PAPER TS10

ACCESS YOUR DATA IN AN OPEN MANNER

Steve Prust, Independent Consultant

1. ABSTRACT

Data processing doesn't necessarily fit iterative techniques and this paper looks at the difficulties inherent in the processing implied by using SET and MERGE processing in SAS. Techniques such as PROC TRANSPOSE and the LAGx function may assist in some situations. However when we need to access disparate data across either observations or across datasets things can get tough resulting in complex, unwieldy code. In these situations we can use the functionality of OPEN, FETCH, GETVARx etc. to flexibly access data, producing well-structured and reasonably concise code. The paper gives an overview of the functions associated with OPEN and demonstrates their use with examples of a) Adverse Event SMQs, and b) the calculation of time post-dose with partial date/times in multiple dose studies

2. ITERATIVE PROCESSING

The SAS Data step is almost exclusively used with the SET or MERGE statements. These statements require data observations to be read sequentially. The ability to access data across observations is limited by the concept of the Program Data Vector (PDV). As each observation is read, the PDV is initialised with its values (and all other variables not deriving from input datasets are initialised to missing). Should data from other observations, or other datasets be needed then programmatic solutions must be found. The RETAIN statement will allow the PDV to hold variables that are not compulsorily initialised and the LAGx function will allow data from previous observations to be accessed (in practice the LAGx function is a barely worth the effort as it is somewhat non-intuitive). These features barely lift the restrictive nature of the SET / MERGE processing.

The solution to such restrictions is typically to build a series of PROCs and Data steps to arrange the data to fit this iterative model. Inevitably, this produces code complexity. It also makes it difficult to make choices about data access: within a program some data may require accessing further data from a particular source whereas other data may require further information from elsewhere. Making all possible data available can result in lots of PROC SORTs, PROC TRANSPOSEs, MERGEs etc. etc.

How much easier it would be to do it all within a single data step .....

3. OPEN FUNCTIONALITY

The OPEN function can be used in the SAS data step to directly access one or more SAS data sets. Use of this function may be combined with or without SET and MERGE statements. Use of OPEN implies a more hands-on approach to data access: records must be explictly accessed, data items may need to be accessed individually. The example below contrasts the two methods:

```
data males;
  set sashelp.class;
  where sex = 'M';
  keep age name;
run;

data males;
  dsid = open('sashelp.class (where=(sex="M"))','I');
  if dsid then do;
    rc=fetch(dsid);               /* fetch statement gets and observation *?
    do while(rc=0);
      age  = getvarn(dsid,varnum(smqdetid,'AGE'));
      name = getvarc(dsid,varnum(smqdetid,'NAME'));
      rc=fetch(dsid);
    end;
  rc=close(dsid):
  keep age name;
run;
```

Clearly the second example is the less easy to code and understand. In particular each observation has to be explicitly fetched (and the code constructed to deal with end-of-file conditions). Then data is not automatically available in the program data vector but must be fetched via a combination of VARNUM and GETVARx functions (note: there is an alternative way of making observation data available but the is considered less suitable for the purpose of this paper).

What the second method *does* gives us is more control. We can open a second dataset if we like, or a third or fourth.

Observations can be read when the processing requires it. We can read one observation or many. We can decide that the data needs updating and so close the dataset, re-open it for update etc. etc. In short we can use the data step to follow the structures inherent in our data rather than try to construct complex sequences of PROCs and Data steps.

3.1 ASSOCIATED FUNCTIONS

There are several other functions associated with the OPEN function:

CLOSE	- to close an open dataset
FETCH	- to fetch an observation from a dataset
VARNUM	- given a valid variable name this returns the column number of that variable in the dataset. Note, several of the following functions require column numbers as arguments rather than variables names.
GETVARC	- to get the contents of a character data value from the current observation (GETVARN does the same for numeric data)
ATTRC	- to get information about a data set's attribute for those attributes that are expressed in character form (ATTRN does the same for attributes expressed in numeric values).
PUTVARC	- to write a character data value into the current observation (PUTVARN does the same for numeric data)
UPDATE	- to write an updated observation to a dataset.

