

Applying IFN and IFC Functions

Jinson J. Erinjeri, Emmes Corporation; Saritha Bathi, Celgene Corporation

ABSTRACT

The SAS® IFC and IFN functions are helpful in compact as well as in elegant coding. Both IFC and IFN functions can be employed in Base SAS and PROC SQL as an alternative to the traditional IF THEN ELSE, SELECT END and SELECT CASE statements. The paper presents the applications of IFN and IFC functions in various scenarios along with a peek at efficiency comparisons of some of the traditional approaches. Also, presented are pitfalls using these functions and the ways around the pit for a successful run.

INTRODUCTION

Programming most likely involves the use of conditional logic and SAS programming is no exception. The most widely used approach to perform conditional logic is the IF THEN ELSE statements. Other statements include SELECT WHEN and SELECT CASE statements. These statements have been in SAS for a long time and there are relatively newer elegant functions such as IFN and IFC which can execute the same logic as the traditional ones. These functions can also serve as alternatives to various other means of executing conditional logic such as FORMAT procedure, comparison operators and functions (WHICHC, WHICHN etc.) In this paper, we will introduce and apply IFN as well as IFC functions in various scenarios along with the associated caveats and work arounds.

IFN and IFC functions are more useful in DATA step expressions when it is not convenient or possible to use an IF/THEN/ELSE construct or a WHERE statement.

SYNTAX

In IFN, 'N' stands for Numeric, and in IFC, 'C' stands for Character. The syntax for both IFC and IFN functions are same, with IFC returning character values, and IFN numeric values respectively.

IFC (*logical-expression, value-returned-when-true, value-returned-when-false, < value-returned-when-missing>*)

Arguments for IFC

1. A logical expression <specifies a numeric constant, variable, or expression>
2. value returned when true < specifies a character constant, variable, or expression that is returned when the value of logical-expression is true>
3. value returned when false < specifies a character constant, variable, or expression that is returned when the value of logical-expression is false>
4. value returned when missing, <which is optional> < specifies a character constant, variable, or expression that is returned when the value of logical-expression is missing>

IFN (*logical-expression, value-returned-when-true, value-returned-when-false, < value-returned-when-missing>*)

Arguments for IFN

1. A logical expression <specifies a numeric constant, variable, or expression>
2. value returned when true < specifies a numeric constant, variable, or expression that is returned when the value of logical-expression is true>
3. value returned when false <specifies a numeric constant, variable, or expression that is returned when the value of logical-expression is false>
4. value returned when missing, <which is optional> < specifies a numeric constant, variable, or expression that is returned when the value of logical-expression is missing>

Based on the first argument - if the logical expression is true, then the value in the second argument is returned and if false, the value in the third argument is returned. The fourth argument is returned when the logical expression is missing and is optional.

Application 1: Basic Application

A basic application of the IFC/IFN function using a simple data set is presented in Table 1.

