

Beyond IF THEN ELSE: Techniques for Conditional Execution of SAS® Code

Joshua M. Horstman, Nested Loop Consulting, Indianapolis, IN

ABSTRACT

Nearly every SAS® program includes logic that causes certain code to be executed only when specific conditions are met. This is commonly done using the IF...THEN...ELSE syntax. In this paper, we will explore various ways to construct conditional SAS logic, including some that may provide advantages over the IF statement. Topics will include the SELECT statement, the IFC and IFN functions, the CHOOSE and WHICH families of functions, as well as some more esoteric methods. We'll also make sure we understand the difference between a regular IF and the %IF macro statement.

INTRODUCTION

The conditional execution of code is one of the most fundamental concepts in computer programming of all types. The ability to take different actions based on different inputs is essential in order to complete complex tasks. Indeed, most SAS programs are chock full of conditional logic.

The most common method for implementing conditional logic in SAS, and probably the first learned by most SAS programmers, is the IF...THEN...ELSE statement. This construct provides a simple means of branching the flow of execution based on whether a specified condition is true or false.

However, there are many other techniques available to the SAS programmer. We will examine several of them in this paper, study examples of their use, and consider what advantages they may offer over the IF...THEN...ELSE statement.

THE SELECT STATEMENT

One alternative to the IF...THEN...ELSE statement is the SELECT statement. The SELECT statement is always used in conjunction with a group of one or more WHEN statements. This structure can operate in one of two ways.

The first and most common way to use the SELECT statement is to specify an expression as part of the SELECT statement. That expression is compared to the expression(s) associated with each WHEN statement. When the SELECT expression is equal to one of the WHEN expressions, an associated statement is executed. A WHEN statement can have more than one WHEN expression. An optional OTHERWISE statement can specify a statement to be executed only if none of the WHEN expressions match the SELECT expression.

EXAMPLE #1 USING IF...THEN...ELSE

Consider the following example written with traditional IF...THEN...ELSE syntax. This code computes a total price which varies depending on the type of customer. Everyone starts with the base price, but some customer types receive a discount. Additionally, certain customer types pay sales tax while others do not.

```
if customer_type = 'STANDARD' then total = price * taxrate;
else if customer_type = 'PREFERRED' then total = price * discount * taxrate;
else if customer_type = 'TAXEXEMPT' then total = price;
else if customer_type = 'GOVERNMENT' then total = price * discount;
```

EXAMPLE #1 USING THE SELECT STATEMENT

Now, observe how the exact same logic can be implemented using the SELECT statement:

```

select (customer_type);
  when ('STANDARD')    total = price * taxrate;
  when ('PREFERRED')  total = price * discount * taxrate;
  when ('TAXEXEMPT')  total = price;
  when ('GOVERNMENT') total = price * discount;
end;

```

While these two methods produce the same result, the structure of the SELECT statement is more readily apparent at a glance, making it easier to read, debug, modify, and reuse.

THE IFC AND IFN FUNCTIONS

Another way to implement conditional logic is through the use of the IFC and IFN functions, which were new in SAS version 9. In certain situations, these functions can produce a much more compact and elegant alternative to IF...THEN...ELSE logic.

Both functions accept the same four arguments: a logical expression to be tested, a value to be returned if the expression is true, a value to be returned if the expression is false, and, optionally, a value to be returned if the expression is missing. The difference between the two functions is that IFC returns a character value while IFN returns a numeric value.

EXAMPLE #1 USING THE IFN FUNCTION

Consider the following alternative way to code the first example shown above.

```

total = price *
  ifn(customer_type in ('PREFERRED','GOVERNMENT'),discount,1) *
  ifn(customer_type in ('STANDARD','PREFERRED'),taxrate,1);

```

This single assignment statement replaces an entire block of IF...THEN...ELSE logic, while simultaneously making the underlying logic behind the derivation of TOTAL more clear. In each call to the IFN function, if the expression is false, we have cleverly returned a value of 1, which will have no effect when used as a factor in multiplication.

