Programming Tips and Examples for Your Toolkit, II

John Morrill, Overland Park KS.

ABSTRACT

This paper describes several independent ideas that should be in the toolkit of intermediate level SAS ® programmers. We begin with a quick illustration of what can go wrong during a merge of two data sets containing identical variable names if these variables are not included in the by statement. Next we review a half-dozen methods of 'commenting out' text that appears within code. While documenting the purpose of each section of code is ideal, the focus of this section is on the technical side, demonstrating methods to accomplish this and discussing interactions between comment methods. We then move to a discussion of a multilevel indicator variable, one that incorporates information from several variables. A major focus is detailing tools needed to create a concordance, an index of words in a given text showing every contextual occurrence of a word. In explaining each step (a fully operational concordance program is given in the appendix, and the example text is the New Testament book of Luke) we cover a suite of tools including reading in the data, the use of arrays, various functions and issues related to working with character data, the colon operator, and data null with put statements used in creating the final product. Various secondary issues such as creating a headings output and what can be done with footnotes are also discussed.

INTRODUCTION

Four sections are included in this tips paper. The first illustrates with small data sets a fairly standard potential merge problem. The second covers the technical side of comments. The third discusses a way of incorporating information from several variables compactly into a single variable. The fourth presents a point-and-click concordance-generating macro: point to a body of literature and output a listing of all words and their relative locations within the text. The focus is on the general text-to-concordance process but code-level suggestions are also featured. Each section contains a word on application to the pharmaceutical industry.

1. MERGES BETWEEN DATA SETS WITH COMMON VARIABLES NOT IN THE BY STATEMENT

The 'by' statements of data step merges are worthy of special attention. We illustrate this with the following example. Section 1.1 presents two data sets with the PAIRNO and FOOD variables in common. Section 1.2 shows various merges of these data sets. Section 1.3 shows proc print output of the data sets produced by these merges. Finally, Section 1.4 discusses various issues surrounding these merges.

1.1 RAW DATA.

```
a

Obs    pairno     food       animal
 1        1      apple      aardvark
 2        2      avocado    anteater

b

Obs    pairno     food      town
 1        1      banana    BocaRaton
 2        2      beans     Boston
 3        3      burrito   Barcelona
```

1.2 VARIOUS MERGES.

```
data ab;                 **1.3.1  **;
 merge a b;
 by pairno;

data ab_food;            **1.3.2  **;
 merge a b;
 by pairno food;

data ba;                 **1.3.3  **;
 merge b a;
 by pairno;
```

```
data b_drop_a;           **1.3.4  **;
 merge b(drop=food) a;
 by pairno;
```

1.3 OUTPUT.

1.3.1 AB BY PAIRNO - NOTE THAT FOOD FROM A IS OVERWRITTEN BY RIGHTMOST DATA SET

Obs	pairno	food	animal	town
1	1	banana	aardvark	BocaRaton
2	2	beans	anteater	Boston
3	3	burrito		Barcelona

1.3.2 AB BY PAIRNO FOOD - NO FOODS MATCH SO MULTIPLE OBSERVATIONS OF PAIRNO ARE CREATED

Obs	pairno	food	animal	town
1	1	apple	aardvark	
2	1	banana		BocaRaton
3	2	avocado	anteater	
4	2	beans		Boston
5	3	burrito		Barcelona

1.3.3 BA BY PAIRNO - SOME FOOD OBSERVATIONS FROM EACH ORIGINAL DATA SET APPEAR IN FINAL FOOD VARIABLE

Obs	pairno	food	town	animal
1	1	apple	BocaRaton	aardvark
2	2	avocado	Boston	anteater
3	3	burrito	Barcelona	

1.3.4 BA BY PAIRNO DROPPING FOOD FROM B - ENSURES THAT FOOD COMES FROM ONLY ONE SOURCE

Obs	pairno	town	food	animal
1	1	BocaRaton	apple	aardvark
2	2	Boston	avocado	anteater
3	3	Barcelona		

1.4 DISCUSSION.

Output 1.3.1 above demonstrates that when multiple data sets are merged containing common variables not in the 'by' statement, the variable from the rightmost data set overwrites any conflicting observations. No observations of the FOOD variable from data set 'a' appear in this final data set because each level of the variable PAIRNO appearing in data set 'a' has a corresponding level in data set 'b' which takes precedence.