Code	Results																																																															
<pre>data sub_appl; set sub; length ifc_out \$8.; ifc_def=ifc(var1<1,"Yes","No","Missing"); ifc_out=ifc(var1<1,"Yes","No","Missing"); ifn_out=ifn(var1<1,1,0,99); keep id if: var1 ; run; proc contents data=sub_appl; run;</pre>	<table><tr><th>Id</th><th>var1</th><th>ifc_out</th><th>ifc_def</th><th>ifn_out</th></tr><tr><td>1001</td><td>0.9254</td><td>Yes</td><td>Yes</td><td>1</td></tr><tr><td>1007</td><td>0.5339</td><td>Yes</td><td>Yes</td><td>1</td></tr><tr><td>1020</td><td>.</td><td>Yes</td><td>Yes</td><td>1</td></tr><tr><td>1024</td><td>.</td><td>Yes</td><td>Yes</td><td>1</td></tr><tr><td>1027</td><td>14.146</td><td>No</td><td>No</td><td>0</td></tr><tr><td>1032</td><td>5.8289</td><td>No</td><td>No</td><td>0</td></tr></table> <table><tr><th colspan="4">Alphabetic List of Variables and Attributes</th></tr><tr><th>#</th><th>Variable</th><th>Type</th><th>Len</th></tr><tr><td>1</td><td>var1</td><td>Num</td><td>8</td></tr><tr><td>2</td><td>Id</td><td>Num</td><td>8</td></tr><tr><td>4</td><td>ifc_def</td><td>Char</td><td>200</td></tr><tr><td>3</td><td>ifc_out</td><td>Char</td><td>8</td></tr><tr><td>5</td><td>ifn_out</td><td>Num</td><td>8</td></tr></table>	Id	var1	ifc_out	ifc_def	ifn_out	1001	0.9254	Yes	Yes	1	1007	0.5339	Yes	Yes	1	1020	.	Yes	Yes	1	1024	.	Yes	Yes	1	1027	14.146	No	No	0	1032	5.8289	No	No	0	Alphabetic List of Variables and Attributes				#	Variable	Type	Len	1	var1	Num	8	2	Id	Num	8	4	ifc_def	Char	200	3	ifc_out	Char	8	5	ifn_out	Num	8
Id	var1	ifc_out	ifc_def	ifn_out																																																												
1001	0.9254	Yes	Yes	1																																																												
1007	0.5339	Yes	Yes	1																																																												
1020	.	Yes	Yes	1																																																												
1024	.	Yes	Yes	1																																																												
1027	14.146	No	No	0																																																												
1032	5.8289	No	No	0																																																												
Alphabetic List of Variables and Attributes																																																																
#	Variable	Type	Len																																																													
1	var1	Num	8																																																													
2	Id	Num	8																																																													
4	ifc_def	Char	200																																																													
3	ifc_out	Char	8																																																													
5	ifn_out	Num	8																																																													

Table 1. Basic Application with Undesired Assignment of Missing Values

Note that we have used the LENGTH statement to specify the length to 8 characters. The default length is \$200 character as shown in PROC CONTENTS output for variable ifc_def in Table 1. For id's 1020 and 1024 with var1=., the ifc_out and ifn_out shows values of 'Yes' and '1', which one would have thought to be missing. In SAS, a missing value is in fact less than 1 and the binary check in the logic expression turns out to be true, thereby satisfying the criteria of 'Yes' or '1'. To counter this pitfall, a multiplication factor of the variable itself can be introduced and this will set the result of the logical expression to be missing, thereby activating the fourth argument which will set the result to missing as shown in Table 2.

Code	Results																												
<pre>data sub_fix1; set sub; length ifc_out \$8.; ifc_out=ifc(var1*(var1<1), "Yes", "No", "Missing"); ifn_out=ifn(var1*(var1<1), 1, 0, 99); keep id if: var1 ; run;</pre>	<table><tr><th>Id</th><th>var1</th><th>ifc_out</th><th>ifn_out</th></tr><tr><td>1001</td><td>0.9254</td><td>Yes</td><td>1</td></tr><tr><td>1007</td><td>0.5339</td><td>Yes</td><td>1</td></tr><tr><td>1020</td><td>.</td><td>Missing</td><td>99</td></tr><tr><td>1024</td><td>.</td><td>Missing</td><td>99</td></tr><tr><td>1027</td><td>14.1465</td><td>No</td><td>0</td></tr><tr><td>1032</td><td>5.8289</td><td>No</td><td>0</td></tr></table>	Id	var1	ifc_out	ifn_out	1001	0.9254	Yes	1	1007	0.5339	Yes	1	1020	.	Missing	99	1024	.	Missing	99	1027	14.1465	No	0	1032	5.8289	No	0
Id	var1	ifc_out	ifn_out																										
1001	0.9254	Yes	1																										
1007	0.5339	Yes	1																										
1020	.	Missing	99																										
1024	.	Missing	99																										
1027	14.1465	No	0																										
1032	5.8289	No	0																										

Table 2. Basic Application with Desired Assignment of Missing Values

Table 2 shows the desired assignment of missing values to ids 1024 and 1027 as coded in the IFC (value=Missing) and IFN functions (value=99).