EXAMPLE #2 USING THE IFC FUNCTION

The IFC function can be particularly useful when building strings consisting of multiple components determined by various pieces of conditional logic. Consider the example below. We wish to construct a text string for a report which will indicate a subject's marital status and number of children.

First look at the code written using a series of IF...THEN statements. Notice that a separate statement is included for each of several different cases that may occur.

```

if married='Y' and num_kids=0 then family_status = 'Married, no children';
if married='N' and num_kids=0 then family_status = 'Unmarried, no children';
if married='Y' and num_kids=1 then family_status = 'Married, 1 child';
if married='N' and num_kids=1 then family_status = 'Unmarried, 1 child';
if married='Y' and num_kids>1 then family_status = 'Married, '||
  strip(put(num_kids,best.))|| ' children';
if married='N' and num_kids>1 then family_status = 'Unmarried, '||
  strip(put(num_kids,best.))|| ' children';

```

Next, observe how the IFC function can be used as a convenient way to simplify the logic and make it easier to change or reuse in the future. The desired string is obtained in a single statement, and the code makes it easy to see that the string is simply a concatenation of three pieces, each of which is the result of some conditional logic.

```

family_status = catx(' ', ifc(married='Y','Married','Unmarried'),
  ifc(num_kids=0,'no',put(num_kids,best.)),
  ifc(num_kids=1,'child','children'));

```

A WORD OF CAUTION ABOUT IFC AND IFN

Although the IFC and IFN functions can provide tremendous versatility, there is one drawback that must be considered. Each of the arguments passed to IFC or IFN are evaluated before the logical test is performed. This can cause problems when one of the arguments is invalid under certain conditions, even when the logic is written to prevent that argument from being returned under those same conditions.

As an illustration, consider the following portion of a DATA step which includes an IF...THEN...ELSE statement:

```
x=0;
if x ne 0 then y=10/x;
else y=999;
```

Note that this code was designed to avoid the possibility of division by zero. We can write this code using the IFN function as follows:

```
x=0;
y = ifn(x ne 0, 10/x, 999);
```

Unfortunately, regardless of the value of x, the IFN function will first evaluate 10/x, which will result in division by zero when x=0. Although the value assigned to y will still be correct, we will see the following note in our SAS log:

```
NOTE: Division by zero detected at line 15 column 23.
x=0 y=999 _ERROR_=1 _N_=1
NOTE: Mathematical operations could not be performed at the following places.
      The results of the operations have been set to missing values.
```

In this specific example, we can sidestep this problem by simply using the DIVIDE function, which gracefully handles the problem by returning a missing value when division by zero is attempted.

```
x=0;
y = ifn(x ne 0, divide(10,x), 999);
```

In general, however, it is not always so easy to avoid these problems. As a result, there may be situations when use of the IFC and IFN functions is not appropriate.

PROC FORMAT

In certain situations, the FORMAT procedure can serve as a form of conditional logic. Suppose we have a dataset containing a variable PLANET_NAME which is populated with the names of planets. Further, suppose we wish to derive a new variable PLANET_ORDER which contains the corresponding ordinal position of each planet (e.g. Mercury = 1, Venus = 2, etc.). This can easily be accomplished using an IF...THEN...ELSE construction as follows:

```
if      upcase(planet_name) = 'MERCURY' then planet_order=1;
else if upcase(planet_name) = 'VENUS'   then planet_order=2;
else if upcase(planet_name) = 'EARTH'   then planet_order=3;
else if upcase(planet_name) = 'MARS'    then planet_order=4;
else if upcase(planet_name) = 'JUPITER' then planet_order=5;
else if upcase(planet_name) = 'SATURN'  then planet_order=6;
else if upcase(planet_name) = 'URANUS'  then planet_order=7;
else if upcase(planet_name) = 'NEPTUNE' then planet_order=8;
```

The same result can also be obtained by creating an informat using PROC FORMAT:

```

proc format;
  invalue planets
    'MERCURY' = 1
    'VENUS'   = 2
    'EARTH'   = 3
    'MARS'    = 4
    'JUPITER' = 5
    'SATURN'  = 6
    'URANUS'  = 7
    'NEPTUNE' = 8
  ;
run;

```

Once this informat is created, it must be applied in a DATA step to derive the new variable PLANET_ORDER based on the values of the existing variable PLANET_NAME:

```
planet_order = input(upcase(planet_name), planets.);
```

One significant advantage of this approach is that the informat can be used multiple times within a program, or even across multiple programs if stored in a library. In contrast, the IF...THEN...ELSE logic must be replicated in each and every place such a derivation is required. By using formats and informats to replace multiple copies of the same conditional logic, one can reduce the total amount of code. Additionally, the maintainability of the code is improved as changes to the logic now need be made in only one place.

