Paper 185-2013

Coaching SAS® Beginner Programmers: Common Problems and Some Solutions

Peter Timusk, Centre for Special Business Projects, Statistics Canada, Ottawa, Ontario Canada

ABSTRACT

This paper will present a number of problems SAS beginner programmers encounter when first writing SAS programs. The paper will cover three cases and show how pointing out patterns to beginner programmers will aid them in avoiding errors in their SAS code.

Here are the three problems we will look at in this paper.

1. The IF statement and the order of operations of logic operators and the use of parentheses.
2. The different between speaking a logic expression and how to code a logic expression with an example from setting ranges for displaying a variable's count in five columns.
3. In the final example a complex assignment is presented to the beginning programmer and they do not know how to proceed because they are overwhelmed with the complexity. Then this examples shows how seeing patterns in the specifications can help make the work straight forward for a beginner and make a complex specification simple.

INTRODUCTION

This paper will present a number of problems SAS beginning programmers encounter when first writing SAS programs. The paper will cover three cases and show how pointing out patterns to beginner programmers will aid them in avoiding errors in their SAS code.

Here are the three problems we will look at in this paper.

1. The IF statement and the order of operations of logic operators and the use of parentheses.
2. The different between speaking a logic expression and how we code a logic expression with an example from setting ranges for displaying a variable's count in five columns.
3. The final example looks at a complex assignment and shows how seeing patterns in the specifications can help make the work straight forward for a beginner and make a complex specification simple.

The paper explains how showing programmers patterns can aid in their avoiding these problems. By seeing patterns complex programming problems are made much simpler.

We will look at three examples.

Example 1: The IF statement and the order of operations of logic operators and the use of parentheses.

In this example, an IF statement should group two related groups of variables in a balanced structure as SAS evaluates their values. Here is the statement coded correctly.

```
IF (A1=1 or A2 = 1) and (B1=2 and B2=2 and B3=2) THEN Var1=1; ELSE Var1 =0;
```

In this case, both the conditions concerning the A variables and the B variables need to be true, so the code surrounds these related variables with parentheses.

The beginner programmer will often code this without parentheses in this way:

```
IF A1=1 or A2 = 1 and B1=2 and B2=2 and B3=2 THEN Var 1=1; ELSE Var1 =0;
```

The problem here is that if A1=1 the evaluation will stop here and not even consider the B series variables. In the intended code that SAS evaluator must evaluate the B variables and these must all be 2 before Var1 will be set to 1.

The pattern to see here to avoid the error can be the similar letters A and B. An alternative pattern that would help is to see the different “or” and “and” statements and take care in grouping these as this is really the key reason the mistakes fails. Being aware of the logical operators and how these w0ork can take some exposure to formal logic or programming courses that would cover logic.

Example 2: The different between speaking a logic expression and how to code a logic expression with an example from setting ranges for displaying a variable’s count in five columns.

Here is the correct coding we are looking for:

```
IF Var1 = 0 THEN Col_1 = 1 ; ELSE Col_1 = 0 ;
IF 0 < Var1 <= 1 THEN Col_2 = 1; ELSE Col_2 = 0 ;
IF 1 < Var1 <= 2 THEN Col_3 =1;  ELSE Col_3 = 0 ;
IF 2 < Var1 <= 3 THEN Col_4 = 1 ; ELSE Col_4 = 0;
IF Var1 > 3 THEN Col_5 = 1; ELSE Col_5 = 0;
```

Here is the way this code is mistakenly programmed as though it were spoken:

```
IF Var1 = 0 THEN Col_1 = 1 ; ELSE Col_1 = 0 ;
IF 0 < Var1 and <= 1 THEN Col_2 = 1; ELSE Col_2 = 0 ;
IF 1 < Var1 and <= 2 THEN Col_3 =1;  ELSE Col_3 = 0 ;
IF 2 < Var1 and <= 3 THEN Col_4 = 1 ; ELSE Col_4 = 0;
IF Var1 > 3 THEN Col_5 = 1; ELSE Col_5 = 0;
```

This could be fixed using this code instead:

```
IF Var1 = 0 THEN Col_1 = 1 ; ELSE Col_1 = 0 ;
IF 0 < Var1 and Var1 <= 1 THEN Col_2 = 1; ELSE Col_2 = 0 ;
IF 1 < Var1 and Var1 <= 2 THEN Col_3 =1;  ELSE Col_3 = 0 ;
IF 2 < Var1 and Var1 <= 3 THEN Col_4 = 1 ; ELSE Col_4 = 0 ;
IF Var1 > 3 THEN Col_5 = 1; ELSE Col_5 = 0 ;
```

The problem here is speaking the code and saying, Var1 is greater than 1 and less than or equal to 2. Again once the pattern seen in this example is show to the beginner programmer they are better able to write the code when needed.