For full details of these and other functions see the SAS help guide.

3.2 PERFORMANCE

The inevitable consequence of flexible access to data is a performance overhead. There is no means by which SAS is going to be as efficient as it can be when performing block reads of sequential data. Performance will be a factor of the number of datasets accessed, records read etc. etc. For the author there have been circumstances where running times of programs using OPEN have been significant but for the far greater part this has not been an issue.

4. EXAMPLE OF OPEN FUNCTIONALITY TO FIND SUBJECTS SATISFYING ALGORITHMIC SMQS

To demonstrate some of the functionality afforded by the OPEN function the following example is presented.

4.1 ALGORITHMIC STANDARDISED MEDDRA QUERIES (SMQS)

SMQs are a means to assessing whether a condition potentially has occurred by looking at a range of MedDRA Preferred Terms and/or Low Level Terms. Algorithmic SMQs require a algorithm, such as "A and (B or C)", to be evaluated with the letters of the algorithm standing for a set of Preferred Terms.

4.2 DATASETS

The data used in this example are

- WORK.AES: a set of Adverse Events from a hypothetical study
- WORK.SUBS: a set of subject numbers
- WORK.SMQDEF: a set of algorithmic SMQs definitions
- WORK.SMQ: a set of termcats, term weights and preferred terms for all SMQs definitions

Example of WORK.SMQDEF

SMQALGO	SMQCD	SMQNAME
A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)
A or (B and C)	20000022	Acute pancreatitis (SMQ)
A or (B and C and D)	20000044	Neuroleptic malignant syndrome (SMQ)
A or Sum(Category Term Weight)>6	20000045	Systemic lupus erythematosus (SMQ)
A or (B and C and D)	20000048	Anticholinergic syndrome (SMQ)
A or (B and C)	20000157	Eosinophilic pneumonia (SMQ)

Example of WORK.SMQ

TERMCAT	TERMWGHT	SMQALGO	SMQCD	SMQNAME	PTCD	PTNAME
A	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10001053	Acute respiratory failure
A	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10002198	Anaphylactic reaction
A	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10002199	Anaphylactic shock
A	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10002216	Anaphylactoid reaction
C	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10002424	Angioedema
B	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10003553	Asthma
D	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10005734	Blood pressure decreased
D	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10005737	Blood pressure diastolic decreased
D	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10005758	Blood pressure systolic decreased
B	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10006482	Bronchospasm
D	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10007515	Cardiac arrest
D	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10007617	Cardio-respiratory arrest
B	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10008469	Chest discomfort
B	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10008589	Choking
B	0	A or (B and C) or (D and (B or C))	20000021	Anaphylactic reaction (SMQ)	10008590	Choking sensation

4.3 PROCESSING OVERVIEW

The objective is to find for each algorithmic SMQ those subjects who have AEs or combinations of AEs that satisfy that SMQ. For each SMQ there are two sets of processing. The first process is to find the MedDRA preferred terms that correspond to each termcat, the second is a a loop through the list of subjects. For each iteration of the loop through the list of subjects the SMQ algorithm will be evaluated by firstly finding, for each termcat in turn, whether any matching AEs are present and, secondly, resolving the algorithm. If a subject's AEs satisfy the SMQ algorithm then this is reported.

This processing can be expressed in pseudo-code as below. The places where the OPEN and associated functions are shown by highlighting:

```
For Each smq in WORK.SMQDEF
   For Each potential termcat ('A','B','C', …) in algorithm definition ❶
      Get preferred terms and term weights (as needed) from WORK.SMQ ❷
   End For
   For Each subject in WORK.SUB ❸
      For Each termcat used in the algorithm
         Get any AEs from WORK.AES for this subject with the termcat's preferred terms ❹
      End For
      Resolve algorithm❺
      If SMQ algorithm is satisfied then
         Report
      End If
  End For
End For
```

The processing of WORK.SMQDEF is done via the SET statement for convenience. The remaining data access is done by use of the OPEN function and associated functions.

