PROGRAMMING FOR JOB SECURITY:

TIPS AND TECHNIQUES TO MAXIMIZE YOUR INDISPENSABILITY

Arthur L. Carpenter

KEY WORDS

STYLE, TECHNIQUE, SECURITY, NAMING CONVENTIONS

ABSTRACT

A great deal has been said about programming techniques for efficiency and maintainability of SAS® programs. We are taught to write code that minimizes machine and programmer resources. Unfortunately, easily maintained code requires fewer programmers, and fewer programmers means pink slips and less job security. In these troubled times, programmers need to be able to maximize their indispensability.

Programmers can increase their job security and thereby protect themselves and their families by applying the tips and techniques discussed within this paper. The programmer will be advised when to apply the techniques, and whether the use of the techniques should be subtle or gross.

Techniques will cover programming style, editing style, statements to use and avoid, and naming conventions. You will learn to blur data steps, make non-assigning assignment statements, and in general write code that not even you will be able to figure out how or why it works.

All the techniques discussed in this paper have been field tested by the author and other SAS programming professionals.

INTRODUCTION

General rules for writing SAS programs have been suggested and promoted for a number of years. When followed, the suggested rules create efficient programs that are easily maintained by either the author or by other programmers. Obviously, the primary beneficiaries of well written code are the stockholders of the company that runs and maintains the programs. In a well ordered and polite society the programmer also receives benefits such as an hourly wage, self satisfaction and an occasional pat on the back.

Well written code is easy to maintain. Flaws in the logic or bugs in the code can be easily ferreted out using the documentation and by understanding the appropriate section of the program. The original programmer may not even be necessary to the process; indeed the original programmer can be freed to work on other tasks that require the special talents of a programmer.

In general, easily maintained programs require fewer programmers and one of the fewer programmers could be you. It is possible to write programs that no one else could possibly maintain. Programs can be written that produce results that cannot be predicted from either a quick or even fairly careful inspection of the code. Once you know these techniques and have learned to properly apply them, your job will be secure for as long as your programs are in use.

Although it is possible to use the described techniques to write a totally indecipherable program that works, often the subtle application of only one or two techniques in an otherwise ordinary program, will achieve the same results. The selection of a technique and its application to the program is an art form that can be achieved with practice and perseverance.

PROGRAMMING STYLE

Programming style refers to the general approach that a programmer takes when designing and coding. Although most programmers develop their own unique programming style, specific guidelines are often imposed by the employer or the client. In the absence of specific guidelines, consider incorporating some of the following.

- Avoid the logical separation of tasks, e.g. separate tasks only if it is logical to keep them together.

- Avoid structured programming approaches.

- Nest function calls whenever possible. Use functions when they are not required.

```
date=
mdy(month(date), day(date), year(date));

lname=
substr(name,index(name,' ')+1,
       length(name));
```

- Write code to replace functions entirely.

```
if a < 0 then b = a*-1;
else b = a;
```

- Nest macro calls and definitions. Macros that call macros that define macros that call macros add a nice touch.

- Symbolic variables may have more than one definition at any given time. These are local and global definitions. The following macros print two different values for &aa. Resetting &aa in the INSIDE macro does not change it in the OUTSIDE macro.

```
%macro inside(aa);
%put inside &aa;
%mend inside;

%macro outside;
%let aa = 5;
%inside(3)
%put outside &aa;
%mend outside;
%outside
```

Prints

```
inside 3
outside 5
```

Most programmers make assumptions about the code they are reading; the objective is to key in on those assumptions.

- Use LENGTH to assign the same variable different lengths in different data sets. This is especially useful if that variable is used in the BY statement during a MERGE.

  More subtle is to define the length of the variables without using the LENGTH statement.

- Variables with the same names could be numeric in one data set and character in another. This is most useful when the variables are flags that take on the values of numbers.

- After compilation eliminate or rename the SCL source code used with SAS/AF® or FSEDIT applications.

- When editing SCL for FSEDIT change the color of the code to match the background.

EDITING STYLE

The layout of the program as seen by the programmer is determined by the programmer's editing style. Once again, in the absence of specified guidelines, consider these to improve the unreadability of your code.

- Never indent.
- Use multiple statements per logical line.
- Break statements in the middle.
- Form your code to make pictures like your company logo.

Occasionally use columns > 80 in the editor (these columns are not usually visible while editing e.g. PROGRAM WINDOW in the DISPLAY MANAGER).

- Place key variables out of sight.

```
                                    80
data newdata;
set olddata (keep=name              | fname
       address city state);         |
```

- Use the asterisk to comment out key formulas or statements.

```
data new; set old;                  | *
   wt = wt/2.2;                     |
```

- Place special equations entirely out of sight.

```
data new; set old;                  | x=x+5;
```

- Place special equations partially out of sight.

```
data new; set old;                  |
degc = (degf                        | +5
            -32)*5/9;               |
```

Comments are very useful when used correctly.

```
* The comments in this section do more
* than it seems;

* modify data to prep for; proc means  ;
* after adjusting the data using
* the; var for weight;
```

CHOICE OF STATEMENTS

Some statements when used properly have the capacity to add several layers of complexity to a SAS program. Others should be avoided as they tend to reduce confusion.

Statements to Avoid

Avoid statements that tend to allow the programmer to retain less mental information.

