \text{TOTTAX}(JJ); \\
\text{End;}
\]

There is nothing wrong with this solution: it is fairly typical of what you would do had you been solving the problem in FORTRAN. But, in this case, there is a quicker (and non-procedural) way to code the solution. Use the SUM( ) function.

\[
\begin{align*}
\text{INCOMSUM} & = \text{Sum(of INCOME(*));} \\
\text{GAINSSUM} & = \text{Sum(of GAINS(*));} \\
\text{INCTXSUM} & = \text{Sum(of INCTAX(*));}
\end{align*}
\]
The SUM() function can be used with a discrete list of arguments, separated by commas. In the case of the contents of the array INCOME, the following would yield the same answer:

\[\text{INCOMSUM} = \text{Sum} (\text{T1, T2, T3, T4, T5, T6, T7, T8, T9});\]

When an array or an indexed list is used for the argument, the syntax changes and the parameter keyword “OF” precedes the array name or indexed list as per:

\[\text{INCOMSUM} = \text{Sum} (\text{of T1-T9});\]

or

\[\text{INCOMSUM} = \text{Sum} (\text{of INCOME(*)});\]

Some other SAS Functions that can be used in this way with arrays or indexed lists are:

- MEAN  Arithmetic average
- STD  Standard Deviation
- VAR  Variance
- RANGE  Range

Taking stock of what we have done so far: If Mr. Fickle demands a further expansion of the number of investment categories – and he will as sure as the sun will rise – none of the following recoded areas of this program will need to be changed:

The INPUT Statement
The DO loop containing the tax calculations
The Formations of the Totals using the SUM() function.

The only maintenance vulnerability so far is the number of elements underlying the five arrays introduced so far.

INTRODUCING A CHARACTER ARRAY WITH THE INVESTMENT DESCRIPTION

Next we introduce an array holding the Investment Type descriptions. These will be supplied to the output variable INVSNOME when we create the output file:

```sas
Array CATEGORY(10) $17 _temporary_
{ "Utility Bonds"  "Transprt Bonds"
 "Industr’l Bonds"  "Financ’l Bonds"
 "C.M.O. Bonds"  "Pre-TRA Muni-Bnd"
 "Prorat’n Bonds"  "Pre-TRA Stocks"
 "Prorat’n Stocks"  "*Total* Invstmts"};
```

Note that this is a character array – the "$" declares it thusly. A length of “17” is specified here: if we do not, the elements of the array will default to a length of 8 characters, too small to hold the descriptive labels.

Note also that the attribute TEMPORARY is embraced by underbars and note the absence of an underlying SAS variable list. This is a temporary array, not a permanent array. Note that CATEGORY does not overlay any SAS Variables, unlike INCOME and its four cousins.

Clearly since there are no underlying elements, temporary arrays need to be explicitly dimensioned. But a temporary array works well here because our uses of it will not require any underlying list of SAS Variables. One advantage of temporary arrays arises from the way SAS implements them as part of compiling a program: the storage locations are contiguous, and thus processing a temporary array is somewhat faster than processing a permanent array. A question should pop into your mind at this point: why did we go to the trouble of making INCOME and its four cousins five permanent arrays in the first place. We never really do make use of the underlying variables. The answer is that, had we not made those arrays permanent, we could not have used the asterisk notation which abbreviated the INPUT statement and we could not have made use of the SUM() function the way we did. It’s a trade-off.

Once final point, in case it was not obvious: the contents of temporary arrays are “retained” throughout the iterations of the DATA-step

INTRODUCING A TWO-DIMENSIONAL ARRAY TO ORGANISE OUTPUT

We’ve streamlined the input, calculation and summation parts of the program. We now turn to making the output file. Recall that this file, which we are calling BONDTAX4, is pretty simple in nature – see Figure 3.1. We begin laying the groundwork by slotting the five output variables in a one dimensional permanent array called (not surprisingly) OUTDATA:

```sas
Array OUTDATA (*) INVINCOM REALGAIN INCOMTAX GAINSTAX TOTALTAX
```
The structure we will utilize to load up OUTDATA will be the following two-dimensional array:

```
Array ALLDATA (5, 10)
  T1-T9  INCOMSUM;  /* Interest Income */
  S1-S9  GAINSSUM   /* Realised Gains. */
  U1-U9  INCTXSUM   /* Tax on Interest */
  V1-V9  GNSTXSUM   /* Tax on Gains.. */
  W1-W9  TOTTXSUM ; /* Tax: Total..... */
```

Notice that array ALLDATA overlays the same storage as do the arrays INCOME, GAINS, INCTAX, GNSTAX and TOTTAX, as well as the sums of the contents of these five arrays. We could not pull off this trick with temporary arrays. The definition shows clearly that **permanent arrays can overlay non-contiguous storage**, a feature not present in older procedural languages like FORTRAN.

Observe the syntax of the declaration. A comma separates the bounds of the respective dimensions. The first index will iterate over the five types of items. The second index will iterate over the instances of these items: nine investment types plus the total across these. If you picture a two-dimensional array as a table or grid, the first counter can be viewed as the “row number” and the second counter as the “column numbers.”

