**Introduction**

The life expectancy of most application systems is approximately five years. The SAS System, however, is upgraded continually with major releases every three years or so. In addition, SAS users are a diverse group with a wide variety of hardware resources at their disposal. These platforms also change significantly every two to three years. Different hardware platforms possess the potential to generate problems unique to those respective systems. So, it follows that conversions, between versions of SAS or between hardware platforms, are major elements in the life of a SAS programmer.

This paper presents three different types of puzzles:

- General puzzles that we find particularly interesting
- Platform conversion puzzles
- SAS version conversion puzzles

**A Few General Puzzles To Get Us Going**

**PUZZLE #1: BUT THAT'S THE NAME I WANT TO USE**

In developing large macro-based systems, good programming practices require a consistent pattern of mnemonics for macro variables. It is not unreasonable to use a "SYS" mnemonic prefix to define global macro variables. What does the following code generate?

```sas
%macro test(sysbob=);
  %put &sysbob;
  %let sysbob=jumped over the lazy cow;
  %put &sysbob;
%mend test;
%test(sysbob= The quick brown fox);
```

**PUZZLE #2: I THOUGHT YOU COULDN'T DO THAT**

Macro variables and macro symbol tables continue to mystify all but the most jaded of SAS programmers. Can you put a macro variable referenced only within a macro definition into the global symbol table without using a `GLOBAL` statement?

**PUZZLE #3: FOR THOSE WHO MIGHT WANT TO COMMENT THEIR MACRO CODE**

Commenting your code is critical in developing applications that have a life span longer than you will be assigned to the project. Commenting becomes interesting when you want to include special cases in your comment. Any SAS experienced programmer knows that quoting is a especially special case. What happens to single quotes in a macro comment? Here are the three different styles of comments that you might use:

```sas
%macro test1;
  "The quick brown fox jump'd"; // PL/1 STYLE
%mend;
%macro test2
  "The quick brown fox jump'd"; // SAS STYLE
%mend;
%macro test3
  "The quick brown fox jump'd"; // MACRO STYLE
%mend;
```

```sas
data _null_;
  REPEAT CODE FOR EACH
  %test1;
  "put 'over the lazy cow';"
run;
```

**PUZZLE #4: I THOUGHT THIS WASN'T SUPPOSED TO WORK**

To really confuse programmers who follow us working on a project, it is helpful to use as many ampersands as possible in defining macro variables. Secondarily, this introduces layers of flexibility in your code. Rarely, you can introduce this flexibility without the aid of the second ampersand.

```sas
%let state1=HAWAII;
%LET l=1;
%PUT &STATE1; // "THIS DOESN'T WORK"
%PUT &STATE1; // "THIS IS HOW YOU SHOULD DO IT"
%PUT "&STATE1"; // "THIS SHOULDN'T WORK BUT DOES"
```

**PUZZLE #5: OUR FAVORITE PUZZLES PROC**

This Code: `data test;
  do col=1 smart 4;
    var1=ranuni(612);
    var2=ranuni(730);
    output;
  end;
run;
proc tabulate data=test format=6.0;
  class col;
  var var1 var2;
  table var1*pctsum var2*pctsum,
         col;
  keylabel pctsum=' '; run;
` Produces this output:

<table>
<thead>
<tr>
<th></th>
<th>COL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>VAR1</td>
<td>161</td>
</tr>
<tr>
<td>VAR2</td>
<td>36</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>VAR1</td>
<td>51</td>
</tr>
<tr>
<td>VAR2</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>25</td>
</tr>
</tbody>
</table>

Please remove the columns of blanks.
SAS VERSION CONVERSION PUZZLES

PUZZLE #6: NEW FORMAT ON THE BLOCK

Converting from proportions to percents for reporting purposes is a widely required function. SAS has provided a new format PERCENT which can be used to eliminate this code completely. What is the smallest format width needed to print the value 1 with a percent format?

PUZZLE #7: YOU WANT YOUR LOG TITLE TO DESCRIBE YOUR JOB?

In SAS Version 6.07 the default title for the log is

"The SAS System"

instead of the more descriptive Version 5 title of

"SAS(R)_LOG OS SAS5.18 ...<jobname>...".

If you have a job that reads the SAS log for information and are depending on the old Version 5.18 title, how to you get this title in Version 6.06?

