IFC and IFN Functions: Alternatives to Simple DATA Step IF-THEN-ELSE, SELECT-END Code and PROC SQL CASE Statements

Thomas E. Billings, Union Bank, San Francisco, California

Abstract

IFC and IFN are Base SAS® functions whose result depends on whether a user-supplied logical expression is true, false, or missing. These functions support single statements that can perform the same processing as more complex IF-THEN-ELSE, SELECT-END code blocks in DATA step code, or CASE statements in native SAS PROC SQL. The use of IFC and IFN are explored, and equivalent alternate DATA step and PROC SQL statement code are illustrated. The question of when/if a logical expression may have a missing value is also investigated.

Introduction

IFN and IFC are Base SAS functions that should be in the repertoire of SAS programmers at the intermediate and advanced levels. They provide elegant alternatives to simple IF-THEN-ELSE, SELECT-END code blocks in DATA step programming, and can be used in place of simple CASE statements in native SAS SQL. The latter is significant, as SQL CASE statement syntax is relatively clumsy.

The IFC and IFN functions are not ANSI standard SQL functions and –for the most part - are not supported by relational databases at this time. That means they should not be used in PROC SQL code that is directly passed-through to an external database (i.e., using “Connect to” syntax). However, these functions are available for use in native SAS SQL applications, and they can make your programming easier. The functions are available in the expression builder GUIs found in SAS Enterprise Guide, SAS Data Integration Studio/Server, and other SAS solutions.

Let’s begin our exploration of these functions with the statement syntax, summarized from Base SAS documentation.

Function statement syntax (summary)

IFC and IFN have parallel syntax; IFC is used for illustration. IFC has 4 operands:

```
IFC( logical-expression,              /* operand #1 */
     value-returned-when-true,        /* operand #2 */
     value-returned-when-false,       /* operand #3 */
     value-returned-when-missing).    /* operand #4 */
```

The difference between IFC and IFN is that the values returned from IFC are character, while they are numeric for IFN. If you use the IFC function to assign a value to a character variable whose length has not yet been declared, the variable length will default to 200.

Operand #1 is a logical (or numeric) expression, which is evaluated to determine if it is true, false, or missing. The next three operands are the values the function returns based on evaluation of the first operand. Operands #2-4 are expressions that yield a numeric result (IFN) or character result (IFC). Here expression is used in the generic sense: a constant, a variable, or a statement with operations on variables and/or constants.

The next section defines some simple data sets and uses those to test the behavior of the functions.

How the functions behave with real data

This is a two-variable data set that will be used for tests of logical expressions in the functions. A format is also created as the results of IFN are numeric, and we want to display those results as true, false, or missing.

```
options nocenter;

* file raw_1: raw data for logical tests: (x=y), (fuzz(x-y));
data raw_1;
       input x y;
       format x y best9.;
datalines;
0 0
0 1
0 .
. .
. .a
1.0e-13 0
-1.0e-13 0
;
run;

* format to label numeric results of ifn function;
proc format;
       value ifn_label
              . = Missing
              0 = False
              1 = True;
run;
```

Next is a simple test: apply the IFN, IFC functions to the logical expression (x=y), and also test IFC using the numeric expression fuzz(x-y) as the first argument. Note that the result from IFN is reformatted to provide output that is easier to understand.

```
data test_1;
       set raw_1;
       length ifc_result ifn_result_fmt fuzz_result $8;
       ifc_result = ifc(x=y,"True","False","Missing");
       ifn_result = ifn(x=y,1,0,.);
       ifn_result_fmt = put(ifn_result,ifn_label.);
       fuzz_result = ifc(fuzz(x-y),"True","False","Missing");
       drop ifn_result;
run;

proc print data=test_1;
       title "Examples of ifc, ifn functions on operands (x=y) and fuzz(x-y)";
run;
```

which yields the results:

Examples of IFC, IFN functions on operands (x=y) and fuzz(x-y)

Obs	x	y	ifc_result	ifn_result_fmt	fuzz_result
1	0	0	True	True	False
2	0	1	False	False	True
3	0	.	False	False	Missing
4	.	.	True	True	Missing
5	.	A	False	False	Missing
6	1E-13	0	False	False	False
7	-1E-13	0	False	False	False

The results are as expected. Note that (x = y) does not resolve to missing, even if one or both of x, y are missing. Contrast that with the results for fuzz(x-y), a numeric expression that evaluates to zero when x is ±1e-13, and a numeric value of zero indicates False. If instead we evaluated an expression like (1 + fuzz(x-y)), then observations 6 & 7 above would yield True.

Remark: (x=y) is true in SAS when both x and y are the same type of missing, but (x=y) is usually evaluated as false in relational databases when one or both of x, y are null.