One thing you will observe is the absence of asterisks in the declaration. Common sense should tell you why. Despite the very suggestive way we laid out the elements in the declaration that the array overlays, this layout is a purely cosmetic device in our display of the code. Looking at the totality of the items overlaid, there is no way for SAS to know whether we intended that the array be 5 by 10, 10 by 5 or 25 by 2. **Two-dimensional arrays must be explicitly dimensioned.**

Here is how we can use the arrays to produce the ten output records from each input record:

```
Do ITYPE = 1 To 10; /* By Investment Type */
  Do ITEM = 1 To 5; /* Variate or Data item */
    OUTDATA(ITEM) = ALLDATA(ITYPE,ITEM);
  End;
  INVSNAME = CATEGORY(ITYPE);
  Output NESG2003.BONDTAX4;
End;
```

The outer DO-Loop iterates by type of investment and produces each output record. The OUTPUT statement is necessary since we want to create multiple records for each iteration of the DATA-step. The inner DO-Loop loads up the contents of the variates in the output data vector. You can see now where the character array CATEGORY is used: it provides the description in INVSNAME using a simple array reference.

As in the tax calculations, we again have DO-Loops with hard-coded bounds and again we ask if there is some way to avoid them. The answer is “yes”. When applied to a two-dimensional array, our friend `DIM()` returns the size of the first dimension, in this case “5”. Its first cousin, `DIM2()`, when applied to array OUTDATA, will return the size of the second dimension, in this case “10”. We can thus set out a cleaned up version of the double DO loop coding that eliminates a maintenance vulnerability:

```
Do ITYPE = 1 To Dim2(ALLDATA); /* By Investment Type */
  Do ITEM = 1 To Dim(ALLDATA); /* Variate or Data item */
    OUTDATA(ITEM) = ALLDATA(ITYPE,ITEM);
  End;
  INVSNAME = CATEGORY(ITYPE);
  Output NESG2003.BONDTAX4;
End;
```

We are almost done. We only need to fix the tax calculations.

**INTRODUCING ARRAYS WITH LOWER BOUNDS OTHER THAN ONE – TAX RATES:**

Recall that Tax Rates are an as yet unresolved problem. In the original application, which dealt only with taxable bonds, the rates simply varied by year. Remember the little subroutine `GETRATE`? But in the larger context of all types of bonds plus stocks, these rates vary by year and also by type of investment. Hard-coding this more complex situation in the program has some real downsides, not only in terms of the complexity of -- or laborious nature of -- the code itself, but also in terms of maintenance vulnerability of the code to any updates as rates change in the future. (See the SAS DATA-step code in figures 1.2 and 2.2.) One beneficial change would be to eliminate the hard-coded tax rates and make use of a table of rates, i.e. a program input. Such a table is easily built and we will call this table `TAXTABLE`. Its contents are shown in table 3.2.

We have three problems to solve:

- How to store the tax rates in the program;
- How to load the tax rates into that storage; and
- How to apply the tax rates to the income and gains data;
The last consideration will drive the other two. What we are shooting for is a calculation routine that should look something like the following:

```
Do ITYPE = 1 to Dim(INCOME)
   INCTAX(ITYPE) = INCRATE(CALYEAR) * INCOME(ITYPE);
   GNSTAX(ITYPE) = GNSRATE(CALYEAR) * GAINS(ITYPE);
   TOTTAX(ITYPE) = INCTAX(ITYPE) + GNSTAX(ITYPE);
End;
```

where INCRATE and GNSRATE would be arrays indexed by calendar year. This suggests defining the arrays in the following manner:

```
Array INCRATE(1995) _temporary_;
Array GNSRATE(1995) _temporary_;
```

This definition, although syntactically correct, is rather silly: we have defined two arrays with roughly 2000 elements in them but we only plan to use ten or twelve from 1983 on. SAS offers a better solution: you can define arrays with both upper AND LOWER bounds:

```
Array INCRATE(1983:1995) _temporary_;
```

Note the syntax: a colon separates the upper and lower bounds. With this set-up, the GNSRATE array holding the gains tax rates by year will work just fine in the DO-loop on the preceding page. But the INCRATE array above is not adequate since the rates also vary by type of investment. There are five variations or “tax categories”, as one can see from Figure 3.1. Both the hold array and the DO-Loop will need to be amended to take this into account. Here is the revised pair of array definitions:

```
Array INCRATE(1983:1995, 5) _temporary_;
```

INCRATE is now two-dimensional, the second dimension being the taxation category. As you can see from TAXTABLE in figure 3.2, there are FIVE of these.