Output 1.3.2 above demonstrates what happens when FOOD is added to the 'by' statement. While the PAIRNO observations in data set 'a' each have a corresponding level in data set 'b', no such matches exist in the FOOD variable. Because of this, an observation for each value of FOOD appears in the final output regardless of which data set it originates from. Missing levels of animal and town appropriately appear.

Output 1.3.3 has the seemingly alarming result of containing observations from each data set in the FOOD variable. In reality, the same thing is happening here as in Output 1.3.1 above: SAS is resolving conflicts among data sets by selecting an observation of the variable from the rightmost data set to take precedence over any other corresponding observation from another data set in the merge. The result here is seemingly alarming simply because data set 'a' has no observation with PARINO=3 so the observation from data set 'b' is not overwritten.

Of course these results may be exactly what is desired. However, when a variable is common to multiple data sets which are being merged, and this variable is not included in the by statement, it could be that a conscious decision to drop variables from all but one data set is in order. This is shown in Output 1.3.4 and ensures that FOOD comes from only one source. An alternative would be to rename one or both of the common variables not in the by statement.

The system option MSGLEVEL=I provides notice of overwritten variables. For example, with this option enabled, the following message regarding the 1.3.1 merge above:

```
INFO: The variable food on data set WORK.A will be overwritten by data set WORK.B.
```

The pharmaceutical application of this tip is clear: a basic understanding of merges is crucial regardless of industry. A simplistic example using food, towns, and animals illustrates the industry-independent point free from needless jargon. This allows a novice to the industry or a reader from another industry to focus on the issue rather than struggle with terminology.

2. COMMENTS ABOUT COMMENTS.

There are several methods of including program comments or rendering existing code inactive. Consider the following code and proceeding discussion.

```
1     %let pref=*;
2    * Bring in variables from fruit to container data set.*;
3    %macro norun;
4    Data basket;
5    Set apple(keep=jonathan /*winesap*/  empire gala jonagold);
6    *Set peach;
7     &pref where location='West';
8     run;* cancel;
9     endsas;
10   %mend norun;
11   %*norun;
```

2.1 * FIRST COMMENT.;

Perhaps the most common form of comment is to begin a line with an asterisk. This will 'comment out' text or code until the next semicolon. Of course it is technically not starting a new line with an asterisk that constitutes this type of comment, but an asterisk at the beginning of a statement which could be in the middle of a line. Above, lines 2, 6, 8, and 11 illustrate this. Line 2 is a traditional comment, a description of the following code. Lines 6 and 8 are formerly used code that is inactivated but left in place to illustrate its former use. Line 11 illustrates a way to inactivate a macro that would otherwise be executed. This asterisk method is useful in spanning very short passages since its end, the semicolon, is so common in ending other SAS statements.

2.2 /* SECOND COMMENT. */

A second comment method is to surround the target text with /* and */. Line 5 illustrates this in that the 'winesap' variable is not kept from the 'apple' data set. There is no pairing or nesting with this comment method in that the first instance of */ will end this type of comment regardless of how many /* sequences have preceded it. Another way of saying this is that instances of /* beyond an opening /*, if not preceded with an */ sequence, are ignored. This method is ideal for comments within a line of code as is shown in Line 5 since it does not interfere with the first method. It can also span multiple lines.

2.3 %MACRO THIRD COMMENT. %MEND

A third comment method illustrated in the above code is accomplished by setting apart the target text within a macro definition that is never called. Lines 3 and 10 begin and end, respectively, the 'norun' macro which is not executed. This is a convenient way to 'comment out' large blocks of text which may span multiple, alternative comment methods. While care must be taken when the interaction between this method and actual working macros exist, the good programming practice of separating macro definitions from working code will minimize this potential complication. This method of commenting nests since macros can be defined within other macros. A small negative of this method is the overhead of reading in the macro.

2.4 FOURTH COMMENT.

Lines 1 and 7 illustrate a fourth method of commenting. This is really a version of the first method in that an asterisk is used to begin the comment and the existing semicolon ends the comment. This method involves defining a macro variable near the beginning of a program and interspersing this variable at select locations. The macro variable, assigned to either an asterisk or to nothing, can be switched on or off in a single place. This allows the select lines of code to be either executed or ignored.