Application 2: Using IN and NOT IN in the Logic Expression

Code					
<pre> data sub_app2; set sub; length ifc_out1 ifc_out2 \$8.; ifc_out1=ifc(var1*(var1 in (1)),"Yes","No","Missing"); ifc_out2=ifc(var1*(var1 not in (1)),"Yes","No","Missing"); ifn_out1=ifn(var1*(var1 in (1)),1,0,99); ifn_out2=ifn(var1*(var1 not in (1)),1,0,99); keep id if: var1; run; </pre>					
Results					
Id	var1	ifc_out1	ifc_out2	ifn_out1	ifn_out2
1001	0.93	No	Yes	0	1
1007	0.53	No	Yes	0	1
1020	.	Missing	Missing	99	99
1024	.	Missing	Missing	99	99
1027	14.1	No	Yes	0	1
1032	5.83	No	Yes	0	1

Table 3. Using IN and NOT IN in the Logic Expression of IFC and IFN Functions.

Table 3. shows the usage of IN and NOT IN the logic expression of IFC and IFN functions with the desired results. Note that, if data integrity checks were appropriately applied to exclude missing values for the variable of interest, then there is no need to apply the multiplication factor (i.e. variable of interest). Depending upon the logical condition needed, one may or may not need the multiplication factor but the authors suggest to make sure that the case of missing's is verified.

Application 3: Passing Variables in the Arguments other than Logic Expression

In applications 1 and 2, the text values (Yes, No, Missing) were passed as values to second, third and fourth arguments. However, variables can also be passed into these arguments. This feature can be very helpful in compact coding. The application of passing variables in the arguments is presented in Table 4.

Code					
<pre> data sub_app3; set sub; ifn_out1=ifn((var1 >1),var2,var1); ifn_out2=ifn((var1 <1),var2,var1); ifn_out3=ifn((var1 <=.),var2,var1); keep id if: var;; run; </pre>					
Results					
Id	var1	var2	ifn_out1	ifn_out2	ifn_out3
1001	0.9254	1	0.92539	1	0.9254
1007	0.5339	1	0.53388	1	0.5339
1020	.	0	.	0	0
1024	.	0	.	0	0
1027	14.147	4	4	14.1465	14.1465
1032	5.8289	.	.	5.8289	5.8289

Table 4. Passing Variables in the Arguments other than Logic Expression

The ifn_out1, ifn_out2 and ifn_out3 in Table 4 are the values of var1 and var2 based on the condition specified in the logic expression. The only syntax difference is the absence of quotes while passing the variable names in the arguments as opposed to passing text entries.

Application 4: Passing Functions in the Arguments other than Logic Expression

Functions can also be passed in the arguments of IFN and IFC similar to how variables were passed in the previous application.

Code						
<pre>data sub_app4; set sub; length ifc_out: \$8.; ifn_out1=ifn(var1*(var1<1),max(var2,10),var1,.); ifn_out_max=ifn(var1*(var1<1),max(var2,var1),var1,.); ifn_out_min=ifn(var1*(var1<1),min(var2,var1),var1,.); ifn_out_mean=ifn(var1*(var1<1),mean(var2,var1),var1,.); keep id if: var;; run;</pre>						
Results						
Id	var1	var2	ifn_out1	ifn_out_max	ifn_out_min	ifn_out_mean
1001	0.9254	1	10	1	0.9254	0.9627
1007	0.5339	1	10	1	0.5339	0.7669
1020	.	0	.	.	.	.
1024	.	0	.	.	.	.
1027	14.147	4	14.1465	14.1465	14.1465	14.1465
1032	5.8289	.	5.8289	5.8289	5.8289	5.8289

Table 5. Passing Functions in the Arguments other than Logic Expression

Table 5 shows the usage of functions such as MAX, MIN and MEAN in the arguments other than the logic expression. This application comes handy with various data manipulations especially those involving imputations.