Full code for the program can be found in the Appendix. Items above marked ❶, ❷, ❸ etc. are explained in more detail below

4.3.1 Pre-processing the algorithm ❶ and resolving the algorithm❺

The algorithm is pre-processed so that it can be evaluated more easily later (principally adding blanks between all elements of the algorithm). An algorithm, for example :

A or (B and C) or (D and (B or C))

will be resolved by replacing each letter with a Boolean true / false value according to the data. For example (where for a particular subject A=0, B=1, C=0, D=1) the expression becomes:

0 or (1 and 0) or (1 and (1 or 0))

The program will act as a Boolean logic processor and resolve this in stages, i.e:

0 or 0 or (1 and 1)

then

0 or 1

Finally, results will be either "0" or "1", which can be tested in an IF statement.

The code for resolving the "and / or" logic is as follows

```
cresalgo = compress(cresalgo);
```

```
do while (index(cresalgo,'AND') or index(cresalgo,'OR') or index(cresalgo,'(') or
index(cresalgo, ')');
  cresalgo = tranwrd(cresalgo,'0AND0','0');
  cresalgo = tranwrd(cresalgo,'1AND0','0');
  cresalgo = tranwrd(cresalgo,'0AND1','0');
  cresalgo = tranwrd(cresalgo,'1AND1','1');
  cresalgo = tranwrd(cresalgo,'0OR0','0');
  cresalgo = tranwrd(cresalgo,'1OR0','1');
  cresalgo = tranwrd(cresalgo,'0OR1','1');
  cresalgo = tranwrd(cresalgo,'1OR1','1');
  cresalgo = tranwrd(cresalgo,'(1)','1');
  cresalgo = tranwrd(cresalgo,'(0)','0');
end;
```

(Note: the code for processing the sum of weights is not shown here – see appendix)

4.3.2 Getting preferred terms / term weight associated with element of the algorithm❷

The getting of the preferred terms and term weights is done using the OPEN function and other associated functions.

```
smqdetds = 'smq (where=( smqcd="'||trim(smqcd)||'" and termcat = "'||trim(item)||'" )';
smqdetid = open(smqdetds,'I');
ptlist = '';
rc = fetch(smqdetid);
do while(rc=0);
  cand_pt = getvarc(smqdetid,varnum(smqdetid,'PTCD'));
  if missing(ptlist) then ptlist = '"' || trim(cand_pt) || '"';
                     else ptlist = trim(ptlist) || ', "' || trim(cand_pt) || '"';
  rc = fetch(smqdetid);
end;
ptlist = 'pt in (' || trim(ptlist) || ')';
rc = close(smqdetid);
```

(Note the getting of the preferred term weights is similar and shown in the appendix)

4.3.3 Looping through the subjects

This code is fairly straightforward:

```
subsds = 'work.subs';
subsid = open(subsds,'I');
if subsid then do;

  /* prepare some counting variables … */

  rc = fetch(subsid);
  do while(rc=0);
    subjno = getvarn(subsid,varnum(subsid,'SUBJNO'));

    /* intermediate code - look at AEs for subject, resolve algorithm, and report */

    rc = fetch(subsid);
  end;
  rc = close(subsid);
end;
```

4.3.4 Finding if AEs match the termcat for a subject

This code is slightly more complex but fairly short. An array (terms[]) holds the list of preferred terms required for each term category. For example

> pt in ("10002198", "10002199", "10002216", "10009192", "10040560", "10045240", "10063119", "10067113", "10068158", "10069167")

This is used as part of the WHERE condition when opening the WORK.AES dataset using the OPEN function.

If the program finds a single observation when trying the read the opened dataset it means that this subject must have an AE corresponding to this termcat. This result is stored in the array *termres*[].

After closing the dataset for each termcat the original SMQ algorithm may be resolved for that particular termcat using the function TRANWRD. That is if, for example, the termcat is “A” and there are AEs for this subject in that termcat, then an algorithm of “A and (B or C)” (say) would be resolved to “1 and (B or C)”.