2.5 FIFTH COMMENT.

The 'run cancel' of Line 8, when not interrupted by the ';*' sequence, is a fifth method of commenting. It is important to note that this method prevents the procedure from being executed but not from being compiled. For example, an error will result in attempting to print a data set which does not exist whether the print procedure is followed by a 'run;' or by a 'run cancel;' statement.

2.6 OTHER COMMENT METHODS AND A DISCUSSION.

The 'endsas' statement on Line 9 can be considered a method of commenting since code beyond this point is not executed. And some would count the %* macro comment as a method separate from the first method discussed above, the *comment; method.

Another word on the first two methods is in order. The first comment method above (*comment;) is an actual SAS statement which is 'tokenized' and thus in certain situations cannot contain characters such as unmatched quotation marks or semicolons. For example, the following macro results in an unbalanced quote warning, aborting the intended results. However, it is the third line that offends. The single quote mark in the second line is contained in a comment of the form /* comment */ and is thus not 'tokenized.' Oddly enough, if lines two and three are switched, the single quote in the /* comment

*/ balances the single quote in the *comment; so that the macro works as if no comments existed. In general, though, comments of the form /* comment */ are robust in that any character contained within the /* and */ is ignored.

```
%macro qwe;
/* testing semicolons ;;;;; and unbalanced quotation marks '   */
*testing unbalanced quote '  ;
run;
data any;
x=1;
run;
proc print;
%mend qwe;
%qwe;
```

Another example of this (courtesy of Kevin Russell of SAS Tech Support, Feb 2004) is shown below. Commenting out the assignment of the macro variable mvar1 is accomplished simply with the %*comment; method but the unbalanced quote in the mvar2 assignment causes problems when commented out.

```
%* %let mvar1=a;
%* %let mvar2=a";
```

Kevin Russell, crediting an unnamed SAS user, also provides another example of why to be cautious of using the *comment; style comment. To quote Kevin, "If you run this code one time, you will see in the log that the 1st %PUT still executes and the * is stored as text and ends up commenting out the RUN statement. Because the RUN statement is never encountered, the data step, including the CALL SYMPUT, does execute and the macro variable TEST is never created."

```
%macro test;
  data _null_;
  call symput('test','xyz');
  * %put creating the macro variable x;
  run;
%mend;
%test
%put the value of the macro variable x is:  &test ;
```

Finally, it seems to be a matter of preference as to whether or not a semicolon is added as part of a macro call. For example, in the second to last line in the above example, the call to the macro **test** does not end with a semicolon. The only thing this author can add to the discussion is his preference to include a semicolon simply because it makes it quicker to comment out using an asterisk after the percent sign. And as long as we are discussing comments, note the optional identification comment on the %mend statement in the most recent two examples (present in %qwe and absent in %test).

Like the first tip, the value of this tip is not limited to the pharmaceutical industry. At the program development level it is important to know a variety of ways to comment out sections of code, and the pitfalls of each method, regardless of industry or even programming language.

3. MULTILEVEL INDICATOR VARIABLE.

In the output below, an input data set of eight subjects are shown each selecting a different combination of the two choices each of drink, vegetable, and fruit. The definition and output of a multilevel indicator variable (MLIV) is shown which incorporates all the information of the other three variables in one. This variable can be designed in such a way to shorten the selection process of subjects. For example, to select subjects who prefer ocra and guava we could specify veg='ocra' and fruit='guava' or simply MLIV >= 6. The principles used in the creation of this variable are (1) the string operation resolves to a boolean (zero or one) result and (2) the summation of 2^n-based values where n is the number of levels. This latter principle is used in the UNIX command "chmod 764 file" where the 7 is the combination of read=4, write=2, and execute=1 permission for the owner (read, write and execute), the 6 is the combination for the group (read and write but not execute), and the 4 is the combination for others (read only).

```
MLIV=1*(drink='milk')+2*(veg='ocra')+4*(fruit='guava');

Obs    subject    drink     veg      fruit   MLIV

 1        a       milk     ocra      guava     7
 2        b       milk     ocra      mango     3
 3        c       milk     turnip    guava     5
 4        d       milk     turnip    mango     1
 5        e       water    ocra      guava     6
 6        f       water    ocra      mango     2
 7        g       water    turnip    guava     4
 8        h       water    turnip    mango     0
```