PUZZLE #6: REUSING VARIABLE NAMES

In 5.18 the rename statement used on an input data set renames the variables in the order in which they appear in the statement. In 6.06, however, variables are renamed in the order of their position in the PDV. Suppose the contents of the SAS data set, OLD, reads as follows:

<table>
<thead>
<tr>
<th>#</th>
<th>variable</th>
<th>type</th>
<th>length</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>one</td>
<td>char</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>two</td>
<td>char</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The following statements:

```
DATA NEW;
SET OLD (RENAME=(TWO=THREE ONE=TWO));
RUN;
```

executes without error in 5.18. Version 6 generates the following message:

ERROR: VARIABLE TWO ALREADY EXISTS ON FILE WORK.OLD.
ERROR: Invalid DROP, KEEP, or RENAME option on file WORK.OLD.
NOTE: The SAS System stopped processing this step because of errors.

This error occurs because SAS Version 6.06 is first trying to rename the variable ONE to TWO, because variable ONE is positioned before the variable TWO in the PDV. What to do?

PUZZLE #9: VERSION/PLATFORM PUZZLES

IF YOU DON'T WRITE GOOD SAS CODE, YOU HAD BETTER WORRY ABOUT WHAT PLATFORM YOU'RE ON

The following sets of code perform differently under different operating systems and different SAS versions. Each set of code is poorly written. They reflect instances of typographical mistakes, lack of testing in data driven code, and mismatched data types.

Focus on these problems with the following scenario in mind. Your Boss decides to reduce mainframe charges by moving all SAS development to PCs and then transfer the code to the mainframe only for final testing and production. What kinds of problems will you encounter as you transfer working code from one platform to the next? If your code is well written and tested, then you will have little, if any, problems. If your code is like most of the code found in the real world, then you may have problems.

Please identify the resultant output with the following SAS versions:

VMS 6.06
VMS 6.07
PC 6.04
OS/2 6.06
UNIX 6.07
MVS 6.06
data test (drop=nowisthetime);
run;

data null;
x=.;
y=2;
do n=x to y;
put _all_;
end;
run;

ERROR: The variable NOWISTHETIME in the DROP, KEEP, or RENAME list has never been referenced.
NOTE: The data set WORK.TEST has 1 observations and 0 variables.

NOTE: The data set WORK.TEST has 0 observations and 0 variables.

ERROR 213:322: Variable name NOWISTHETIME is longer than 8 characters.
NOTE: The SAS System stopped processing this step because of errors.

data test;
array nomatrix _numeric_;
do over nomatrix;
if nomatrix>32;
end;
run;

ERROR: Invalid DO loop control information, either the INITIAL or TO expression.
NOTE: The SAS System stopped processing this step because of errors.

data test;
array nomatrix _numeric_;
do over nomatrix;
if nomatrix>32;
end;
run;

WARNING: Defining an array with zero elements.
I=1 _ERROR_ =0 _N_=1
NOTE: The data set WORK.TEST has 1 observations and 0 variables.

WARNING: Defining an array with zero elements.
%SAS-F-INTSASERR,...
%SAS-F-CALTECH,...
%SYSTEM-F-ACCVIO,...
data test;
  do i=1 to 10;
    output;
  end;
run;

proc print data=test;
  options obs=3;
run;

ERROR 117-185: There were 1 unclosed DO blocks.
NOTE: The SAS System stopped processing this step because of errors.
NOTE: SAS set option OBS=0 and will continue to check statements....

NOTE: There were 1 unclosed DO blocks.
NOTE: The data set WORK.TEST has 1 observations and 1 variables.

OBS  1
  1  1
  2  2
  3  3
  4  4
  5  5
  6  6
  7  7
  8  8
  9  9
  * 10

OBS  1
  1  1
  2  2
  3  3
  4  4
  5  5
  6  6
  7  7
  8  8
  9  9
  10 10
data _null_;  
x='1';  
y='3';  
do i=x to y;  
   put_all;  
end;  
run;

NOTE: Character values have been converted to numeric values 1 at 0:0

ERROR 289-185: The TO value for the DO statement must be numeric.
NOTE: The SAS System stopped processing this step because of errors.

NOTE: Character values have been converted to numeric values 4:6 4:11
X=1 Y=3 I=1 _ERROR_=0 _N_=1  
X=1 Y=3 I=2 _ERROR_=0 _N_=1  
X=1 Y=3 I=3 _ERROR_=0 _N_=1  

NOTE: The data set WORK.TEST has 1 observations and 1 variables.

WARNING: The variable C in the DROP, KEEP, or RENAME list has never been referenced.
NOTE: The data set WORK.TEST has 1 observations and 1 variables.