### Figure 3.2 – Layout of Tax Rate Table TAXTABLE

<table>
<thead>
<tr>
<th>Category</th>
<th>Varname</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys</td>
<td>CALYEAR</td>
<td>4</td>
<td>Calendar Year</td>
</tr>
<tr>
<td>Variates</td>
<td>R_INTRST_DIVD</td>
<td>percent7.2</td>
<td>Standard Corporate Tax Rate</td>
</tr>
<tr>
<td></td>
<td>R_PRETRA_MUNI</td>
<td>percent7.2</td>
<td>Pre-TRA Muni Bond Rate</td>
</tr>
<tr>
<td></td>
<td>R_PRORTN_MUNI</td>
<td>percent7.2</td>
<td>Post-TRA Muni Bond Rate</td>
</tr>
<tr>
<td></td>
<td>R_PRETRA_STKS</td>
<td>percent7.2</td>
<td>Pre-TRA Equity Investment Rate</td>
</tr>
<tr>
<td></td>
<td>R_PRORTN_STKS</td>
<td>percent7.2</td>
<td>Post-TRA Equity Investment Rate</td>
</tr>
<tr>
<td></td>
<td>R_REALZD_GNS</td>
<td>percent7.2</td>
<td>Capital Gains Rate</td>
</tr>
</tbody>
</table>

Before we revise the tax calculation DO-loop, let’s deal with bringing the tax rates into the program. We need to input these prior to processing any of the investment data and we want to input these once and only once. The multiple income tax rates suggest a permanent array for the five of them, like:

```
Array FEDINCRATES(*) R_INTRST_DIVD R_PRETRA_MUNI R_PRORTN_MUNI R_PRETRA_STKS R_PRORTN_STKS
```

The input routine can be isolated into a subroutine, rather like the following:

```
TAXTABLE:
   Do While ( Not(ENDDATA) AND
      (CALYEAR LE 1995) )
      Set NESG2003.TAXTABLE End=ENDDATA;
   If CALYEAR GE 1983 Then Do;
      Do TXCATEG = 1 to 5;
         INCRATE(CALYEAR, TXCATEG) = FEDINCRATES(TXCATEG);
      End;
      GNSRATE(CALYEAR) = R_REALZD_GNS;
   If ENDDATA and CALYEAR LT 1995 then do;
      Put / @5 "** Error ** Bad Tax Table. Last Year is " CALYEAR /;
      Stop;
   End;
```
End;
Return;

The subroutine will be invoked once in the first iteration of the DATA-step, as per.

If _N_ = 1 Then Link TAXTABLE;

The form of the coding merits some comment. Our Data happens to run from 1983 to 1995. But the table TAXTABLE should actually have rates both prior to 1983 and also after 1995, if the table is current. If the table fails to have rates up to the date of interest (here 1995), it is an error, and in this case there is no sense continuing the execution of the program, hence the error message and "stop". Also, given our array definitions for INCRATE and GNSRATE, if we attempted to load a tax rate for a date prior to 1983 or a date subsequent to 1995, the program would abend with a "subscriptrange" error.  SUBSCRIPTRANGE is an attempt to reference a SAS Array element outside the Array bounds. Avoiding such an indelicate abend is the reason for both IF statement and the DO WHILE condition.

Again we see hard-coded array bounds in the DO-loops and again we ask if there is some way to eliminate them. The DIM( ) function will take care of the inner loop, but the other references represent a different problem. Our friend DIM( ) won't help us there since it would simply return the number of elements in the lower-bounded arrays. So for this type of situation, SAS supplies the functions LBOUND( ) and HBOUND( ). This will result in a modified version of the code above

TAXTABLE:
Do While ( Not(ENDDATA) AND
      (CALYEAR LE Hbound(GNSRATE) )
    Set NESG2003.TAXTABLE End=ENDDATA;
If CALYEAR GE Lbound(GNSRATE) Then Do;
  Do TXCATEG = 1 to Dim(FEDINCRATES);
    INCRATE(CALYEAR, TXCATEG) = FEDINCRATES(TXCATEG);
  End;
  GNSRATE(CALYEAR) = R_REALZD_GNS;
If ENDDATA and CALYEAR LT Hbound(GNSRATE) then do;
  Put / @5 "** Error ** Bad Tax Table. Last Year is " CALYEAR /;
  Stop;
End;
End;
Return;

USING AN ARRAY AS A FUNCTION: FIXING THE TAX RATE CALCULATION:

The remaining item to be fixed is the Tax Calculation. With the INCRATE array now two-dimensional to take into account the Tax category, the code now looks some like this

Do ITYPE= 1 to Dim(INCOME)
  INCTAX(ITYPE) = INCRATE(CALYEAR, ??? ) * INCOME(ITYPE);
  GNSTAX(ITYPE) = GNSRATE(CALYAR) * GAINS(ITYPE);
  TOTTAX(ITYPE) = INCTAX(ITYPE) + GNSTAX(ITYPE)
End;

where the question marks are a place holder for “the proper tax category”. Here is one solution to the problem using a SELECT group:

Do ITYPE= 1 to Dim(INCOME)
  Select( ITYPE )
    When (1,2,3,4,5)   ITXTYPE = 1;
    When (6)           ITXTYPE = 2;
    When (7)           ITXTYPE = 3;
    When (8)           ITXTYPE = 4;
    When (9)           ITXTYPE = 5;
    Otherwise          ITXTYPE = 1;
  End;
  INCTAX(ITYPE) = INCRATE(CALYEAR, ITXTYPE ) * INCOME(ITYPE);
  GNSTAX(ITYPE) = GNSRATE(CALYAR) * GAINS(ITYPE);
  TOTTAX(ITYPE) = INCTAX(ITYPE) + GNSTAX(ITYPE)
End;