A similar concept is sometimes used in defining a population variable if it can be ordered. For example, POP=1 might indicate that the subject was screened, 2 might indicate randomized, 3 that the subject took drug, and 4 that the subject was per protocol. So, a safety table might select POP >=2 while a per protocol table might select POP=4. (A downside of this method is, of course, when strange things happen such as a subject taking drug without being randomized.) Another example from the pharmaceutical industry is to build a variable assigning successive powers of 2 to the presence of a certain concomitant medication: MLIV=1*(drug0)+2*(drug1)+4*(drug2)+ … +2^n*(drug(n)); where drug(n) is either a Boolean expression or an indicator (zero/one) variable itself. Yet another way to use this idea, though it does not involve multiplication, would be to set STATUS=(VIOL1='YES')+(VIOL2='YES')+(VIOL3='YES');. Then VIOLATOR="YES" if STATUS GT 0.

4. TIPS FROM BUILDING A CONCORDANCE.

On first glance it is not readily apparent that a concordance has application in the pharmaceutical industry. Perhaps one could find application in some sort of customized drug interaction tableau or a customized index of a set of standard operating procedures, a statistical analysis plan, or a protocol. And a section of this code has been successfully used to identify aberrant words and phrases in the titles and footnotes of pharmaceutical tables and listings – for details contact the author, search the macro for ‘pharmaceutical’, or stay tuned as this may be developed in future work. However, the author invites the reader to consider this section as an exercise in basic research. What better way to learn SAS than to find and solve a problem in an area of natural interest, learning useful techniques along the way. It is not far-fetched to assert that many readers of this paper developed their SAS skills through the satisfaction of such curiosity – it is certainly true of this author.

A concordance gives the proverbial ‘book, chapter, and verse’ of every word in a passage of literature. To enable identification of specific passages in a large text, the text can be divided into numbered chapters and the chapters can be divided into numbered verses. An unabridged concordance lists all words alphabetically and gives the chapter and verse of each occurrence of each word. A variety of Bible concordances exist representing different translations, levels of abridgment, or subsets (just New Testament, for example), but the present effort was motivated by the desire to have a customized concordance of a single book of the Bible at a level of abridgment of the author’s choosing. It has also been used for a collection of Bible books as well as keying on phrases in addition to just words. Shown immediately below are sample input and output from the New Testament book of Luke. Below this is a discussion of the process used to create the concordance, focusing on handy tips used in each section. Fully operational code is presented in the appendix. The code is shown as it existed before clean-up in order to suggest to the reader the various checks that went into producing the code.

The sample input below is composed of book and chapter identifications (e.g., Luke_ 1), section headings (e.g., The Birth of John the Baptist Foretold), the text including verse identifications, and footnotes. The book identification has been altered with an underscore (Luke_) to distinguish it from identical words within verses: this is not an issue within the book of Luke but does occur in the New Testament book of James.

The principle of this concordance code is to create a SAS data set containing one observation for each word and the corresponding chapter and verse in which it appears. The book and chapter identifications, section headings, and footnotes are processed in order to be available for other outputs, but only the text appears in the concordance. Once the data set has been formed, it is sorted and put statements are used within a data null to create the output shown below.

Sample input:

```
Luke_ 1

Introduction

     1Many have undertaken to draw up an account of the things that have been fulfilled[1] among
us, 2just as they were handed down to us by those who from the first were eyewitnesses and
servants of the word. 3Therefore, since I myself have carefully investigated everything from the
beginning, it seemed good also to me to write an orderly account for you, most excellent
Theophilus, 4so that you may know the certainty of the things you have been taught.
```

```
The Birth of John the Baptist Foretold

     5In the time of Herod king of Judea there was a priest named Zechariah, who belonged to the
priestly division of Abijah; his wife Elizabeth was also a descendant of Aaron. 6Both of them
were upright in the sight of God, observing all the Lord's commandments and regulations
blamelessly. 7But they had no children, because Elizabeth was barren; and they were both well
along in years.

…

17And he will go on
before the Lord, in the spirit and power of Elijah, to turn the hearts of the fathers to their children
and the disobedient to the wisdom of the righteous--to make ready a people prepared for the
Lord."

…

Footnotes

1:1 Or been surely believed
1:15 Or from his mother's womb
1:35 Or So the child to be born will be called holy,
1:69 Horn here symbolizes strength.

Luke_ 2

The Birth of Jesus

     1In those days Caesar Augustus issued a decree that a census should be taken of the entire
Roman world. 2(This was the first census that took place while Quirinius was governor of Syria.)
3And everyone went to his own town to register.

…
```

