STRUCTURED SAS CODE: A PC-aided Approach

Gary N. Griffiths, The Upjohn Company

INTRODUCTION

Software production

Much of today's software is developed with only the short-range benefit in mind: meeting an immediate production deadline. While deadlines are obviously important, there are other long-range considerations which are equally important - - and frequently overlooked in the rush to meet the dreaded production deadline.

"Quick and Dirty"

This short-range goal often forces the programmer to resort to what are sometimes called "quick and dirty" programming techniques. The result: a program which will accomplish the immediate task at hand, but will be difficult (if not impossible) to maintain, modify or audit at some later date.

If considerable time has lapsed between when the program was originally written and when it becomes necessary to maintain, modify or audit, it is even more difficult to figure out the logic and data flow of the "old" program.
Imagine the problem if the original programmer is no longer available.

Management's role

It is the author's contention that good management must consider more than the short-range benefit of meeting a production deadline. The evaluation criteria for a software project needs to include more than the short-range software life cycle, (code & debug, testing, production run).

Management must expand their evaluation criteria to include the long-range software life cycle, (code & debug, testing, production run, maintenance & modification, and auditing results).

The importance of being able to efficiently make modifications to an existing program cannot be over-emphasized. If an existing program (written in a non-structured format) does "almost" what is necessary; yet would require more time to modify than to re-write from the beginning, then I question how much was saved by writing the original program in a non-structured form.

Purpose of this paper

The purpose of this paper is to present some guidelines and software tools for the preparation of SAS programs. These guidelines and tools should aid the programmer in meeting both the short-range production deadlines and the long-range criteria of producing software which may be maintained, modified, and audited.

This paper will not attempt to present universal guidelines for the preparation of SAS code. Its function, instead, is to provide "food for thought" on some programming guidelines which should be considered.

STRUCTURED SAS CODE

Programming guidelines

These guidelines may very well be different from company to company - - and even different for divisions within a company. But every unit needs to have some kind of guidelines for the production of SAS source code.

To illustrate the general structure of the guidelines, the paper concentrates on:

1) selected statements in the data step; and

2) some data manipulation procedures.

At this point, it should be noted that the guidelines were developed for use with SAS software for microcomputers, but apply, with minor modification, to SAS software for mainframe. However, the software tools discussed below may not be available for use with all mainframe editors.

With each suggested guideline, by using a keyboard enhancer such as SuperKey®, the basic template may be produced with only one keystroke. This will allow the programmer to follow the guidelines and is an easy way to become familiar with the general sturcture of the guidelines.

Another reward: It is possible to follow the guidelines and reduce the number of keystrokes necessary to enter the source code for a SAS program.

DATA STEP AND DATA MANIPULATION GUIDELINES

Listed below are guideline templates for the SAS statements: DATA, SET, INPUT, MERGE, BY, and subsetting IF; along with some SAS code segments to illustrate the application of the guidelines. Note that: for all of the templates below, the characters printed in bold type would be entered via a single keystroke (using a keyboard enhancer), and the cursor would be placed at the position indicated by the letter "c". The lower case "t" is used to denote the location of the tabs.

The DATA statement template:

```
t    t    t    t    t    t    t
DATA cxxxxxxx.xxxxxxx     (LABEL= '
                              ) ;
```

Also note that the proper syntax is provided to enter a label for the data set by easily tabbing over to the tab stop just beyond the opening quote mark.

The SET statement template:

```
t    t    t    t    t    t    t    t

  SET cxxxxxxx.xxxxxxx
                    ;
```

In the SET statement the start of the SAS data set name is positioned on a tab stop and on the next line the terminating semicolon is provided.

The INPUT statement template:

```
t    t    t    t    t    t    t    t    t    t

  INPUT     cxxxxxxx    $    xxx   -xxx  .x
            xxxxxxxx    $    xxx   -xxx  .x
            xxxxxxxx    $    xxx   -xxx  .x
                              ;
```

For the INPUT statement important syntax divisions, where source code needs to be entered, are located on tab stops and the terminating semicolon is provided.

The MERGE statement template:

```
t    t    t    t    t    t    t    t    t    t

  MERGE     cxxxxxxx.xxxxxxx
            xxxxxxxx.xxxxxxx
            xxxxxxxx.xxxxxxx
                         ;
```

Perhaps one of the more important statements, which needs to be easy to read, is the subsetting IF statement. In the templates listed below note that important syntax divisions, where one needs to start entering source code, are located on tab stops and only one logical condition is on a line.