For reporting purposes a list of qualifying AE preferred term codes together with the associated termcat is retained in the variable *fptlist*.

```
do k = 1 to 4;
  termres[k] = 0;
  length letter $1;
  letter = substr('ABCD',k,1);
  if not missing(terms[k]) then do;
    aeds = 'work.aes (where=(subjno=' || compress(put(subjno,best.)) ||
                              ' and (' || trim(terms[k]) || ')))';
    aeid = open(aeds,'I');
    if aeid then do;
      rc = fetch(aeid);
      if rc = 0 then termres[k] = 1;
      do while(rc=0);
        getpt = getvarc(aeid,varnum(aeid,'PT')) || '(' || letter || ')';
        if missing(fptlist) then fptlist = getpt;
                            else fptlist = trim(fptlist) || ', ' || getpt;
        rc = fetch(aeid);
      end;
      rc = close(aeid);
    end;
    nresalgo = tranwrd(nresalgo,' ' || letter || ' ',put(termres[k],1.));
  end;
end;
```

4.4 RESULTS

Combining the above code produces a single data step that is able to search for all algorithmic SMQs in a set of data.

A simple reporting mechanism of PUT statements on a set of sample data produced these results:

```
SMQ: Anaphylactic reaction (SMQ) ,  algorithm: A or (B and C) or (D and (B or C))
  Subject satisfying SMQ: 8061
  Contributing PTs :10001053  (B), 10016029  (C)
  Subject satisfying SMQ: 8182
  Contributing PTs :10002198  (A)
  Subject satisfying SMQ: 8461
  Contributing PTs :10011224  (B), 10008589  (B), 10065929  (D)
  Subject satisfying SMQ: 8462
  Contributing PTs :10015967  (C), 10065929  (D)
  Subject satisfying SMQ: 9863
  Contributing PTs :10011224  (B), 10016825  (C)
Number of subjects processed:252
Number of subjects satisfying SMQ:5

SMQ: Acute pancreatitis (SMQ) ,  algorithm: A or (B and C)
  Subject satisfying SMQ: 8462
  Contributing PTs :10033635  (A), 10028813  (C), 10028813  (C)
  Subject satisfying SMQ: 8463
  Contributing PTs :10005364  (B), 10067715  (C)
Number of subjects processed:252
Number of subjects satisfying SMQ:2

SMQ: Neuroleptic malignant syndrome (SMQ) ,  algorithm: A or (B and C and D)
  Subject satisfying SMQ: 8464
  Contributing PTs :10029282  (A)
  Subject satisfying SMQ: 8629
  Contributing PTs :10005911  (B), 10020651  (C), 10061536  (D)
Number of subjects processed:252
Number of subjects satisfying SMQ:2

SMQ: Systemic lupus erythematosus (SMQ) ,  algorithm: A or Sum(Category Term Weight)>6
  Subject satisfying SMQ: 8941
  Contributing PTs :10040968  (A)
  Subject satisfying SMQ: 9503
  Contributing PTs :10051246  (Sum Cat:B, Weight:1), 10035618  (Sum Cat:E, Weight:3), 10047942
(Sum Cat:H, Weight:3)
Number of subjects processed:252
Number of subjects satisfying SMQ:2

SMQ: Anticholinergic syndrome (SMQ) ,  algorithm: A or (B and C and D)
  Subject satisfying SMQ: 8628
  Contributing PTs :10002757  (A)
  Subject satisfying SMQ: 8642
  Contributing PTs :10013573  (B), 10019070  (C), 10037660  (D), 10037660  (D), 10047532  (D)
Number of subjects processed:252
Number of subjects satisfying SMQ:2
```

```
SMQ: Eosinophilic pneumonia (SMQ) ,  algorithm: A or (B and C)
  Subject satisfying SMQ: 8645
  Contributing PTs :10014962  (A)
  Subject satisfying SMQ: 8663
  Contributing PTs :10058133  (B), 10063527  (C)
Number of subjects processed:252
Number of subjects satisfying SMQ:2
```

5. RECONCILING ADVERSE EVENTS

When reconciling events (such as Adverse Events with dosing data), it may be necessary to apply complex rules, especially when processing partial dates. Again this is a situation where the ability to combine Data step conditional statements with OPEN functionality can be an effective approach.