Sample output:

```
Concordance of Luke's 24,160 words (NIV).

Plural possessives (such as Jesus') included without apostrophe.
Select high frequency words not shown with chapter and verse detail.
Key: WORD [24 chapter frequency] Chapter:Verse (intra-verse frequency if > 1).

A [347] (Select high frequency word)
AARON [1] 1:5

…

MOTHER [18] 1:43, 1:60, 2:33, 2:34, 2:48, 2:51, 7:12, 7:15, 8:19, 8:20, 8:21,
    8:51, 11:27, 12:53 (2), 14:26, 18:20, 24:10
MOTHER-IN-LAW [3] 4:38, 12:53 (2)
MOTHERS [1] 21:23
MOUNT [4] 19:29, 19:37, 21:37, 22:39
MOUNTAIN [3] 3:5, 9:28, 9:37

…
```

4.1 READING THE DATA.

Though there are a variety of ways to read in text, the present method is to have the text reside in an external file and reference the filename associated with this file in an **infile** statement. The **missover** option is used to keep SAS from going to the next line to find values for input variables. An input statement with the **varying** length format is used to read lines of different length. As the goal is to end up with one word per observation, the **tranwrd** function is used to separate two-word units such as is found in Luke 1:17 (righteous--to) into single words by replacing the double hyphen with a space. The **scan** function is used to parse the input line into individual words. An **array** is used along with a **colon operator** to compactly refer to a set of variables.

4.2 PROCESSING THE DATA.

The printout shown below gives a sense of how the text is transformed into the target data set. The **w** variable is the raw unit containing the target word, the verse number (if it is the initial word in a verse), and punctuation and various other non-alphabetic characters to be stripped. The **t**, **c**, and **v** variables record the unit type (H for header, V for verse, and F for footnote; an intermediate stage not shown populates C for chapter) and chapter and verse, respectively. The **wb** and **wc**

variables show advanced stages of the raw unit. The remaining variables are used in producing output. Some special cases such as hyphenated words spanning multiple observations are processed at this stage.

The **indexc** and **index** (determines the position of a character or character string) and **substr** and **compress** (pulls out or deletes a selection of characters from a word) character functions are used to process the input into final form. A **retain** statement is used in assigning chapter, header, verse and footnote status as well as processing hyphenated words such as mother-in-law (found twice in Luke 12:53 – see concordance example above).

w	t	c	v	wb	wc	seq	tn
Introduction	H	1	1	Introduction	INTRODUCTION	1	1
1Many	V	1	1	Many	MANY	2	2
have	V	1	1	have	HAVE	3	2
undertaken	V	1	1	undertaken	UNDERTAKEN	4	2
to	V	1	1	to	TO	5	2
draw	V	1	1	draw	DRAW	6	2
up	V	1	1	up	UP	7	2
an	V	1	1	an	AN	8	2
account	V	1	1	account	ACCOUNT	9	2
of	V	1	1	of	OF	10	2
the	V	1	1	the	THE	11	2
things	V	1	1	things	THINGS	12	2
that	V	1	1	that	THAT	13	2
have	V	1	1	have	HAVE	14	2
been	V	1	1	been	BEEN	15	2
fulfilled	V	1	1	fulfilled	FULFILLED	16	2
among	V	1	1	among	AMONG	17	2
us,	V	1	1	us	US	18	2
2just	V	1	2	just	JUST	19	2
as	V	1	2	as	AS	20	2
they	V	1	2	they	THEY	21	2
were	V	1	2	were	WERE	22	2
handed	V	1	2	handed	HANDED	23	2
down	V	1	2	down	DOWN	24	2