While a respectable solution to the problem, it does raise a red flag. If the investment and tax categories proliferate in a subsequent expansion of this request, the code will have to be revisited and modified. To make the code immune to such changes and localize the impact of future maintenance, we will instead introduce one final temporary array, TAXCATEG:

Array TAXCATEG(9) _temporary_ (1 1 1 1 1 2 3 4 5);
We will use the array TAXCATEG as a function mapping investment type id to the tax category:

\[
\text{Do ITYPE= 1 to Dim(INCOME)} \\
\quad \text{If ITYPE le Dim(TAXCATEG)} \\
\quad \quad \text{Then ITXTYPE = TAXCATEG(ITYPE);} \\
\quad \text{Else DO; Put / @5 "** Error ** TAXCATEG Needs Enlarging" /; STOP;} \\
\quad \text{End;} \\
\quad \text{INCTAX(ITYPE) = INCRATE(CALYEAR, ITXTYPE) * INCOME(ITYPE);} \\
\quad \text{GNSTAX(ITYPE) = GNSRATE(CALYAR) * GAINS(ITYPE);} \\
\quad \text{TOTTAX(ITYPE) = INCTAX(ITYPE) + GNSTAX(ITYPE)} \\
\text{End;} \\
\]

The IF-statement prevents an uncontrolled program blow-up when the number of investments increases but the array TAXCATEG is inadvertently not changed to reflect that increase.

**HOUSEKEEPING: LABELS AND FORMATS FOR THE OUTPUT FILE:**

This is now trivial: the PROC DATASETS execution needed to assign labels and formats to the data on output file BONDTAX4 is as follows:

```plaintext
PROC DATASETS Library=NESG2003; 
Modify BONDTAX4 
  Attrib CALYEAR Format=4. Label="Calen Year"; 
  Attrib SUBSIDCO Format=$char17. Label="Subsidiary"; 
  Attrib ITYPE Format=Z2. Label="Inv Typ"; 
  Attrib INVSNAME Format=$char17. Label="Investment Descriptn"; 
  Attrib INVINCOM Format=comma9. Label="Interest Dividends"; 
  Attrib REALGAIN Format=comma9. Label="Realised Cap-Gains"; 
  Attrib TOTALTAX Format=comma9. Label="Fed-Taxes Total"; 
Run; Quit;
```

**BEHOLD THE FINAL HANDIWORK:**

If you now go to Figure 4.0, you can see in this two column exhibit the totality of the code we built using arrays to cut down on code volume and to make the application an easier one to work with, as future enhancements are requested. Expanding the number of investment categories will not cause this program to enlarge a bit. Only some changes to the array bounds are required.

**CREATING REPORTS FROM THE DATA:**

The file we produced is simple in nature, and it is easy to create a formatted dump of the data using PROC REPORT in the manner below:

```plaintext
PROC REPORT Data=NESG2003.BONDTAX4 Split=' ' 
  Colwidth=9 Nowindows Headline Missing; 
Column SUBSIDO CALYEAR ITYPE INVSNAME INCOMTAX INCOMTAX 
  REALGAIN GAINSTAX TOTALTAX; 
By SUBSIDCO CALYEAR NOTSORTED; 
Define SUBSIDCO /Order width=18; 
Define CALYEAR /Order width=6; 
Define ITYPE /Display width=4; 
Define INVSNAME /Display width=6; 
Define INVINCOM /Display width=9; 
Define REALGAIN /Display width=9; 
Define INCOMTAX /Display width=9; 
Define GAINSTAX /Display width=9; 
Define TOTALTAX /Display width=9; 
RUN;
```

This is but an slightly enhanced version of the simple PROC PRINTS we used before. The report is shown at the top of the next page as Figure 3.3. But this report is really just a cosmetically enhanced data dump, and it is not going to cut it for Mr. Fickle and the higher-ups he reports to. A more usual -- and more useful -- type of request would be a report that looks like the one depicted in Figure 5.0 attached at the end of the paper. In this latter report, the data on the file is rotated. On the
file, the data items are columns and the Investment types are row instances. In the rotated report, on the other hand, the investment vehicles are the columns and the data items (the taxes) form the rows.