NOTE: Invalid second argument to function SUBSTR
LONG=YN SHORT=1 NEW=1 _ERROR_=1 _N_=1  

NOTE: Invalid second argument to function SUBSTR
LONG=YN SHORT=Y NEW=Y _ERROR_=1 _N_=1
ANSWERS

Puzzle #1:
The example code generates the following log:

```sas
%macro test(sysbob=);
%put &sysbob;
%let sysbob=jumped over the lazy cow;
%put &sysbob;
%mend test;
%test{sysbob= The quick brown fox);
ERROR: Attempt to %GLOBAL a name (SYSBOB) which exists in a local environment.
The quick brown fox
SAS treats macro variables that begin with SYS as special case macro variables. You may not change the value of SYS* macro variables even if you created them. If you change the SYS portion of the macro variable name, then things occur as you would expect.

%macro test(macbob=);
%put &macbob;
%let macbob=jumped over the lazy cow;
%put &macbob;
%mend;
%test(macbob= The quick brown fox);
jumped over the lazy cow
```

The quick brown fox
SAS treats macro variables that begin with SYS as special case macro variables. You may not change the value of SYS* macro variables even if you created them. If you change the SYS portion of the macro variable name, then things occur as you would expect.

Thus, for global variables you should use a mnemonic representing the system being designed rather than a more generic SYS.

Puzzle #2:
How can you put a local macro variable into the global table?

Use CALL SYMPUT in a macro with neither a %LET statement or calling parameters. Local macro symbol tables are generated only in those two cases. If neither is present, then the local symbol table is not generated and when the CALL SYMPUT is processed SAS puts the resultant macro variable in the global symbol table--its only alternative.

This has particular interest in that if the macro software does not force the creation of the local symbol table, then your "local" macro variables are, in fact, global. This may result in unexpected macro variable values. This, in turn, only affects your code when you really don't want it to.

```sas
%macro test;
%let one::one;
%put &one;
%mend;
%test
WARNING' Apparent symbolic reference ONE not resolved.

%macro test2;
data _null_; call symput('two', 'two'); run;
%MEND;
%TEST2;
NOTE: The DATA statement used 2.00 seconds.

%macro test3;
data _null_; call symput('two', 'two'); run;
%MEND;
%TEST3;
NOTE: The DATA statement used 1.00 seconds.
```

Puzzle #3:
The safest bet is to use PL/1 style comments everywhere. They are handled in every case. This is how SAS dealt with each of the different types of comments within a macro:

PL/1 STYLE COMMENTS

```sas
%macro test1;
"The quick brown fox jump'd"
%mend;
data _null_; %test1;
%put 'over the lazy cow'; run;
```

SAS STYLE COMMENTS

```sas
%macro test2;
"The quick brown fox jump'd"
%mend;
```

MACRO STYLE COMMENTS

```sas
%macro test3;
""/The quick brown fox jump'd"
%mend;
```

ARC encounters this problem primarily in block comments within macro modules. In the block comments, it is not uncommon for programmers to be developing text materials for external program documentation. Exceptions and special cases are often discussed at length in block comments. Thus it is more likely that contractions or label and title examples ("SAM LORD'S CASTLE") will be used.
Puzzle #4

Double quotes with imbedded macro variables cause SAS to recursively scan the text string until no macro variables can be resolved. In this example, during the first scan of the text string, &STATE is not resolved and &1 is resolved. During the second scan, &STATE1 is resolved to HAWAII.

Puzzle #5

Within the PROC TABULATE section of code you need to include ROW=FLOAT as a Table statement option.

```
proc tabulate data=test format=6.0;
  class col;
  var var1 var2;
  table var1"pctsum var2·pctsum,
    col / row=float;
  keylabel pctsum=' ';
run;
```

Page 20 of the SAS Guide to TABULATE Processing reads:

> When ROW=CONSTANT, all row title elements have
> space allotted to them, even if the title has been blanked out. When ROW=FLOAT, the row title space is divided equally among the nonblank title elements in the crossing.

The moral of the puzzle is read the manuals and look at the option defaults. The answers are almost always there. You just have to know what you are looking for and then how to find it. Of course this implies that you have all the manuals and that you can find them.

Puzzle #6

6 (PERCENT). There are several points of interest here:

- The percent format interprets the data as a proportion and thus converts the 1 into a 100. DO NOT use the PERCENT format with PERCENTS.
- You have to add another column for the percent sign.
- The tricky part is that two additional columns are required for printing the two parentheses used to indicate negative percents (these are not printed with positive numbers). Since these are not printed most of the time, they are more likely to be forgotten.

Puzzle #7

The old-style LOG Title can be obtained by specifying the system options "$LOGNOTE1 = $" at the SAS invocation:

```
// EXEC SASG06,OPTIONS="$LOGNOTE1 = $"
```

This will give you "SAS LOG(r) OS SAS 6.06 ... <jobnames>..." which is similar to Version 5.