…

certainty	V	1	4	certainty	CERTAINTY	73	2
of	V	1	4	of	OF	74	2
the	V	1	4	the	THE	75	2
things	V	1	4	things	THINGS	76	2
you	V	1	4	you	YOU	77	2
have	V	1	4	have	HAVE	78	2
been	V	1	4	been	BEEN	79	2
taught.	V	1	4	taught	TAUGHT	80	2
The	H	1	4	The	THE	81	3
Birth	H	1	4	Birth	BIRTH	82	3
of	H	1	4	of	OF	83	3
John	H	1	4	John	JOHN	84	3
the	H	1	4	the	THE	85	3
Baptist	H	1	4	Baptist	BAPTIST	86	3
Foretold	H	1	4	Foretold	FORETOLD	87	3
5In	V	1	5	In	IN	88	4
the	V	1	5	the	THE	89	4
time	V	1	5	time	TIME	90	4
of	V	1	5	of	OF	91	4
Herod	V	1	5	Herod	HEROD	92	4
king	V	1	5	king	KING	93	4

4.3 FREQUENCIES AND OUTPUT.

An abridgement feature allows the user to avoid including an exhaustive listing of common word locations. Though this could be automated, it is presently a manual process of viewing word counts from **PROC SUMMARY**. An overall word count is captured into a macro variable via **PROC SQL's SELECT INTO** code. The concordance itself is generated using **PUT** statements inside a **DATA _NULL_** step. And while the present output parameters such as page width are manually controlled, such functions as a **`%let option_ls=%trim(%sysfunc(getoption(linesize)));`** would allow this to be generated dynamically.

4.4 AN ASIDE: HEADINGS.

Prior to verse 1 and 5 in the first chapter of the input above are examples of section headings. A collection of just these headings and the verses they span is an ancillary output to the concordance. **First dot processing** and **concatenating string variables** with the double pipes (||) are featured in this section. A PROC REPORT with the **panels** option is used to produce two column output such as is found in newspapers. Part of this is shown below.

```
  1: 1 -  1: 4  Introduction                    12: 1 - 12:12  Warnings and Encouragements
  1: 5 -  1:25  The Birth of John the           12:13 - 12:21  The Parable of the Rich Fool
                Baptist Foretold                12:22 - 12:34  Do Not Worry
 1:26 -  1:38  The Birth of Jesus Foretold      12:35 - 12:48  Watchfulness
 1:39 -  1:45  Mary Visits Elizabeth            12:49 - 12:53  Not Peace but Division
 1:46 -  1:56  Mary's Song                      12:54 - 12:59  Interpreting the Times
```

4.5 LIMITATIONS.

In its present form, this is a naïve concordance containing nothing about parts of speech or translation from the original language. For example, forms of the word 'mother' appear 19 times in the text. The instances are not grouped into a single entry but occupy separate entries (18 for mother and one for mothers). Also, plural possessives are naively cataloged without the ending apostrophe.

KEY WORDS

Concordance, tips, array, null, put, merge, multilevel, indicator, comment

REFERENCES

ACKNOWLEDGMENTS

The author wishes to thank David J. Austin, Dr. Indra Fernando, John Gorden, Dr. Larry Hoyle, Doug Moore, Paul Rowland, and Kelley Weston for valuable review and suggestions as well as Kevin Russell of SAS Tech Support for technical support above and beyond the call of duty. Thanks also go to Anthony Shaw of Wyeth for suggesting a version of the multilevel indicator variable. 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 author by mail at 9833 Walmer Street, Overland Park KS 66212-1559 or by E-mail at JSMorrill@gmail.com.

BIO INFORMATION

John first used SAS in 1985 and has used it fulltime in the pharmaceutical industry since 1998.

