ABSTRACT
This paper describes several independent ideas that perhaps should be in the toolkit of intermediate level programmers. We cover methods of supplying missing summarization levels using SAS 8.2 PROC MEANS options and the DATA step UPDATE statement. We examine a nifty example of adverse event reporting by month. We suggest an alternative to standard if-then-else logic, the DATA step SELECT statement. We review Proc Tabulate and a macro to obtain a quick glance at all data sets in a directory. We also present some thoughts on formats and a few other ideas along the way.

INTRODUCTION
Like most people, we programmers can get in a rut. Of course there is merit to not fixing something if it is not broken. At the same time, no one wants to be unable to learn tricks, old or new. This motivated the following few ideas that we hope are profitable for review.

1. SUPPLYING MISSING LEVELS AND UPDATING VARIABLES.
We begin with the raw data and a summary and then demonstrate three methods of adding missing levels. Note that three sites and two types are represented in this raw data.

<table>
<thead>
<tr>
<th>Obs</th>
<th>site</th>
<th>type</th>
<th>var1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

proc means data=one noprint;
class site type;
var var1;
output out=one_0(where=(_type_=3)) sum=sum;
run;

<table>
<thead>
<tr>
<th>Obs</th>
<th>site</th>
<th>type</th>
<th><em>TYPE</em></th>
<th><em>FREQ</em></th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1.1 Creating all possible combinations of the class values using raw data.
Let’s suppose we want to summarize this data by site and type. However, we want to explicitly show in our summary both types for each site, even though our raw data does not contain this information. By specifying COMPLETETYPES in the MEANS procedure we are able to show site=2 type=2 and site=3 type=1 even though these particular levels do not appear in the raw data. This is reflected in observations 4 and 5, where _FREQ_=0.

proc means data=one noprint completetypes;
class site type;
var var1;
output out=one_1(where=(_type_=3)) sum=sum;
run;

<table>
<thead>
<tr>
<th>Obs</th>
<th>site</th>
<th>type</th>
<th><em>TYPE</em></th>
<th><em>FREQ</em></th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
1.2 Creating all possible combinations of the class values using formats.
Let's suppose we still want to summarize this data by site and type, but we know that a type exists that is not represented in the data. We can force this to occur by specifying a format for type. The summary below shows type=3 (formatted as 'C') in observations 3, 6, and 9, even though type=3 does not appear in the raw data for any level of site.

```
proc format;
  value type 1='A' 2='B' 3='C';
run;

proc means data=one noprint completetypes;
  class site type;
  var var1;
  output out=one_2(where=(_type_=3)) sum=sum;
  format type type.;
run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>site</th>
<th>type</th>
<th><em>TYPE</em></th>
<th><em>FREQ</em></th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>A</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>B</td>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>C</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>A</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>B</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>C</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>A</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>B</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>C</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

1.3 Updating data with the DATA step UPDATE statement.
Of course an older alternative to the above methods is to create a template data set and then merge or update this with a summarized data set. Below, such a template data set, called z, is created and is then combined with output from the original, plain PROC MEANS above using the DATA step UPDATE statement.

```
data z;
  do site=1 to 3;
    do type=1 to 3;
      sum=0;
      output;
    end;
  end;
run;

data one_3;
  update z one_0;
  by site type;
run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>site</th>
<th>type</th>
<th>sum</th>
<th><em>TYPE</em></th>
<th><em>FREQ</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
2. A DIFFERENT LOOK AT ADVERSE EVENTS.

Here’s a fun way to look at adverse events (AE), different from standard AE tables. Suppose we are asked to summarize AEs by month including the number of days on study drug, the number of days sick, and the number of AEs started during each month. The following three blocks of code comprise the source data in the form a programmer is likely to receive it: begin and end dates. We are assuming three subjects, each on study drug the same 10 days. There are four AEs in the study; each subject has at least one. The subject with more than one AE has them overlapping four days and starting on the same day. Note the DO loops distributing activity over the proper days.

```
data two_study_drug;
  retain SDSTDT '23Oct2003'd SDENDT '01Nov2003'd;
  do id=1 to 3;
    output;
  end;
run;
data two_on_study(drop=SD:);
  set two_study_drug;
  do date=SDSTDT to SDENDT;
    output;
  end;
run;
proc sort data=two_on_study;
  by id date;
run;
data two_ae;
  ID=1;
  AESTDT='27Oct2003'd;
  AEENDT='30Oct2003'd;
  output;
  ID=1;
  AESTDT='27Oct2003'd;
  AEENDT='01Nov2003'd;
  output;
  ID=2;
  AESTDT='01Nov2003'd;
  AEENDT='01Nov2003'd;
  output;
  ID=3;
  AESTDT='27Oct2003'd;
  AEENDT='29Oct2003'd;
  output;
run;
data two_sick(drop=AE:);
  set two_ae;
  do date=AESTDT to AEENDT;
    output;
  end;
run;
proc sort data=two_sick nodupkey;
  by id date;
run;
data two_events;
  set two_ae(rename=(AESTDT=date) drop=AEENDT);
  event=1;
run;
proc means noprint data=two_events;
  class id date;
  var event;
  output out=two_events_sum(where={_type_=3}) sum=event;
run;
```