THE WHICHC AND WHICHN FUNCTIONS

The WHICHC and WHICHN functions were new in version 9.2 of the SAS software. Both functions search for a value equal to the first argument from among the remaining arguments and return the index of the first matching value. WHICHC is used with character values while WHICHN is for numeric values, although both functions return a numeric value.

The code below demonstrates how we can harness this functionality to implement the same conditional logic used above to generate the PLANET_ORDER variable:

```
planet_order = whichc(planet_name,
  'MERCURY', 'VENUS', 'EARTH', 'MARS', 'JUPITER', 'SATURN', 'URANUS', 'NEPTUNE');
```

If the variable PLANET_NAME has the value "MERCURY", then the WHICHC function will return a 1 since "MERCURY" is the first argument in the list of arguments to be searched (that is, the list beginning with the second argument). Similarly, if PLANET_NAME has the value "NEPTUNE", the value returned by WHICHC will be 8 since "NEPTUNE" is the eighth argument, again not counting the first one.

Note that if the first argument to WHICHC or WHICHN is missing, then the function returns a missing value. If the first argument is not missing but fails to match any of the subsequent arguments, WHICHC or WHICHN will return zero. It's important that conditional logic be designed to accommodate this behavior.

THE CHOOSEC AND CHOOSEN FUNCTIONS

Another pair of functions which can be quite useful in implementing conditional logic are the CHOOSEC and CHOOSEN functions, which were new in SAS 9. These functions are essentially the inverse of the WHICHC and WHICHN functions. That is, rather than searching an argument list and returning the index of a matching value, they accept an index and return the corresponding argument from the list. As might be expected, CHOOSEC works with lists of character values, while CHOOSEN is for selecting from among numeric arguments.

By combining the CHOOSEC or CHOOSEN function with the WHICHC or WHICHN function, it is possible to construct a sort of in-line table lookup, which is a variety of conditional logic. As an example of this method, consider the following traditional IF...THEN...ELSE logic. Here we are deriving the variable RESPONSE by decoding the value of RESPONSE_CODE.

```

if response_code = 'PD' then response = 'Progressive Disease';
if response_code = 'PR' then response = 'Partial Response';
if response_code = 'CR' then response = 'Complete Response';
if response_code = 'SD' then response = 'Stable Disease';
if response_code = 'NE' then response = 'Not Evaluable';

```

There are many ways to streamline this logic, including some we have already discussed such as the SELECT statement and PROC FORMAT. It can also be done using the CHOOSEC and WHICHC functions in combination:

```

response = choosec(whichc(response_code, 'PD', 'PR', 'CR', 'SD', 'NE'),
  'Progressive Disease', 'Partial Response', 'Complete Response',
  'Stable Disease', 'Not Evaluable');

```

The call to WHICHC operates much like the example above relating to planetary order. WHICHC returns a number, in this case from 1 to 5, depending on the value of RESPONSE_CODE. This number is the first argument to CHOOSEC and serves to select the correct corresponding value that should be assigned to REPSONSE.

THE COALESCE AND COALESCEC FUNCTIONS

The COALESCE and COALESCEC functions can be used to implement conditional logic when the purpose of the logic is to select the first non-missing value among several expressions. That is, in fact, precisely what these functions do. Note that COALESCE is used with numeric values while COALESCEC operates on character data.