A simple example of determining which dose precedes an Adverse Event.

```
doseds = 'dose (where=( subjid="' || trim(subjid) ||
                '" and dosedtm <= "' ||put(aedtm,datetime18.) || 'dt" )';
doseid = open(doseds,'I');
dosedtm = .;
rc = fetch(doseid);
do while(rc=0);
  dosedtm = getvarc(doseid,varnum(doseid,'DOSEDTM'));
  rc = fetch(doseid);
end;
rc = close(doseid);
```

Should the datetime information be only partial that we can adjust the logic according to whatever rules might be in place for such situations. So :

```
if timepart(aedtm) then
  doseds = 'dose (where=( subjid="' || trim(subjid) ||
                '" and dosedtm <= "' ||put(datepart(aedtm),date9.) || ':09:00:00dt" )';
else
  doseds = 'dose (where=( subjid="' || trim(subjid) ||
                '" and dosedtm <= "' ||put(aedtm,datetime18.) || 'dt" )';
```

Perhaps we also want to check that when an AE has an action of concomitant medication given then there is a corresponding record :

```
if putn(aeact,vformat(aeactn)) in ('medication given') then do;
  cmds = 'cm (where=( subjid="' || trim(subjid) ||
                '" and aenum = "' ||put(aeid,z3.) || '" )';
  cmid = open(cmds,'I');
  rc = fetch(cmid);
  if rc ne = 0 then
    put 'no corresponding Conmed found';
  rc = close(cmid);

etc...
```

The possibilities for cross-checking, etc. are extensive. The code can be as flexible as whatever requirements are in place.

6. PRACTICALITIES

The OPEN function can look a little difficult to beginners and there are plenty of pitfalls. Notably these are:

- leaving datasets open (remember to use the CLOSE function)
- getting the syntax of the various functions correct (the OPEN function when using the WHERE data set option can require accurate coding especially in terms of quoting values)
- code complexity
- lack of planning can lead to even worse programs – techniques such as psuedo-coding are recommended for complex situations
- performance can be degraded by, what is effect, random access to data. Be prepared to test out time-critical applications.
- Using OPEN when it is not necessary – if there is a simpler solution then use it!

7. CONCLUSION

Used appropriately the OPEN function can make programming in SAS easier and more robust. Successful applications of the technique will typically require planning and an appreciation of the structure of the processing required.

There are potential issues with performance. When scaling up applications then make sure to test this aspect of your code.

8. REFERENCES

John van Bemmelen, 2008, Applying SMQs to Adverse Event Data, PhUSE 2008,
http://www.lexjansen.com/phuse/2008/tu/tu05.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Steve Prust
prustconsulting@gmail.com