### Figure 3.3: A Simple Report from the Data

<table>
<thead>
<tr>
<th>Subsidiary Company</th>
<th>Calen Year</th>
<th>Invst Type Code</th>
<th>Investment Description</th>
<th>Interest Income</th>
<th>Investmt Capital Gains</th>
<th>Realized Gains</th>
<th>Total Investmt Taxes</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORDENFELDT LTD</td>
<td>1989</td>
<td>2</td>
<td>Transpnt Bonds</td>
<td>44,233</td>
<td>15,039</td>
<td>348</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Industr'l Bonds</td>
<td>141,484</td>
<td>48,105</td>
<td>1,114</td>
<td>379</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>Financ'l Bonds</td>
<td>25,411</td>
<td>8,640</td>
<td>200</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>C.M.O. Bonds</td>
<td>47,998</td>
<td>16,319</td>
<td>378</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>Pre-TRA Muni-Bnd</td>
<td>65,018</td>
<td>0</td>
<td>933</td>
<td>317</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>ProRat'n Muni-Bn</td>
<td>15,054</td>
<td>768</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>Pre-TRA Stocks</td>
<td>5,742</td>
<td>586</td>
<td>3,021</td>
<td>1,027</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>ProRat'n Stocks</td>
<td>3,319</td>
<td>457</td>
<td>1,087</td>
<td>370</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td><em>Total</em> All-Inv</td>
<td>402,845</td>
<td>108,473</td>
<td>7,511</td>
<td>2,554</td>
</tr>
<tr>
<td>1990</td>
<td>1</td>
<td>Utility Bonds</td>
<td>61,846</td>
<td>21,028</td>
<td>-451</td>
<td>-153</td>
<td>20,874</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Transpnt Bonds</td>
<td>50,116</td>
<td>17,039</td>
<td>-366</td>
<td>-124</td>
<td>16,915</td>
</tr>
</tbody>
</table>

How one transforms the data from the form we created it to the form desired in such a report can be another exercise using arrays. Actually, we are not going to use that approach here, but we will leave it as an exercise for the reader to construct a DATA-Step to make the transformation. Instead, we are going to make use of an often overlooked and very handy SAS procedure, PROC TRANSPOSE.

The actual code necessary to rotate the data is as follows:

```sas
PROC SORT Data=NESG2003.BONDTAX4
    Out=NESG92D1;
    By SUBSIDCO CALYEAR ITYPE;
Run;
/* ************************************************** */
PROC TRANSPOSE Data=NESG92D1
    Out=NESG92D2
    Label=ITEMNAME
    Prefix=INVS;
    By      SUBSIDCO CALYEAR;
    Id      ITYPE;
    Idlabel INVSNAME;
    Var     INCOMTAX GAINSTAX TOTALTAX;
Run;
```

The sorting is simply a good idea simply to make sure that the data is in the BY-order expected in the TRANSPOSE step. The rotation will be performed in data clusters defined by the BY sort-break criteria. For those unfamiliar with the tool, TRANSPOSE is going to create a file with the following Variables on it:

**BY Variables**
- SUBSIDCO: the subsidiary company
- CALYEAR: the calendar reporting year

**Row Ident**
- ITEMNAME: This will carry the descriptive Label associated with the Financial Item SAS variables on the source file, namely the tax items listed in the VAR stmt

**Variates**
- INVS1-INVS10: This indexed List is defined by the row instances of the Investment Vehicles on the source file. The indexing comes from the values of variable ITYPE. The output SAS variables are formed from that index and the prefix "INVS".

The LABEL option defines a variable that will carry the former variable labels of the input SAS Dataset. The PREFIX option and the ID statement define the new form of the columns on the output file. The IDLABEL statement defines a variable on the source file whose values will constitute the SAS Labels for these new columns. You may recall the discussion about
investment description variable INVSNAM E and its apparent redundancy. Although we did make use of INVSNAME in the data dump report, a more interesting employment is made here make the columns in the "landscape" data report (Figure 5.0) self explanatory.

One the data is rotated, the report in Figure 5.0 is generated by the following execution of PROC REPORT:

```
Title "STAG INSURANCE COMPANY";
Title2 "Rotated Investment Taxes Summary";
Title3 "Company Subsidiary : #byval(SUBSIDCO)"
PROC REPORT Data=NESG92D2
Colwidth=9 Nowindows
Split=' ' Headline Missing;
******************************************;
By SUBSIDCO;
Column SUBSIDCO CALYEAR ITEMNAME INVS1-INVS10;
Define SUBSIDCO /order noprint;
Define CALYEAR /order width=5;
Define ITEMNAME /display width=22;
Define INVS1-INVS10 /display width=8;
******************************************;
Break after CALYEAR / Skip ;
Format INVS1-INVS10 comma7.;
Run;
```

and with this our mission is accomplished.

**POST-MORTEM:**

Having gone to all the trouble of minimizing areas of maintenance, by using arrays and then minimizing the actual appearance of the bounds on these arrays, can't we somehow also eliminate the bounds in the declarations, for example the explicit extents? The answer is that you most surely can, but accomplishing that gets into the use of SAS Macros, a topic which is outside the scope of this tutorial.

While you are musing about that possibility, in walks Mr. Fickle. “Heh, heh, heh. You know I've been thinking about that little investment report. What would really hit the spot would be ……” No sweat, this time!

```
************************************************
```

**REFERENCES:**

SAS is a registered Trademark of the SAS Institute, Inc of Cary N.C.

Arrays are documented in the *SAS Language Reference Dictionary, Version 8*, published by the SAS Institute, Inc., Cary N.C.

PROC TRANSPOSE is documented in the *SAS Procedures Guide, Version 8*, published by the SAS Institute, Inc., Cary N.C.

The paper “Working With Arrays” has been previously published in an earlier form in both the NESUG 1992 Proceedings and also the SUGI Proceedings for 1994 and 1995. The current version reflects changes in the language with the advent of SAS version 8.2.