Now let's create another data set to examine how unary operands (e.g., z, fuzz(z)) are evaluated.

```
* file raw_2: raw data for unary logical tests: (z), (fuzz(z));
data raw_2;
       input z;
       format z best9.;
datalines;
2
1
0.01
0
-1
.
.A
.M
.Z
._
1.0e-13
-1.0e-13
;
run;
```

Apply the IFC, IFN functions to unary operands, z and fuzz(z).

```
data test_2;
       set raw_2;
       length ifc_result ifn_result_fmt $8;
       ifc_result = ifc(z,"True","False","Missing");
       ifn_result = ifn(z,1,0,.);
       ifn_result_fmt = put(ifn_result,ifn_label.);
       fuzz_result = ifn(fuzz(z),1,0,.);
       format fuzz_result ifn_label.;
       drop ifn_result;
run;

proc print data=test_2;
       title "Examples of ifc, ifn functions on operands (z) and fuzz(z)";
run;
```

which yields the results:

Examples of IFC, IFN functions on operands (z) and fuzz(z)

Obs	z	ifc_result	ifn_result_fmt	fuzz_result
1	2	True	True	True
2	1	True	True	True
3	0.01	True	True	True
4	0	False	False	False
5	-1	True	True	True
6	.	Missing	Missing	Missing
7	A	Missing	Missing	Missing
8	M	Missing	Missing	Missing
9	Z	Missing	Missing	Missing
10	_	Missing	Missing	Missing
11	1E-13	True	True	False
12	-1E-13	True	True	False

For observations 11 and 12, the value of z is ± 1e-13, which is less than 1e-12 in absolute value, the epsilon used for FUZZ calculations. We observe that ±1e-13 are not zero hence the unary expression (z) is True, but fuzz(z)

evaluates to zero, which is False. Similar comments (per the first example data set) apply to the expression (1 + fuzz(z)).

The next two sections illustrate equivalent code in the DATA step and PROC SQL. There are many ways to do things in the SAS system. The code below illustrates one method; there are other, equivalent ways to do the same thing.

Equivalent DATA Step Code for logical test: (x=y)

This illustrates IF-THEN_ELSE and SELECT-END code that is analogous to the IFC, IFN function code.

```
* check if-then-else and select-end logic, file raw_1;
data test_31;
       set raw_1;
       length if_then_var $8;

       if (x=y) then
              if_then_var = "True";
       else if_then_var = "False";
       length select_var $8;
       select;
              when (x=y) select_var = "True";
              otherwise select_var = "False";
       end;
run;

proc print data=test_31;
       title "data step code equivalent to ifc, ifn functions on (x=y)";
run;
```

which yields the results:

Results from DATA step code equivalent to IFC, IFN functions on (x=y)

Obs	x	y	if_then_var	select_var
1	0	0	True	True
2	0	1	False	False
3	0	.	False	False
4	.	.	True	True
5	.	A	False	False
6	1E-130		False	False
7	-1E-130		False	False

Comparison of the results above with the preceding table for IFC, IFN on (x=y) shows that the results are identical.

Equivalent DATA Step Code for (z) as a unary logical expression

This code is more complex, to accommodate both system and special missing values.

```
* check if-then-else and select-end logic, file raw_2;
data test_32;
       set raw_2;
       length if_then_var $8;
       if_then_var = "Missing";

       if (z=0) then
              if_then_var = "False";
       else if ((z ne 0) and not missing(z)) then
              if_then_var = "True";
       length select_var $8;
       select;
              when (z=0) select_var = "False";
```

```
              when ((z ne 0) and not missing(z)) select_var = "True";
              otherwise select_var = "Missing";
       end;
run;

proc print data=test_32;
       title "data step code equivalent to ifc, ifn functions on (z)";
run;
```

which yields the results:

Results from DATA step code equivalent to IFC, IFN functions on (z)

Obs	z	if_then_var	select_var
1	2	True	True
2	1	True	True
3	0.01	True	True
4	0	False	False
5	-1	True	True
6	.	Missing	Missing
7	A	Missing	Missing
8	M	Missing	Missing
9	Z	Missing	Missing
10	_	Missing	Missing
11	1E-13	True	True
12	-1E-13	True	True

Comparison of the results above with the preceding table for IFC, IFN on (z) shows that the results are identical.

Equivalent PROC SQL CASE statement code for logical test: (x=y)

Here is one way to code the test of (x=y) in a CASE statement.

```
* check sql case logic, file raw_1;
proc sql;
       create table test_41 as
              select *,
                     case
                            when (x=y) then "True"
                            else "False"
                     end as sql_case_var length=8
              from raw_1;
quit;

proc print data=test_41;
       title "proc sql code equivalent to ifc, ifn functions on (x=y)";
run;
```

which yields the results:

Results from PROC SQL code equivalent to IFC, IFN functions on (x=y)

Obs	x	y	sql_case_var
1	0	0	True
2	0	1	False
3	0	.	False
4	.	.	True
5	.	A	False
6	1E-13	0	False
7	-1E-13	0	False

Comparison of the results above with the preceding table for IFC, IFN on (x=y) shows that the results are identical.