APPENDIX – COMPLETE CODE

```
data smqres;
  set smqdef;
  /* definitions of work areas, arrays etc. */
  length item newalgo smqdetds gtterm $200;
  length resalgo ptlist subsds aeds fptlist getpt $4096;
  length aterm bterm cterm dterm atermc btermc ctermc dtermc etermc ftermc gtermc htermc
         itermc $4096;
  array terms[4] aterm bterm cterm dterm;
  array termcs[9] atermc btermc ctermc dtermc etermc ftermc gtermc htermc itermc;
  format wgt1-wgt9 best.;
  array termcw[9] wgt1-wgt9;
  array termres[4] ares bres cres dres;
  keep subjno fptlist smqcd smqname smqalgo ares bres cres dres gtterm;

  /* pre-processing the algorithm */
  i = 1;
  newalgo = smqalgo;
  newalgo = tranwrd(newalgo,'(',' ( ');
  newalgo = tranwrd(newalgo,')',' ) ');
  newalgo = tranwrd(newalgo,'>',' > ');
  newalgo = tranwrd(newalgo,'Category Term Weight','CATEGORY_TERM_WEIGHT');

  /* loop through each potential termcat ('A','B','C', …) in algorithm definition */
  item = scan(newalgo,i,' ');
  resalgo = ' ';
  do while (not missing(item));
    select ;
      when ( upcase(item) in ('A','B','C','D') ) do;
        idx = index('ABCD',trim(upcase(item)));
        if missing(terms[idx]) then do;
          link get_tl; /* find list of terms for termcat */
          terms[idx] = ptlist;
        end;
        resalgo = trim(resalgo) || ' ' || upcase(item);
      end;
      when ( upcase(item) in ('OR','AND','(',')','>','SUM','6','7') ) do;
        resalgo = trim(resalgo) || ' ' || upcase(item);
      end;
      when ( upcase(item) in ('CATEGORY_TERM_WEIGHT') ) do;
        link get_tlc; /* find list of terms for summary termcat */
        resalgo = trim(resalgo) || ' ' || item;
      end;
      otherwise put item=;
    end;
    i = i+1;
    item = scan(newalgo,i,' ');
  end;

  put 'SMQ: ' smqname ',  algorithm: ' smqalgo;

  /* loop through each subject */
  subsds = 'work.subs';
  subsid = open(subsds,'I');
  if subsid then do;
    subcount = 0;
    smqcount = 0;
    rc = fetch(subsid);
    do while(rc=0);
      subjno = getvarn(subsid,varnum(subsid,'SUBJNO'));
      subcount = subcount + 1;
      nresalgo = resalgo;
      fptlist = ' ';

      /* find if this subject has any AEs that match the SMQ algorithm */

      do k = 1 to 4;
        termres[k] = 0;
        length letter $1;
        letter = substr('ABCD',k,1);
        if not missing(terms[k]) then do;
          aeds = 'work.aes (where=(subjno=' || compress(put(subjno,best.)) ||
                                  ' and (' || trim(terms[k]) || ')))';
          aeid = open(aeds,'I');
          if aeid then do;
```

```
      rc = fetch(aeid);
      if rc = 0 then termres[k] = 1;
      do while(rc=0);
        getpt = getvarc(aeid,varnum(aeid,'PT')) || '(' || letter || ')';
        if missing(fptlist) then fptlist = getpt;
                            else fptlist = trim(fptlist) || ', ' || getpt;
        rc = fetch(aeid);
      end;
      rc = close(aeid);
    end;
    nresalgo = tranwrd(nresalgo,' ' || letter || ' ',put(termres[k],1.));
  end;
end;

if index(resalgo,'SUM') then do;
  catsum = 0;
  do j = 1 to 9;
    if termcw[j] ne 0 then do;
      letter = substr('ABCDEFGHI',j,1);
      aeds = 'work.aes (where=(subjno=' || compress(put(subjno,best.)) ||
                        ' and (' || trim(termcs[j]) || ')))';
      aeid = open(aeds,'I');
      if aeid then do;
        rc = fetch(aeid);
        if rc = 0 then catsum = catsum + termcw[j];
        do while(rc=0);
          getpt = getvarc(aeid,varnum(aeid,'PT')) ||
          '(Sum Cat:' || letter || ', Weight:' || compress(put(termcw[j],best.)) || ')';