The subsetting IF statement template:

```
t    t    t    t    t    t    t    t    t

  IF (xxxxxxxx    RR    xxxxxxxx    )
     CCC
     (xxxxxxxx    RR    xxxxxxxx    )
     CCC
     (xxxxxxxx    RR    xxxxxxxx    )
                                    ;

t    t    t    t    t    t    t    t    t


  IF (    ( xxxxxxxx  RR  xxxxxxxx  )
          CCC
          ( xxxxxxxx  RR  xxxxxxxx  )   )
     CCC
     (  xxxxxxxx RR  xxxxxxxx   )
     CCC
     (    ( xxxxxxxx  RR  xxxxxxxx  )
          CCC
          ( xxxxxxxx   RR  xxxxxxxx   ) )   ;
```

Listed below are three examples illustrating the usage of the guideline templates for a simple DATA step, which reads data from a file, and two DATA steps, which do some subsetting of the data.

EXAMPLE 1

```
DATA   EXAMPLE1          (LABEL =   'Example 1 Data'
                                  );
       INFILE      'G:\DATA\RAWDATA.DAT'    ;

       INPUT       FNAME       $    1    -    10
                   MI          $    15
                   LNAME       $    20   -    34
                   HEIGHT           40   -    42   1.
                   WEIGHT           45   -    48   1.
                   GENDER      $    50

       ;
RUN;
```

EXAMPLE 2

```
DATA   FATMAN            (LABEL =   'Example 2 Data'
                                  );
       SET   EXAMPLE1    ;

       IF    (WEIGHT          >=    285.7 )
             AND
             (HEIGHT          <     55.0  )
             AND
             (GENDER          =     'M'   )
                                              ;
RUN;
```

EXAMPLE 3

```
DATA   THINFAT              (LABEL =   'Example 3 Data'
                                    );
       SET    EXAMPLE1     ;

       IF     (     ( WEIGHT     >=  285.7    )
                    AND
                    ( HEIGHT     <   55.0     )
                    AND
                    ( GENDER     =     'M'    )     )
              OR
              (     ( WEIGHT     <=    105.3  )
                    AND
                    ( HEIGHT     =     66.3   )
                    AND
                    ( GENDER     =     'M'    )     )
                                               ;

RUN;
```

THE IF-THEN-ELSE STATEMENTS

The guideline templates below cover only a few of the many possible forms of the IF-THEN-ELSE statement, however, the general structure of the guidelines is illustrated. The main emphasis of the guidelines is to improve the readability of the code.

The simple IF-THEN template:

```
t    t    t    t    t    t    t    t
  IF c
     THEN
          _____________________ ;
  *END;
```

In the guideline template for the simple IF-THEN structure a comment line "*END;" has been added, which is not required by the SAS language syntax, but is useful in emphasizing the range of the IF-THEN statement.

The multiple statement IF-THEN template:

```
t    t    t    t    t    t    t    t
  IF c
     THEN DO ;
          _____________________ ;
          _____________________ ;
          _____________________ ;
  END;
```

The simple IF-THEN-ELSE statement template:

```
t    t    t    t    t    t    t    t
  IF c
     THEN
          _____________________ ;
     ELSE
          _____________________ ;
  *END;
```

Again, note that the "*END;" has been added only to terminate the range of the IF statement.

The multiple statement IF-THEN-ELSE statement - one level template:

```
t    t    t    t    t    t    t    t    t
  IF c
     THEN DO;
          _____________________ ;
          _____________________ ;
          _____________________ ;
     END;
  ELSE
     DO;
          _____________________ ;
          _____________________ ;
          _____________________ ;
  END;
```

The multiple statement IF-THEN-ELSE statement - two levels:

```
t    t    t    t    t    t    t    t    t
  IF c
     THEN DO;
          _____________________ ;
          _____________________ ;
          _____________________ ;
     END;
  ELSE
     IF
          THEN DO;
               _____________________ ;
               _____________________ ;
               _____________________ ;
          ELSE
               DO;
               _____________________ ;
               _____________________ ;
               _____________________ ;
          END;
*END;
```