Equivalent PROC SQL CASE statement code for test of unary (z)

Here is one way to code the test of (z) in a CASE statement. The syntax is more complex in order to differentiate missing vs. non-missing.

```
* check sql case logic, file raw_2;
proc sql;
      create table test_42 as
            select *,
                  case
                        when (z=0) then "False"
                        when ((z ne 0) and not missing(z)) then "True"
                        else "Missing"
                  end as sql_case_var length=8
            from raw_2;
quit;

proc print data=test_42;
      title "proc sql code equivalent to ifc, ifn functions on (z)";
run;
```

which yields the results:

Results from PROC SQL code equivalent to IFC, IFN functions on (z)

Obs	z	sql_case_var
1	2	True
2	1	True
3	0.01	True
4	0	False
5	-1	True
6	.	Missing
7	A	Missing
8	M	Missing
9	Z	Missing
10	_	Missing
11	1E-13	True
12	-1E-13	True

Once again, comparison of the results above with the preceding table for IFC, IFN on (z) shows that the results are identical.

Additional information

Use in SAS GUI-based tools/solutions. IFC and IFN are supported in SAS Enterprise Guide, SAS Data Integration Studio, and many other SAS solutions. Under the appropriate circumstances, their use may help one to reduce the number of CASE statements in a project/job.

Use in WHERE clauses. IFC, IFN functions can be used in WHERE clauses to support complex row-selection logic.

When can a logical expression be missing? Consider the following situation. You have what you believe may be a "logical variable" (non-RETAIN), say "Z = (x=y)" in your program, and the logic to set Z is conditional. Now Z is actually a numeric variable, set to missing at the start of the DATA step, and conditionally evaluated later in DATA step. But if circumstances exist for which the conditional code is not executed, variable Z won't be evaluated and thus can be missing when the IFC, IFN function operates. This is an indirect way a logical expression can be effectively missing, but it really happens via a proxy numeric variable that can be missing. (*SAS does not support a logical variable type.* This example assigns – to a numeric variable - the result of evaluation of a logical expression.)

FUZZ and COMPFUZZ functions. If you want to use IFC or IFN to evaluate unary operands like (z), where (z) is derived by numeric computation, be aware of the precision of floating point calculation versus the exact (z=0), (z ^=0 and not missing(z)) comparisons done in IFC, IFN. Use the FUZZ or COMPFUZZ functions to round results when needed for use in IFC, IFN. (The COMPFUZZ function is supported in SAS 9.3; available but unsupported in 9.2.)

CHOOSEC and CHOOSEN functions. In some contexts, these functions can - with suitable coding - provide functionality similar to that of IFC, IFN.

Comparison with Oracle® DECODE function. The DECODE function can support functionality similar to that provided by IFC, IFN. Interestingly, null = null evaluates as the equivalent of true in the DECODE function.

All return values for IFN, IFC functions are evaluated by SAS. This happens even though two of the values will not be used. This means that use of IFC, IFN may yield the note that missing values were generated as a result of operations.

Function statement syntax (summary) - revisited

Our exploration of these functions began with a summary of the statement syntax. Here is a restatement of part of the summary, in more precise form, using IFC as model for both functions. IFC has 4 operands:

```
IFC( logical or numeric expression,
     value-returned-when: logical expression is true OR (numeric expression is not
          exactly zero AND numeric expression is not system or special missing
          value)
     value-returned-when: logical expression is false OR numeric expression is
           exactly zero,
     value-returned-when: logical expression is missing (usually does not happen)
           OR numeric expression is system or special missing value.
```

Epilogue

IFC and IFN are valuable additions to a SAS programmer's toolkit, whether one is working in Base SAS, SAS Enterprise Guide, or other SAS solutions/tools. Appropriate use of these functions can make your code more elegant and easier to maintain. If you have not used them yet, they are worth considering and trying.

References:

SAS Institute, Inc. *SAS(R) 9.2 Language Reference: Dictionary, Fourth Edition.* Online documentation:

- IFC Function: http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a002604570.htm
- IFN Function: http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a002604573.htm
- FUZZ Function: http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000245897.htm
- CHOOSEC Function: http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a002604617.htm
- CHOOSEN Function: http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a002604619.htm

SAS Institute, Inc. *SAS(R) 9.3 Functions and CALL Routines: Reference.* Online documentation:

- COMPFUZZ Function: http://support.sas.com/documentation/cdl/en/lefunctionsref/63354/HTML/default/viewer.htm#p0ifledavu3rv1n10gclzxbyy0ul.htm

Oracle, Inc. Oracle® Database SQL Reference, 10g Release 2 (10.2). Online documentation:

- DECODE: http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/functions040.htm#i1017437

Acknowledgements:

The author wishes to thank Jack Hamilton of the Division of Research, Kaiser Permanente, Oakland, California, for valuable suggestions and reviewing this paper. Any mistakes herein are the responsibility of the author.

Contact Information:

Thomas E. Billings
Union Bank
Basel II - Retail Credit BTMU
400 California St.; 9th floor
MC 1-001-09
San Francisco, CA 94104

Phone: 415-765-2567
Email: tebillings@yahoo.com