          if missing(fptlist) then fptlist = getpt;
                              else fptlist = trim(fptlist) || ', ' || getpt;
          rc = fetch(aeid);
        end;
        rc = close(aeid);
      end;
    end;
  end;
  nresalgo = tranwrd(nresalgo,'SUM ( CATEGORY_TERM_WEIGHT )',put(catsum,best.));
end;

cresalgo = nresalgo;

/* resolve ">" logic */
do while ( index(cresalgo,' >') or index(cresalgo,'> ');
  cresalgo = tranwrd(cresalgo,' >','>');
  cresalgo = tranwrd(cresalgo,'> ','>');
end;
foundgt = 0;
do i = 1 to length(cresalgo) while(not foundgt);
  if substr(cresalgo,i,1) = '>' then foundgt = i;
end;
if foundgt then do;
  from = 0;
  do i = 1 to foundgt;
    if substr(cresalgo,i,1) = ' ' then from = i;
  end;
  gtterm = scan( substr(cresalgo,from) , 1,' ');
  if input( scan(gtterm,1,'>') ,best.) > input( scan(gtterm,2,'>') ,best.)
    then cresalgo = tranwrd(cresalgo,gtterm,'1');
    else cresalgo = tranwrd(cresalgo,gtterm,'0');
end;

/* resolve "AND" / "OR " logic */
cresalgo = compress(cresalgo);
do while (index(cresalgo,'AND') or index(cresalgo,'OR') or index(cresalgo,'(') or
          index(cresalgo,')');
  cresalgo = tranwrd(cresalgo,'0AND0','0');
  cresalgo = tranwrd(cresalgo,'1AND0','0');
  cresalgo = tranwrd(cresalgo,'0AND1','0');
  cresalgo = tranwrd(cresalgo,'1AND1','1');
  cresalgo = tranwrd(cresalgo,'0OR0','0');
  cresalgo = tranwrd(cresalgo,'1OR0','1');
  cresalgo = tranwrd(cresalgo,'0OR1','1');
  cresalgo = tranwrd(cresalgo,'1OR1','1');
  cresalgo = tranwrd(cresalgo,'(1)','1');
  cresalgo = tranwrd(cresalgo,'(0)','0');
end;
if input(cresalgo,best.) then smqcount = smqcount + 1;
if cresalgo = '1' then do;
  put '  Subject satisfying SMQ: ' subjno ;
  put '  Contributing PTs :' fptlist;
```

```
        output;
      end;
      rc = fetch(subsid);
    end;
    rc = close(subsid);

    put 'Number of subjects processed:' subcount;
    put 'Number of subjects satisfying SMQ:' smqcount;
    put ;
  end;

return;

get_tl:
/* subroutine to get a list of preferred terms for "letter" items */

  length cand_pt $200;
  smqdetds = 'smq (where=( smqcd="' || trim(smqcd) ||'" and termcat = "' ||trim(item)|| '" )';
  smqdetid = open(smqdetds,'I');
  ptlist = '';
  if smqdetid then do;
    rc = fetch(smqdetid);
    do while(rc=0);
      cand_pt = getvarc(smqdetid,varnum(smqdetid,'PTCD'));
      if missing(ptlist) then
        ptlist = '"' || trim(cand_pt) || '"';
      else
        ptlist = trim(ptlist) || ', "' || trim(cand_pt) || '"';
      rc = fetch(smqdetid);
    end;
    ptlist = 'pt in (' || trim(ptlist) || ')';
    rc = close(smqdetid);
  end;
  else put 'unexpectedly unable to open using ' smqdetds=;

return;

get_tlc:
/* subroutine to get a list of preferred terms for the sum of category terms */

  length cand_pt $200 itemc $1;
  do j = 1 to 9;
    itemc = substr('ABCDEFGHI',j,1);
    smqdetds = 'smq (where=( smqcd="' ||trim(smqcd)||'" and termcat = "' ||trim(itemc)||'" )';
    smqdetid = open(smqdetds,'I');
    ptlist = '';
    wt     = 0;
    if smqdetid then do;
      rc = fetch(smqdetid);
      wt = getvarn(smqdetid,varnum(smqdetid,'TERMWGHT'));
      do while(rc=0);
        cand_pt = getvarc(smqdetid,varnum(smqdetid,'PTCD'));
        if missing(ptlist) then ptlist = '"' || trim(cand_pt) || '"';
                           else ptlist = trim(ptlist) || ', "' || trim(cand_pt) || '"';
        rc = fetch(smqdetid);
      end;
      ptlist = 'pt in (' || trim(ptlist) || ')';
      rc = close(smqdetid);
    end;
    else put 'unexpectedly unable to open using ' smqdetds=;
    termcs[j] = ptlist;
    termcw[j] = wt;
  end;

return;
run;
```