Example 3: The final example looks at a complex assignment and shows how seeing patterns in the specifications can help make the work straight forward for a beginner and make a complex specification simple.

The third example explores the idea of patterns guiding programming a bit further. Here the problem is not a mistake in code. The beginning programmer being assigned specifications for a table that at first glance looks very confusing and complex. The beginning programmer who worked on programming this table was stumped at first and not sure they could do the work. In this case, after being shown the patterns the programmer found no problem with coding this table.

Here is a sample version of our specifications from the client.

Table 1 A Complex Specification:

	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7	Column 8	Column 9	Column 10	Column 11	Column 12
Row 1	A45060	A47060	C45060	C47060	A45070	A47070	C45070	C47070	B55070	B57070	D55070	D57070
Row 2	A45061	A47061	C45061	C47061	A45071	A47071	C45071	C47071	B55071	B57071	D55071	D57071
Row 3	A45062	A47062	C45062	C47062	A45072	A47072	C45072	C47072	B55072	B57072	D55072	D57072
Row 4	A45063	A47063	C45063	C47063	A45073	A47073	C45073	C47073	B55073	B57073	D55073	D57073
Row 5	A45064	A47064	C45064	C47064	A45074	A47074	C45074	C47074	B55074	B57074	D55074	D57074
Row 6	A46060	A48060	C46060	C48060	A46070	A48070	C46070	C48070	B56070	B58070	D56070	D58070
Row 7	A46061	A48061	C46061	C48061	A46071	A48071	C46071	C48071	B56071	B58071	D56071	D58071
Row 8	A46062	A48062	C46062	C48062	A46072	A48072	C46072	C48072	B56072	B58072	D56072	D58072
Row 9	A46063	A48063	C46063	C48063	A46073	A48073	C46073	C48073	B56073	B58073	D56073	D58073
Row 10	A46064	A48064	C46064	C48064	A46074	A48074	C46074	C48074	B56074	B58074	D56074	D58074

This presents a complex specification that at first may appear to be just a sea of letters and numbers. Although in this public example the rows and columns do not have titles, in this case the real titles did not help much in figuring this out. What did help was being shown the repeated patterns.

Notice that columns 1, 2, 5 and 6 have variable names that start with the same letter, A. Also note the same pattern two columns over with the letter C. This is again seen as part of the pattern in the last four columns with the letters B and D.

Another pattern is in the columns of the second and third digits. Columns 1, 3, 5 and column 7 both have five rows of 45 and then five rows of 46. Similarly columns 2, 4 6 and 8 have five rows of 47 and then five rows of 48. The last four columns show a similar pattern except the first digit is changed so instead of 45 we have 55, etc. .

The second last digit in the first four columns shows another pattern and is always 6 for the first four columns and for the next four columns this is changed to a 7 and that between these two sets of four columns this is the only difference.

There is also a row pattern in that the last digit runs across the rows from 1 to 5 then starts at 1 again and goes to 5 by the tenth row.

CONCLUSION

Seeing these patterns, the code also became patterns and easier for the programmer to verify their work.

The paper explained how showing beginner programmers patterns can aid the beginner programmer to avoid these problems. This pattern recognition approach also making complex programming problems much simpler. And this approach allows more ready verification and self checking as the programmer writes their program.

ACKNOWLEDGMENTS

I wish to thank my supervisor Frances Anderson for help reviewing and organizing my paper.

RECOMMENDED READING

- *The Little SAS® Book*

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Peter Timusk
Enterprise: Statistics Canada
Address: 170 Tunney's Pasture Driveway
Ottawa, Ontario K1A 0T6
Work Phone: (613) 951-2531
E-mail: peter.timusk@statcan.gc.ca