Puzzle #8

The solution is to ensure that the PDV is built appropriately for the code that follows. If your code requires that the variables be processed in a particular order, then you must take programming steps to ensure the variables will be in that order in the PDV.

A simple LENGTH statement before the SET statement will suffice. Another approach would be to explicitly code the desired transformation using assignment statements.

Good programming practices require that your software NOT depend on structure information within the PDV or within the data unless your program explicitly controls the structure. If you don't control the structure, it will be changed and your software will fail.

Acknowledgements

The authors wish to thank the ARC staff (especially Felix Meale, Jeff Phillips, Warren Repeol and Lee-Ann Taylor) for their comments on earlier drafts of this paper. Certainly we would like to thank Judie Mopsik for encouraging the staff to develop this paper and allowing us to present it in HAWAII.

The authors can be contacted at:

Atlantic Research Corporation
Information Systems Division
1301 Piccard Drive, 2nd Floor
Rockville, MD 20850
(301) 258-5533
Puzzle #9

data test (drop=nowisthetime);
  run;

PC 6.04
ERROR: The variable NOWISTHE in the DROP, KEEP, or RENAME list has never been referenced.
NOTE: The data set WORK.TEST has 1 observations and 0 variables.

VAX/VMS 6.06
NOTE: The data set WORK.TEST has 0 observations and 0 variables.

VAX/VMS 6.07
ERROR 213.322: Variable name NOWISTHETIME is longer than 8 characters.
NOTE: The SAS System stopped processing this step because of errors.

OS2 6.06; UNIX 6.07; MVS 6.06
NOTE: The SAS System stopped processing this step because of errors.

PC 6.04
X=. Y=5 N=0 _ERROR_=0 _N_=1
X=. Y=5 N=1 _ERROR_=0 _N_=1
X=. Y=5 N=2 _ERROR_=0 _N_=1
X=. Y=5 N=3 _ERROR_=0 _N_=1
X=. Y=5 N=4 _ERROR_=0 _N_=1
X=. Y=5 N=5 _ERROR_=0 _N_=1

VAX/VMS 6.06, VAX/VMS 6.07, OS2 6.06, UNIX 6.07; MVS 6.06
ERROR: Invalid DO loop control information, either the INITIAL or TO expression.
X=. Y=5 N= _ERROR_=1 _N_=1
NOTE: The SAS System stopped processing this step because of errors.

PC 6.04; OS2 6.06; UNIX 6.07; VAX/VMS 6.07
WARNING: Defining an array with zero elements.
I=1 _ERROR_=0 N=1
NOTE: The data set WORK.TEST has 1 observations and 0 variables.

VAX/VMS 6.06; MVS 6.06
WARNING: Defining an array with zero elements.
%SAS-F-INTSASERR...
%SAS-F-CALLTECH...
%SYSTEM-F-ACCVIO...
data test;
  do i=1 to 3;
    output;
  end;
run;

proc print data=test;
  options obs=3;
run;

NOTE: There were 1 unclosed DO blocks.
NOTE: The data set WORK.TEST has 1 observations and 1 variables.
data null;
  x='Y';
  y='3';
  do i=x to y;
    put _all_;
  end;
run;

PC 6.04
NOTE: Character values have been converted to numeric values
1 at 0:0

VAX/VMS 6.06; MVS 6.06; OS2 6.06; UNIX 6.07; MVS 6.06
ERROR 289-185: The TO value for the DO statement must be numeric.
NOTE: The SAS System stopped processing this step because of errors.

VAX/VMS 6.07
NOTE: Character values have been converted to numeric values
4:6:4:11
X=1 Y=3 I=1 _ERROR_=0 _N_=1
X=1 Y=3 I=2 _ERROR_=0 _N_=1
X=1 Y=3 I=3 _ERROR_=0 _N_=1

data test (keep=a c);
  a=1;
  b=1;
run;

VAX/VMS 6.06; OS2 6.06; UNIX 6.07; MVS 6.06
NOTE: The data set WORK.TEST has 1 observations and 1 variables.

PC 6.04; VAX/VMS 6.07
WARNING: The variable C in the DROP, KEEP, or RENAME list has never been referenced.
NOTE: The data set WORK.TEST has 1 observations and 1 variables.

data null;
  long='YN';
  short='1';
  short=long;
  new=substr(short,3,1);
  put _all_;
run;

PC 6.04
NOTE: Invalid second argument to function SUBSTR
LONG='YN' SHORT='1' NEW= _ERROR_=1 _N_=1

VAX/VMS 6.06; VAX/VMS 6.07; OS2 6.06; UNIX 6.07; MVS 6.06
NOTE: Invalid second argument to function SUBSTR
LONG='YN' SHORT='1' NEW= _ERROR_=1 _N_=1