EXAMPLE 4

```
IF     GENDER = 'M'
       THEN
            PREFIX = 'Mr.'      ;
       ELSE
            PREFIX = 'Ms.'      ;
*END;
```

EXAMPLE 5

```
IF     HEIGHT >= 60
       THEN DO;
            INDEX1 = HEIGHT * WEIGHT          ;
            INDEX2 = (HEIGHT - 60) * WEIGHT   ;
       END;
ELSE
       DO;
            INDEX1 = 60 * WEIGHT              ;
            INDEX2 = (60 - HEIGHT) * WEIGHT   ;
END;
```

To illustrate the application of the guidelines to data manipulation procedures, guideline templates for the SORT and MEANS procedure are listed below.

The SORT Procedure

```
t    t    t    t    t    t    t    t    t

PROC SORT
     DATA =      cxxxxxxx.xxxxxxx
     OUT  =      xxxxxxx.xxxxxxx
                                  ;
     BY
          [DECENDING]  xxxxxxx
                       xxxxxxx
                       xxxxxxx
                                  ;
RUN;
```

The SORT procedure is an example of a procedure which has both an input data set and an output data set. The guidelines template provides an easy method for specifying both the input and output data set -- the goal of which is to make the SAS code as readable as possible. The RUN command is not always necessary, hovever using a RUN statement does produce an easy to locate end of procedure specifications mark.

The MEANS Procedure

```
t    t    t    t    t    t    t    t

PROC MEANS   DATA = cxxxxxxx.xxxxxxx
                                  ;
     VAR
          xxxxxxx
          xxxxxxx
          xxxxxxx
                  ;
     .
     .
     .
RUN;
```

The MEANS procedure is an example of a procedure for which the input is a data set and the output, much of the time, is a printout. Since the means procedure is an extremely flexible procedure it is not practical to set up rigid and inflexible guideline templates which will handle all possible applications. However, following the general format illustrated above will allow a programmer to produce a more readable program.

CONCLUSION

Implementation

Suppose one decides to adopt some programming guidelines. What sort of implementation difficulities will be encountered.

1) Will additional equipment be necessary? Since it is assumed that SAS software for microcomputers is being used the answer is "no." To use a keyboard enhancer only requires the use of a microcomputer and one of many possible editors.

2) What about additional software? Yes, to easily implement and gain the efficiencies in reducing the number of keystrokes will require purchasing one of the mentioned keyboard enhancers or

perhaps you may be in an operating environment, such as DESQview® which includes a keyboard enhancer. In either case, the capital outlay is small.

3) Will implementing programming guidelines require extensive training? Yes, some training will be required, however the major impediment in implementing structured programming guidelines is "resistance to change".

Resistance to change

In a recent interview with the editor of MANAGEMENT REVIEW, the author of IN SEARCH OF EXCELLENCE, Tom Peters, made the following statements in reply to the question:

In what ways could U.S. corporations manage technology better and do a better job training employees to deal with technology?

Peters: I see this area as a major weak point. The answer is, to me, transparently simple -- and god-awfully difficult to implement. The problem is that it's an attitude issue.[1]

The attitude issue that makes it so difficult to manage technology better is resistance to change. As early as 1948 researchers were looking at the topic of resistance to change and its effects on the work environment.

This is a very human problem; we all have it to some degree, and should. And, resistance to change has been around for a very long time and is not likely to be eradicated any time soon.

But it becomes a very difficult management problem when it is carried to an unhealthy level.

In my Introduction (Management's Role) I stated to correct problems

associated with "quick and dirty" programming that management must expand their evaluation criteria to include the long-range software life cycle. If this sounds too simplistic, it is. Because recognizing the problems associated with non-structured programming, and discovering solutions to the problems, is only half of the issue. Acting on these problems is the other half.

And since:

"Both rational and irrational resistance to change can bring the wheels of progress to a halt."[2]

it makes good sense to recognize its existence and to go to work as a team (managers and programmers) to overcome the unhealthy resistance to change we see in ourselves, as well as that we see in others.

TRADEMARKS

REFERENCES

1. Abetti, Pier A., (Feb 1989), Management Review, "Technology a Key Strategic Resource", p. 38.

2. Kreitner, Robert (1983). MANAGEMENT. Boston: Houghton Mifflin Co, p. 38.