- Maintain extra variables in the Program Data Vector (PDV). Avoid the use of the KEEP and DROP statement.
- When variables must be eliminated from the PDV use the DROP statement. The KEEP statement lets the programmer know what variables remain in the PDV, while the DROP statement only reveals what was eliminated.
- Comments are to be avoided unless used in the ways mentioned above.
- RUN and QUIT statements are rarely required, and usually only serve to reduce program complexity by separating steps.

Statements to Use

Several statements and combinations of statements can be used to advantage.

- Statement style macros can be used to redefine variables.

```
* hide this definition;
%macro sat (name) / stmt;
set &name;
wt = wt + 5;
%mend sat;

data old;wt=1;output;
data new;sat old;
```

- Define arrays using array names. This technique was used on purpose prior to the advent of multidimensioned arrays.

```
array a (I) wt ht;
array b (I) xwt xht;
array c (j) a b;
x = c;
```

- Implicit arrays are preferred to explicit arrays.
- The GOTO statement can be used to breakup program structure; use it liberally.
- The label statement can be disguised. SAS statement key words make excellent labels.

```
do: I=1;

* did you notice that, this comment
* contains a; label:
```

Data steps with multiple SET statements or better yet SET and MERGE statements add complexity quickly, but not quietly.

NAMING CONVENTIONS

Naming conventions are perhaps the most useful tool in the arsenal of the job security specialist. A novice may think that no naming conventions or the random selection of names will create the most secure program. In actuality there are a number of subtle and not so subtle techniques which can be applied by the security specialist programmer. These rules can be applied to both variables and data sets, and come from the schools of confusion, misdirection, and inconsistency.

Confusion

- Use 8 digit names when possible.
- Avoid the use of vowels, include subtle variations.

  ```
  QWRTXZQR, QWRTZXQR, QWRZTXQR
  ```

- Some letters go well together.

  ```
  H & I   HHHIIHIH HHIHIHIH

  V & W   WVWVWVVW WVWWVWVW

  l & 1   testnum1 testnuml
  ```
  (number 1 and lower case L)
  ```
  0 & O   test0001 testOOO1

  Z & 2   QWRTZXQR QWRT2XQR
  ```

- Use of SAS statement key words

  ```
  DATA SET; SET DATA; DO = 5+ TO -15;
  ```

Misdirection

- Obvious names work well for other purposes.

  ```
  SEX    ===> number of fish caught

  WEIGHT ===> height of patient

  INCHES ===> height in centimeters
  ```

- Placement of observations and data set names.

  ```
  IF SEX = 'MALES' THEN OUTPUT FEMALES;
  ```

- The LABEL statement can be used to provide information to the user e.g. if the variable SEX contains the number of fish caught then:

  ```
  LABEL sex = 'Sex of the Patient';
  ```

- Variable and data set names should not have unique definitions throughout the program.

- Variables should on occasion disappear and reappear later with different definitions.

Inconsistency

Subtle inconsistencies are very difficult to detect and can be very useful, especially in a program that has clearly defined parameters. YES/NO variables should take on the values of YES=0 and NO=1 (of course never Y & N), except somewhere for some variable that has YES=1 and NO=0.

BLURRING THE DATA STEP

The SAS supervisor recognizes the end of steps by detecting a RUN, QUIT, or the start of the next step. We already know not to use the RUN or QUIT, so all we need to do to blur two steps into one is to hide the start of the second step. Comments work well for this.

```
* start of the step;
data new ; set old;
x = 5* y;

* this starts the second step
data second; set gudstuff;
x= zzstuff;
```

The data set NEW in this example will have the variable x=zzstuff as read from the data set GUDSTUFF. Notice that the second comment has no ; hence the data set SECOND is never replaced. This data step has two SET statements which is usually good for a few laughs by itself.

Unbalanced quotes can also be used to advantage.

```
* start of the step;
data new;y=5;frankwt=0;x=5*y;
length name $6;
name='weight;
data second;set gudstuff;
*for weight use Franks';
x=frankwt;
proc print;run;
```

The unbalanced quote for the variable NAME completely masks the creation of the data set SECOND and the use of the data set GUDSTUFF. Unbalanced quotes can also be used in LABEL statements and to mask the end and beginning of macro definitions.

SUMMARY

In reality there is a constant demand for SAS programmers. Just knowing how to write good, clean, and tight SAS code provides a high level of job security. Obviously, the techniques discussed in this paper are to be avoided; however, they have all been encountered in actual programs. Usually they were the result of accidents and ignorance, but no matter the source, they still caused problems. Being aware of the potential of these types of problems is a major step in the direction of writing better code.

ABOUT THE AUTHOR

Arthur L. Carpenter has over nineteen years of experience as a statistician and data analyst and has served as a senior consultant with California Occidental Consultants, CALOXY, since 1983. His publications list includes a book on SAS/GRAPH®, a number of papers and posters presented at SUGI, and he has developed and presented several courses and seminars on statistics and SAS programming.

CALOXY offers SAS contract programming and in-house SAS training nationwide.

AUTHOR

Arthur L. Carpenter
California Occidental Consultants
PO Box 6199
Oceanside, CA 92058-6199

(619) 945-0613

email: 72212.211@compuserve.com

PRESENTER

This paper was presented at SEUGI '97, Madrid, by:

Tony Payne
Software Product Services Limited
19-20 The Broadway
Woking, Surrey, GU21 5AP, UK.

Tel: +44 (0)1483 730771.
Fax: +44 (0)1483 727417.
Mail: tpayne@sps-uk.co.uk
URL: http://www.sps-uk.co.uk

Many thanks to Art for allowing me to present this paper.

TRADEMARK INFORMATION