Merge the above data sets by subject ID and date to get the layout seen below. Note that due to the NODUPKEY option in the PROC SORT of the two_sick data set above, the overlapping AEs for the first subject are counted as sick days only once. The fact that two events started on the same day shows up on 27Oct03 for this subject as it should.
Summarize the above data with PROC MEANS and you obtain the variables requested, as shown below. Note that the three subjects each being on the study 10 days comes to the 30 on-study days shown, 10 days are shown as sick days (14 minus 4 which overlapped), and the four events are shown. Likely, a ratio of sick days to on-study days will be requested in the real world.

```
proc means noprint data=two;
  class year month;
  var onstudy sick event;
  output out=two_2(where=(_type_=3))
    sum=onstudy_days sick_days event_starts;
run;
```

As you suspect, this is a simplification of a production request. In the real world, missing start and stop dates must be dealt with. Some AEs qualify to be counted and others don’t. AEs start before the study drug starts and/or end after discontinuation of study drug. Instead of events they will want number of patients experiencing an event. There are different treatments and treatment periods to deal with, and headaches about how to properly account for AEs that span multiple periods. And there is always a twist; in the study that motivated this example, the client requested “days at risk” as on-study days minus sick days. It took a while to get used to thinking of a subject who was always sick as technically having no “days at risk” of being sick.
3. CATCH ALL OCCURRENCES WITH THE DATA STEP SELECT STATEMENT.

Several years ago SAS (1990) came out with a nice little programming tips booklet. In one of this booklet’s examples, the IF-THEN-ELSE method shown below is used to create the comb variable from the gender and dept variables in the data set below. We reproduced these results using the SELECT method also shown below.

<table>
<thead>
<tr>
<th>gender</th>
<th>dept</th>
<th>comb</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>develop</td>
<td>1</td>
</tr>
<tr>
<td>M</td>
<td>writing</td>
<td>2</td>
</tr>
<tr>
<td>M</td>
<td>testing</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>develop</td>
<td>4</td>
</tr>
<tr>
<td>F</td>
<td>writing</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>testing</td>
<td>6</td>
</tr>
</tbody>
</table>

**IF-THEN-ELSE METHOD**;
data three_1;
set three;
if gender='M' then do;
  if dept='develop' then comb=1;
  else if dept='writing' then comb=2;
  else comb=3;
end;
else do;
  if dept='develop' then comb=4;
  else if dept='writing' then comb=5;
  else comb=6;
end;

**SELECT METHOD**;
data three_2;
set three;
select (gender);
when ('M') do;
  select (dept);
  when ('develop') comb=1;
  when ('writing') comb=2;
  when ('testing') comb=3;
end;
when ('F') do;
  select (dept);
  when ('develop') comb=4;
  when ('writing') comb=5;
  when ('testing') comb=6;
end;
end;

Now the IF-THEN-ELSE method (henceforth called IF) is a tried and true workhorse: it is not going anywhere. Most of us cut our teeth on it either in SAS or in other languages. To produce the same results above, the IF method uses significantly fewer unique statements (10 to 16) than the SELECT method. And may we apologize for perhaps not formatting the above IF example in the most ideal manner. (Indeed, we see so many ways of formatting IF code, especially when do statements are involved, we suspect that not only is there no preferred way, the variety of ways indicate that a problem exists in writing in a format that is easy for everyone to follow.)

However, perhaps there is an occasional place for the SELECT method. In this example, we assert that there is no problem following the logic of the SELECT while some may become entangled in the IF. Clearer code translates to reduced programming and maintenance time and fewer errors.

Not to be overlooked, however, is a pleasant little error trapping feature of the SELECT. SELECT assumes that you specify mutually exclusive categories. Note that in the IF method, in the case of gender=‘M’ and dept equals anything but ‘develop’ or ‘writing’, comb is assigned 3. In the SELECT method in the case of gender=‘M’, comb is assigned 3 only in the case of dept=‘testing’. In the IF case of gender=‘M’, a missing dept results in comb=3 while SELECT returns an error message. If this was your intent, perhaps SELECT has a place in your toolkit.

Of course either case can be considered a feature. If your intent is really to assign comb=3 in cases where dept is missing or elephant or even a dangerous misspelling of a comb=1 level (say, ‘develope’), IF is properly specified. And, of course, both methods give override control. You can write another line or two of code to make the IF statement trap such errors and you can override this feature in SELECT by specifying the OTHERWISE statement.

The DATA step SELECT statement should not be confused with the SELECT statement found in SQL.
4. USE FUNCTIONS, FEATURES, AND CARE.