APPENDIX 1. CONCORDANCE.

```
%macro loop(ids=);

filename book "C:\LTC\&ids..txt";

data any;
 length line $800.;
 infile book missover length=reclen obs=55400;
 input line $varying800. reclen;
 line=tranwrd(line,'--',' ');
 array word_(*) $99 word_1-word_25;
 do i=1 to 25;
  word_(i)=scan(line,i,' ');
  end;
if word_1='' then word_1='XXXXX';
run;
** Important to see final column is empty so we know we have enough columns. **;
proc print width=min;
 var word_1-word_3;*word_:;
 *where word_25 NE '';
```

```
run cancel;
data anyt(keep=w);
 set any;
array word(*) word_:;
 do i=1 to 25;*3;
 w=word(i);
 if w NE '' then output;
 end;
run;

* Use this output to identify aberrant words and phrases in the *;
*   titles and footnotes of pharmaceutical tables and listings. *;
title "&ids";
proc freq data=anyt;
 tables w / missing list;
 format w $50.;
run cancel;
title;
run;

data anyt(drop=w_last bin where=(NOT (
  (t='C') OR
  (t='F' AND w='Footnotes') OR
  (w='XXXXX')
  )));
 set anyt;
** Assign status: Chapter, Header, Verse, & Footnote. **;
retain t bin;
if _n_=1 then t='C';
if bin=0 AND t IN ('C' 'V') AND w='XXXXX' then do; t='H'; bin=1; end;
if bin=0 AND t='H'          AND w='XXXXX' then do; t='V'; bin=1; end;
if t='H' and substr(w,1,1)='1' then t='V';
if w='Footnotes'                                then t='F';
if w="&ids._"                                     then t='C';
bin=0;
** Strip footnote bracket. **;
if index(w,'[')>1 then w=substr(w,1,(index(w,'['))-1);
** Obtain verse number. Note multiple delimiters. **;
 s=indexc(lowcase(w),'abcdefghijklmnopqrstuvwxyz"(?',"'");
 sc=indexc(w,'0123456789');
 if (t='V' AND s>1 AND sc>0) then va=substr(w,1,s-1);
** Initialize chapter. **;
 if _n_=1 then do; c=1; v=1; end;
 else if w="&ids._" then c+1;
** Strip non-alphabetic characters from word. **;
if t IN ('V' 'H' /*'F'*/) then do;
 wb=compress(dequote(w),'1234567890{}()?"’”“.!,;:');
 if substr(wb,length(wb),1)="'" then wb=substr(wb,1,length(wb)-1);
 if substr(wb,1,1)="'" then wb=substr(wb,2,length(wb)-1);
 end;
** Handle hyphenated cases. **;
 if substr(wb,length(wb),1)="-" then h=1;
** Remove adjacent empty rows.  Place verse on each row. **;
** Further handle hyphenated words (demon- and mother-). **;
 length w_last $99;
 retain w_last v ha;
  if h=1 then ha=w;
  if ha NE '' and h=. then do;
   wb=compress(ha)||compress(w);
   ha='';
   end;
  if va NE '' then v=input(va,4.);
  if NOT (((w_last='XXXXX') AND (w='XXXXX')) OR (wb='')) then output;
  w_last=w;
run;
data anyt(keep=w: t tn c v seq);
 set anyt(where=(h EQ .));
```

```
 wc=upcase(wb);
 seq=_n_;
retain tn tp;
if _n_=1 then do; tp=t; tn=1; end;
else if t NE tp then do;
 tp=t;
 tn+1;
 end;

%macro headers;
** Header text. **;
data header(keep=tn header);
 set anyt(keep=wb t: where=(t='H'));
 by tn;
 retain header;
 tn=tn+1;
 if first.tn then header=compress(wb);
 else header=trim(left(header))||' '||compress(wb);
 if last.tn then output;
run;
** Header reference. **;
data ref(keep=ref tn);
 set anyt;
 by tn;
 retain beg_c beg_v;
 if first.tn and t='V' then do;
  beg_c=c; beg_v=v;
  end;
 if last.tn and t='V' then do;
  end_c=c; end_v=v;
  ref=put(beg_c,2.)||':'||put(beg_v,2.)||' - '||put(end_c,2.)||':'||put(end_v,2.);
  output;
  end;
 run;
** Header with text reference. **;
data header;*(keep=heading);