**CONTACT INFORMATION**

Ralph Leighton works for The Hartford Insurance Financial Services Group, in Hartford, Conn., where he has been employed in various areas of Actuarial and Information Management. He is currently in the Corporate Actuarial Department in charge of automation support for the Corporate Reserving function. He may be reached using the following

E-mails: rleighton@thehartford.com and rleighton@cox.net
Phone: (860) 547-3014
Figure 4.0 : Final Version Of DATA=step Code, Using Arrays

```/* *************************************** */
/* ** Step 1. Read in Data and Calc Tax ** */
/* *************************************** */
DATA NESG2003.BONDTAX4
(KEEP = SUBSIDCO CALYEAR ITYPE
 INVSNAME INVINCOM REALGAIN INCOMTAX GAINSTAX TOTALTAX );
LENGTH INVSNAME $ 16;
/* *** Array Declarations **************** */
/* ** For Input, Calculation, Summing **** */
Array  INCOME (*) T1-T9; /* Inv Income  */
Array  GAINS  (*) S1-S9; /* Real Gains  */
Array  INCTAX (*) U1-U9; /* Incm Taxes  */
Array  GNSTAX (*) V1-V9; /* Gains Taxes */
Array  TOTTAX (*) W1-W9; /* Total Taxes */
** To Map Invstment ID to Description.* */
Array  CATEGORY (10) $ 17
("Utility Bonds"     "Transpnl Bonds"
"Industr'l Bonds"   "Financ'nl Bonds"
"C.M.O. Bonds"     "Pre-TRA Muni-Bnds"
"ProRat'n Muni-Bnds" "Pre-TRA Stocks"
"ProRat'n Stocks"    "*Total* All-Inv");
/* ** For the Output Routine ************ */
Array  ALLDATA (5,10)
T1-T9  INCOMSUM  /* Investmt Income */
S1-S9  GAINSSUM  /* Realised Gains */
U1-U9  INCTXSUM  /* Income Taxes */
V1-V9  GNSTXSUM  /* Gains Taxes */
W1-W9  TAXESSUM; /* Total Taxes */
Array  OUTDATA(*)   INVINCOM  REALGAIN
INCOMTAX  GAINSTAX  TOTALTAX;
/* ** For Tax Table Input and Storage *** */
Array  INCRATE(1983:1995,5) _temporary_;
Array  FEDINCRATES(*)   R_INTRST_DIVD
R_PRETRA_DIVD  R_PRETRA_MUNI  R_PRETRA_MUNI
R_PRETRA_STKS  R_PRETRA_STKS;
/* ** To Map the Tax Rate Categ to Inv ID */
Array  TAXCATEG(9)   _temporary_(1 1 1 1 1 2 3 4 5);
/* ********** Processing Code ********** */
/* ** On First Pass Get the Tax Table *** */
If _N_ = 1 Then Link TAXTABLE;
/* ** Get Investment Return Data ********** */
Infile 'D:\NESG2003\INDATA\NESG92BF.DAT';
Input @6  CALYEAR 4.
@11  SUBSIDCO $CHAR16.
@28  INCOME(*)
@100  GAINS(*) ;
/* ** Calculate Federal Taxes *********** */
Do ITYPE = 1 To Dim(INCOME);
TXTYPE = TAXCATEG(ITYPE);
INCTAX(ITYPE) = INCRATE(CALYEAR,TXTYPE)*INCOME(ITYPE);
GNSTAX(ITYPE) = GNSRATE(CALYEAR) * GAINS(ITYPE);
TOTTAX(ITYPE) = INCTAX(ITYPE) + GNSTAX(ITYPE);
End;
/* ** Form Totals across Investments ** */
INCOMSUM = Sum(of INCOME(*));
GAINSSUM = Sum(of GAINS(*));
INCTXSUM = Sum(of INCTAX(*));
GNSTXSUM = Sum(of GNSTAX(*));
TAXESSUM = Sum(of TOTTAX(*));
/* ** Create Output, a Rec per Invest ** */
Do ITYPE = 1 To Dim(ALLDATA);
Do ITEM = 1 To Dim(ALLDATA);
OUTDATA(ITEM) = ALLDATA(ITEM,ITYPE);
INVSNAME = CATEGORY(ITYPE);
Output NESG2003.BONDTAX4;
End;
Return;
/* ** SUBROUTINE : IMPORT TAX TABLE *** */
TAXTABLE:
Do Until( (CALYEAR=HBOUND(GNSRATE))
  Or ENDDATA );
Set NESG2003.TAXTABLE End=ENDDATA;
If CALYEAR >= LBOUND(GNSRATE) Then Do;
  Do CATEG = 1 To Dim(FEDINCRATES);
    INCRATE(CALYEAR,CATEG) = FEDINCRATES(CATEG);
  End;
  GNSRATE(CALYEAR) = R_REALSD_GNS;
End;
If CALYEAR LT Hbound(GNSRATE)
or ENDDATA Then do;
  Put / @5 "* Err * Bad Tax Table. **"