Here's a quick reminder about SAS functions. They are quite useful as long as they are used with care. In the example below, plusa adds the four variables beginning with "a" the old fashioned way and properly indicates missing values. On the other extreme is sumofa which treats missing values as zero. In between is suma13 which is manually defined to ignore variables containing missing values. Note that the commented out suma variable would return an error if active.

```sas
data four;
input a1 a2 a3 a4;
plusa=a1+a2+a3+a4;
*suma=sum(a:);
suma13=sum(a1,a3);
sumofa=sum(of a:);
cards;
1 2 3 4
5 . 7 8
9 10 11 .
;
Obs  a1  a2  a3  a4  plusa  suma13  sumofa
1    1   2   3   4    10      4      10
2    5   .   7   8     .     12      20
3    9  10  11   .     .     20      30
```

5. CREATING A GLANCE OF THE DATA FOR AN ENTIRE LIBRARY.

New projects often bring new data set structures; we find ourselves running PROC CONTENTS, PROC PRINTS or viewing the data frequently. This helpful reference tool is an alternative: it prints a list of all data sets found in a directory as well as producing a PROC CONTENTS and PROC PRINT of each data set. Parameters allow the user to specify the target directory and the number of observations to be printed. Note also that the macro sorts by the variable SUBID, contingent upon its existence.

```sas
%macro x (libref=, num=);
proc contents data=&libref.._all out=cont (keep = memname name);
run;

data _null_; set cont; by memname; retain flag 0; if _n_=1 then n=0; if first.memname then do; n+1; call symput('DATFL'||left(n), left(trim(memname))); flag=0; end; if upcase(name)='SUBID' then flag=1; if last.memname then call symput('cv'||left(n), left(trim(flag))); run;
proc sort data=cont nodupkey out=cont; by memname; run;

data cont; set cont end=eof; by memname; if _n_ = 1 then ord = 0; if first.memname then do; ord+1; call symput('FILE'||left(ord), trim(memname)); end; if eof then do; call symput('nummem', left(ord)); end; run;
%do i=1 %to &nummem;
title "&&file&i FILE";
proc contents data=&libref..&&datfl&i;
run;
```
%if &&cv$i -1 %then %do;
  proc sort data=&libref..&&datfl$i out=&&datfl$i;
  by subid;
  run;
%end;
%else %do;
  data &&datfl$i;
  set &libref..&&datfl$i;
  run;
%end;
  proc print data=&&datfl$i (obs=&num);
  run;
%end;
%mend x;
%x (libref=dds, num=25);

6. USING PROC TABULATE TO CREATE IN-TEXT TABLES.

PROC TABULATE seems to be an uncommon, forgotten tool. But this is a good tool to use when programming in-text tables such as enrollment totals or demographic breakdowns. Many medical writers prefer this type of output inserted in the text of a report. Below is an example of a demographic table produced using PROC TABULATE:

```
Proc format;
  value agefmt     1='0 - 15'
                  2='16 - 64'
                  3='>= 65';
  value sexfmt     1='Male'
                  2='Female';
  value racefmt    1='White'
                  2='Black'
                  3='Asian/Oriental'
                  4='Multiracial'
                  5='Other';
run;

data one(keep=pno patno pt age race agecat agecol sexcol sexcat racecol coltot);
  set one;
  if a;
  if 0 le age le 15 then agecat=1;
  else if 16 le age le 64 then agecat=2;
  else if age ge 65 then agecat=3;
  if gender='M' then sexcat=1; else
    if gender='F' then sexcat=2;
  agecol =1; racecol=1; sexcol =1; coltot =1;
run;
proc tabulate data=one missing;
  var pt;
  class agecol sexcol racecol agecat racecat sexcat coltot;
  format agecat agefmt. sexcat sexfmt. agecol agec. racecol racec. sexcol sexc. race racefmt. coltot colfmt. ;
  keylabel N=' ' PCTN='%';
  tables agecol='*agecat=' ' sexcol=' '*sexcat=' ' racecol=' '**race=' ',
             coltot='*f=5.'/printmiss row=float;
run;
```
Demographics for Protocol AAA/BBB

<table>
<thead>
<tr>
<th>AGE (years)</th>
<th>0 - 15</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16 - 64</td>
<td>251</td>
</tr>
<tr>
<td></td>
<td>&gt;= 65</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GENDER</th>
<th>Male</th>
<th>72</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Female</td>
<td>198</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RACE</th>
<th>White</th>
<th>201</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Black</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>Asian/Oriental</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Multiracial</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>31</td>
</tr>
</tbody>
</table>

REFERENCES

ACKNOWLEDGMENTS
SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the authors by phone at 816-767-6000, by mail at Quintiles, Inc., P.O. Box 9708, Kansas City, MO 64134-0708, or by E-mail at Kristi.Wiser@Quintiles.com and John.Morrill@Quintiles.com.

BIO INFORMATION
Both authors are Analysts in Statistical Programming at Quintiles, Inc. in Kansas City. Kristi has 11 years of SAS experience with 10 years in the pharmaceutical industry. John first used SAS in 1985 and has used it full-time in the pharmaceutical industry since 1998.

KEYWORDS
Toolkit, Combinations, Completetypes, Adverse Event, Select, Macro, Glance, Directory, Tabulate