 merge header ref;
 by tn;
 heading=trim(left(ref))||' '||trim(left(header));
run;
options ps=74;
proc report panels=2;
 columns ref header;
 define header / flow width=28;
 define ref / width=13;
run;
%mend headers;
%headers;

*proc freq;
* tables va / missing list;
 *tables wc / missing list;
proc print noobs uniform;
*where w NE wb AND c=3 and v=6;
*where indexc(w,"-")>0;
*where (c=8 and v=36) or (c=12 and v=53);
*where va=:'17'; * (String starts with 17. *;
*where w NE wa;*/*(t='V' and s=1) OR*/ indexc(w,'[');* and c=9;
** Check by printing H,F,C before removing unneeded lines in above drop. **;
*where t='F' and w NE 'XXXXX';
run cancel;
*endsas;

data any(rename=(wc=w) keep=c v wc);
 set anyt(where=(t='V'));
run;
proc print;
```

```
 *where w IN ('EVI' 'GADDI' 'GUNI' 'JOGLI' 'SALU' 'SODI' 'TENS' 'TOLA');
 *where index(w,'-')>0;
 *where wi>0;
 *where substr(w,1,3)='DIE';
run cancel;
** Use this to determine word count for title. **;
*proc freq data=any;
* *where index(w,'-')>0;
* tables w / missing list;
* tables /*b*/c*v / missing list;
*run cancel;
proc sql noprint;
 select count(v) into :w_count
  from any;
quit;
* Should not be necessary as sql INTO option should trim automatically. *;
data _null_;
call symput('w_count',compress(&w_count));
run;
%put _all_;
proc sort data=any;
 by c v w;
run;
proc summary data=any;
  *class b c v w;
   class c v w;
   output out=any2;
run;
data anya; set any2;
 if _type_=1;
 freqa=_freq_;
 keep w freqa;
run;
** Used to Select high frequency words.  **;
*proc sort;
* by freqa;
*proc print;

proc sort; by w;
run;
data anyb; set any2;
*if _type_=15;
 if _type_=7;
run;
proc sort; by w;
run;
data anyc; merge anya anyb; by w;
run;
proc sort;
 by w c v;
run;

** New Courier 7.5 font size, L,R=1.5, T,B=.5 **;
options ps=80;
data _null_; set anyc;
 *where freqa > 200;
by w c v;
file print notitles column=q;
     a1=compress("[" || freqa || "]");
     a2=compress(put(c,2.) || ":" || put(v,2.));
        f=compress('(' || _freq_ || ')');
if _n_=1 then do;
put "Concordance of &ids.: &w_count words in NIV." /;
put "Plural possessives (such as Jesus') included without apostrophe.";
put "Select high frequency words not shown with chapter and verse detail.";
put "Key: WORD [total frequency] Chapter:Verse (intra-verse frequency if > 1)." /;
end;
```

```
*titus and ephesians; if w IN (
'IN' 'TO' 'AND' 'OF' 'THE'
) then do;
*amos* if w IN (
'YOUR' 'FOR' 'IN' 'A' 'NOT' 'YOU' 'TO' 'I' 'AND' 'OF' 'WILL' 'THE'
) then do;
*luke* if w IN (
 'AS' 'MY' 'UP' 'SO' 'HAS' 'OUT' 'AT' 'ME' 'DO' 'ARE' 'FROM' 'WERE' 'HAVE' 'THEN'
 'HAD' 'MAN' 'WHAT' 'ALL' 'ONE' 'YOUR' 'ON' 'WHEN' 'WITH' 'THIS' 'THAT' 'BUT' 'SAID'
 'IT' 'NOT' 'THEM' 'WHO' 'I' 'HIS' 'BE' 'WAS' 'FOR' 'IS' 'THEY' 'IN' 'WILL' 'A'
 'HIM' 'HE' 'YOU' 'OF' 'TO' 'AND' 'THE'
) then do;
 if first.w then put @1 w a1 "(Select high frequency word)";
 end;
 else do;
 if first.w then do;
     put @1 w a1 a2 @;
     if _freq_ > 1 then put f @;
     end;
 else
   if q<78 then do;
  *if q<100 then do;
     put +(-1) ", "  a2 @;
     if _freq_ > 1 then put f @;
     end;
   else do;
     put +(-1) ", " / @5 a2 @;
     if _freq_ > 1 then put f @;
     end;
 if last.w then put;
 end;
run;

%mend loop;
%loop(ids=Luke);
%*loop(ids=Amos);
%*loop(ids=Ephesians);
%*loop(ids=James);
%*loop(ids=Titus);
```