  / @20 "Last Year is " CALYEAR /;
  Stop;
End;
End;
Return;
RUN;
/* ****************************************************** */
/* ** Step 2. Assign Labels, Formats ** */
/* ****************************************************** */
PROC DATASETS Library=NESG2003 Nolist;
Modify BONDTAX4
( Label='Reconfig Tax Data');
Format CALYEAR 4.  ITYPE 2.;
Format INVINCOM REALGAIN INCOMTAX
GAINSTAX TOTALTAX comma9. ;
Label CALYEAR = 'Calen Year'
SUBSIDCO = 'Subsidiary Company'
INVSNAME = 'Investment Descriptn'
ITYPE = 'Invst Type Code'
Label INVINCOM = 'Interest Dividends'
REALGAIN = 'Realized Gains'
INCOMTAX = 'Invstmt Income Taxes'
GAINSTAX = 'Realised Gains Taxes'
TOTALTAX = 'Total Investmt Taxes';
Run; Quit

GNSTAX(ITYPE) =
GNSRATE(CALYEAR) * GAINS(ITYPE);
TOTTAX(ITYPE) =
INCTAX(ITYPE) + GNSTAX(ITYPE);
End;
*/```
### Figure 5.0: Tax Information Report Using Transposed Data

<table>
<thead>
<tr>
<th>Calen Year</th>
<th>VARIABLE</th>
<th>Utility</th>
<th>Transpn</th>
<th>Industr'</th>
<th>Financ'l</th>
<th>C.M.O.</th>
<th>Pre-TRA</th>
<th>ProRat'n</th>
<th>Pre-TRA</th>
<th>PreRat'n</th>
<th><em>Total</em>*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Bonds</td>
<td>Bonds</td>
<td>Bonds</td>
<td>Bonds</td>
<td>Bonds</td>
<td>Muni-Bnd</td>
<td>Muni-Bnd</td>
<td>Stocks</td>
<td>Stocks</td>
<td>All-Inv</td>
</tr>
<tr>
<td>1983</td>
<td>Investmt Income Taxes</td>
<td>30,158</td>
<td>24,438</td>
<td>78,169</td>
<td>14,039</td>
<td>26,519</td>
<td>0</td>
<td>0</td>
<td>603</td>
<td>0</td>
<td>173,926</td>
</tr>
<tr>
<td></td>
<td>Realised Gains Taxes</td>
<td>58</td>
<td>47</td>
<td>150</td>
<td>27</td>
<td>51</td>
<td>256</td>
<td>0</td>
<td>533</td>
<td>0</td>
<td>1,122</td>
</tr>
<tr>
<td>1983</td>
<td>Total Investmt Taxes</td>
<td>30,216</td>
<td>24,485</td>
<td>78,319</td>
<td>14,066</td>
<td>26,570</td>
<td>256</td>
<td>0</td>
<td>1,136</td>
<td>0</td>
<td>175,048</td>
</tr>
<tr>
<td>1984</td>
<td>Investmt Income Taxes</td>
<td>32,179</td>
<td>26,076</td>
<td>83,406</td>
<td>14,980</td>
<td>28,295</td>
<td>0</td>
<td>0</td>
<td>622</td>
<td>0</td>
<td>185,558</td>
</tr>
<tr>
<td></td>
<td>Realised Gains Taxes</td>
<td>61</td>
<td>49</td>
<td>158</td>
<td>28</td>
<td>53</td>
<td>-269</td>
<td>0</td>
<td>597</td>
<td>0</td>
<td>677</td>
</tr>
<tr>
<td>1984</td>
<td>Total Investmt Taxes</td>
<td>32,240</td>
<td>26,125</td>
<td>83,563</td>
<td>15,008</td>
<td>28,349</td>
<td>0</td>
<td>-269</td>
<td>0</td>
<td>1,219</td>
<td>0</td>
</tr>
<tr>
<td>1985</td>
<td>Investmt Income Taxes</td>
<td>34,335</td>
<td>27,823</td>
<td>88,994</td>
<td>15,984</td>
<td>30,191</td>
<td>0</td>
<td>0</td>
<td>641</td>
<td>0</td>
<td>197,968</td>
</tr>
<tr>
<td></td>
<td>Realised Gains Taxes</td>
<td>-64</td>
<td>-52</td>
<td>-165</td>
<td>-30</td>
<td>-56</td>
<td>283</td>
<td>0</td>
<td>668</td>
<td>0</td>
<td>584</td>
</tr>
<tr>
<td>1985</td>
<td>Total Investmt Taxes</td>
<td>34,271</td>
<td>27,771</td>
<td>88,829</td>
<td>15,954</td>
<td>30,134</td>
<td>-269</td>
<td>0</td>
<td>1,310</td>
<td>0</td>
<td>198,552</td>
</tr>
<tr>
<td>1986</td>
<td>Investmt Income Taxes</td>
<td>36,635</td>
<td>29,687</td>
<td>94,957</td>
<td>17,055</td>
<td>32,214</td>
<td>0</td>
<td>0</td>
<td>661</td>
<td>39</td>
<td>211,248</td>
</tr>
<tr>
<td></td>
<td>Realised Gains Taxes</td>
<td>67</td>