Consider the following example. We wish to derive the last date of contact (LSTCONDТ) for each subject in a clinical trial. If the subject has died, then the date of death (DTHDT) is used. If DTHDT is missing, then the subject has not died. If the subject is alive but has withdrawn from the study, the date of withdrawal (WDDT) will be used. If WDDT is missing, then the subject has not withdrawn. If the subject is still on study, then the date of the last dose (LSTDOSDT) is used. If LSTDOSDT is missing, then the subject has not been dosed and we will instead use the date of the last visit (LSTVISDT).

The logic described above can be implemented using the following IF...THEN...ELSE syntax.

```

if not missing(dthdt) then lstcondt = dthdt;
else if not missing(wddt) then lstcondt = wddt;
else if not missing(lstvisdt) then lstcondt = lstdosdt;
else lstcondt = lstvisdt;

```

Observe how the COALESCE function allows us to condense the above logic into a single, succinct statement.

```

lstcondt = coalesce(dthdt, wddt, lstdosdt, lstvisdt);

```

COMPARISON OPERATORS

In some situations, it is possible to use a comparison operator to perform conditional logic. For instance, suppose we wish to derive the study day, AESTDY, corresponding to a particular adverse event record which began on AESTDT. In this example, the date of the first dose, EXSTDТ, is defined as Day 1 with subsequent days incremented sequentially, while the day immediately prior to the first dose is defined as Day -1. Since there is no Day 0, simply subtracting the dates is insufficient. We resort instead to the following IF...THEN...ELSE code:

```

if aestdt >= exstdt then aestdy = aestdt - exstdt + 1;
else aestdy = aestdt - exstdt;

```

We can simplify this somewhat using the IFN function as shown below. Recall that the IFN function will return the second argument (1 in this case) when the first argument evaluates to true, and will otherwise return the third argument (0 here).

```

aestdy = aestdt - exstdt + ifn(aestdt >= exstdt, 1, 0);

```

As an alternative, we can obtain the same result in a single statement by using a comparison operator within the expression on the right-hand side of an assignment statement.

```
aestdy = aestdt - exstdt + (aestdt >= exstdt);
```

When the expression (AESTDT >= EXSTD) is processed, it will evaluate to 1 if true, or 0 if false. This has the convenient effect of adding 1 to the study day when AESTDT is equal to or later than EXSTD, which is precisely what we desire.

THE SUBSETTING IF

Sometimes a SAS programmer will need to use conditional logic to exclude or subset records. In such cases, the subsetting IF statement can be useful. The syntax of the subsetting IF statement different from that of the ordinary IF...THEN...ELSE statement in that there is neither a THEN or an ELSE. Rather, the statement consists solely of an IF statement followed by an expression to be evaluated. If the expression evaluates to true, data step processing continues and the current record is written to the output data set. If not, the current iteration of the DATA step terminates without writing any output and control returns to the top of the DATA step.

The subsetting IF statement is functionally equivalent to an IF...THEN statement written in the following manner.

```
if <expression> then
  do;
    <all subsequent DATA step statements>
  output;
end;
```

THE %IF MACRO STATEMENT

One thing that is frequently confusing to new SAS programmers is the difference between the IF...THEN...ELSE statement and the %IF...%THEN...%ELSE macro statement. While these two are similar in their logical construction, they operate at entirely separate levels within a SAS program. The IF...THEN...ELSE statement is a DATA step statement and can only be used therein. The %IF...%THEN...%END macro statements are actually part of the SAS/Macro facility, which is a separate language with its own syntax that can be used to generate SAS code.

The %IF macro statement allows you to conditionally generate SAS code, usually based on the value of one or more macro variables. The SAS code generated might be DATA step code, SAS procedures, or portions or combinations thereof. It could even be conditional code like that described previously.

CONCLUSION

There are many ways to implement conditional logic with SAS. One method is not necessarily better than another. Programmers should strive to write code that is simple, intuitive, and easy to modify, extend, or reuse. Sometimes it will not be possible to satisfy all of these criteria, so professional judgment must be exercised. Savvy SAS programmers will still use the IF...THEN...ELSE statement frequently, perhaps even primarily, but they also benefit from having a variety of tools available and knowing how and when to use them.

CONTACT INFORMATION

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

Joshua M. Horstman
Nested Loop Consulting
317-721-1009
josh@nestedloopconsulting.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.