<td>54</td>
<td>174</td>
<td>31</td>
<td>59</td>
<td>297</td>
<td>0</td>
<td>748</td>
<td>0</td>
<td>1,430</td>
</tr>
<tr>
<td>1986</td>
<td>Total Investmt Taxes</td>
<td>36,702</td>
<td>29,741</td>
<td>95,131</td>
<td>17,086</td>
<td>32,373</td>
<td>297</td>
<td>0</td>
<td>1,410</td>
<td>39</td>
<td>212,678</td>
</tr>
<tr>
<td>1987</td>
<td>Investmt Income Taxes</td>
<td>33,991</td>
<td>27,544</td>
<td>88,103</td>
<td>15,824</td>
<td>29,889</td>
<td>0</td>
<td>1,036</td>
<td>790</td>
<td>361</td>
<td>197,539</td>
</tr>
<tr>
<td></td>
<td>Realised Gains Taxes</td>
<td>-70</td>
<td>-57</td>
<td>-183</td>
<td>-33</td>
<td>-62</td>
<td>-311</td>
<td>0</td>
<td>838</td>
<td>0</td>
<td>122</td>
</tr>
<tr>
<td>1987</td>
<td>Total Investmt Taxes</td>
<td>33,921</td>
<td>27,487</td>
<td>87,921</td>
<td>15,791</td>
<td>29,578</td>
<td>-311</td>
<td>1,036</td>
<td>1,629</td>
<td>361</td>
<td>197,661</td>
</tr>
<tr>
<td>1988</td>
<td>Investmt Income Taxes</td>
<td>30,828</td>
<td>24,982</td>
<td>79,905</td>
<td>14,351</td>
<td>27,108</td>
<td>0</td>
<td>942</td>
<td>857</td>
<td>470</td>
<td>179,442</td>
</tr>
<tr>
<td></td>
<td>Realised Gains Taxes</td>
<td>-90</td>
<td>-73</td>
<td>-233</td>
<td>-42</td>
<td>-79</td>
<td>397</td>
<td>0</td>
<td>1,140</td>
<td>123</td>
<td>1,144</td>
</tr>
<tr>
<td>1988</td>
<td>Total Investmt Taxes</td>
<td>30,738</td>
<td>24,909</td>
<td>79,673</td>
<td>14,309</td>
<td>27,029</td>
<td>397</td>
<td>942</td>
<td>1,997</td>
<td>592</td>
<td>180,586</td>
</tr>
<tr>
<td>1989</td>
<td>Investmt Income Taxes</td>
<td>29,287</td>
<td>23,732</td>
<td>75,910</td>
<td>13,634</td>
<td>25,752</td>
<td>0</td>
<td>1,008</td>
<td>728</td>
<td>568</td>
<td>170,620</td>
</tr>
<tr>
<td></td>
<td>Realised Gains Taxes</td>
<td>94</td>
<td>77</td>
<td>244</td>
<td>44</td>
<td>83</td>
<td>417</td>
<td>0</td>
<td>1,277</td>
<td>459</td>
<td>2,695</td>
</tr>
<tr>
<td>1989</td>
<td>Total Investmt Taxes</td>
<td>29,381</td>
<td>23,809</td>
<td>76,155</td>
<td>13,678</td>
<td>25,835</td>
<td>417</td>
<td>1,008</td>
<td>2,005</td>
<td>1,027</td>
<td>173,315</td>
</tr>
<tr>
<td>1990</td>
<td>Investmt Income Taxes</td>
<td>27,823</td>
<td>22,546</td>
<td>72,115</td>
<td>12,952</td>
<td>24,465</td>
<td>0</td>
<td>1,079</td>
<td>619</td>
<td>688</td>
<td>162,285</td>
</tr>
<tr>
<td></td>
<td>Realised Gains Taxes</td>
<td>-99</td>
<td>-80</td>
<td>-257</td>
<td>-46</td>
<td>-87</td>
<td>-438</td>
<td>-31</td>
<td>1,085</td>
<td>514</td>
<td>562</td>
</tr>
<tr>
<td>1990</td>
<td>Total Investmt Taxes</td>
<td>27,724</td>
<td>22,466</td>
<td>71,858</td>
<td>12,906</td>
<td>24,378</td>
<td>-438</td>
<td>1,048</td>
<td>1,704</td>
<td>1,202</td>
<td>162,847</td>
</tr>
<tr>
<td>1991</td>
<td>Investmt Income Taxes</td>
<td>29,686</td>
<td>24,056</td>
<td>76,946</td>
<td>13,820</td>
<td>26,104</td>
<td>0</td>
<td>1,154</td>
<td>526</td>
<td>832</td>
<td>173,125</td>
</tr>
<tr>
<td></td>
<td>Realised Gains Taxes</td>
<td>104</td>
<td>84</td>
<td>269</td>
<td>48</td>
<td>91</td>
<td>460</td>
<td>227</td>
<td>922</td>
<td>576</td>
<td>2,782</td>
</tr>
<tr>
<td>1991</td>
<td>Total Investmt Taxes</td>
<td>29,790</td>
<td>24,141</td>
<td>77,215</td>
<td>13,868</td>
<td>26,195</td>
<td>460</td>
<td>1,381</td>
<td>1,448</td>
<td>1,408</td>
<td>175,907</td>
</tr>
</tbody>
</table>