Application 5: Passing Functions in the Logic Expression

In application 4, functions such as MAX, MEAN and MIN were passed in the argument of IFN functions, the same can be passed in the logic expression as shown in Table 6.

Code						
<pre>data sub_app5; set sub; length ifc_out: \$8.; ifc_out_max=ifc(max(var2,var1)>1,'Yes','No',.); ifn_out_min=ifn(min(var2,var1)>1,1,0); ifn_out_mean=ifn(mean(var2,var1)>1,min(var2,var1),max(var2,var1,0)); /* if zero's are needed at missing then use*/ ifn_out_mean0=ifn(mean(var2,var1)>1,min(var2,var1,0),max(var2,var1,0)); keep id if: var;; run;</pre>						

Results						
id	var1	var2	ifc_out_max	ifn_out_min	ifn_out_mean	ifn_out_mean0
1001	0.9254	1	No	0	1	1
1007	0.5339	1	No	0	1	1
1020	.	0	No	0	0	0
1024	.	0	No	0	0	0
1027	14.1465	4	Yes	1	4	0
1032	5.8289	.	Yes	1	5.82888	0

Table 6. Passing Functions in the Logic Expression

Table 6 shows the conditional usage of MIN and MAX functions in the logic expression. Also, presented is the usage of these functions both in the logic expression as well as in the arguments. The variables ifc_out_mean and ifc_out_mean0 uses the MEAN function in the logic expression with MIN and MAX functions in the arguments.

Application 6: IFC and IFN in PROC SQL

The IFC and IFN functions can also be employed in PROC SQL as shown in Table 7. However, it is important to note that these functions are not ANSI standard SQL functions and hence cannot be used in codes with external database connections.

Code				
<pre>proc sql; create table sub_app6 as select *, ifc(<var1<1, 'Yes', 'No', 'Missing') as ifc_out_sql, ifn(<var1<1, 1, 0, 99) as ifn_out_sql from sub; quit;</pre>				
Results				
Id	var1	var2	ifc_out_sql	ifn_out_sql
1001	0.9254	1	Yes	1
1007	0.5339	1	Yes	1
1020	.	0	No	0
1024	.	0	No	0
1027	14.1465	4	No	0
1032	5.8289	.	No	0

Table 7. IFN and IFC in PROC SQL

Application 7: Miscellaneous

In all the prior applications, IFN and IFC were the main functions but these functions can also be used as arguments to other functions as presented in Table 8. In this case, the CATX functions uses IFC function as input to its arguments. This approach can be used for coding complicated logic in an elegant way.

Code				
<pre> data sub_app7; set sub; length ifc_out: \$20.; ifc_out=cats(' ',ifc(var3>25,'Positive','Negative'), ifc(var1>.,'Var1>.','Var1<=.'), ifc(var2>.,'Var2>.','Var2<=.')); keep id if: var;; run; </pre>				
Results				
Id	var1	var2	var3	ifc_out
1001	0.9254	1	100	Positive,Var1>.,Var2>.
1007	0.5339	1	50	Positive,Var1>.,Var2>.
1020	.	0	33.333	Positive,Var1<=.,Var2>.
1024	.	0	25	Negative,Var1<=.,Var2>.
1027	14.1465	4	20	Negative,Var1>.,Var2>.
1032	5.8289	.	16.667	Negative,Var1>.,Var2<=.

Table 8. Miscellaneous Application (E.g. IFC as Input to a Function)

EFFICIENCY COMPARISON

The efficiency comparison for the IFC and IFN functions were compared with some of the traditional methods, mainly with regards to the User CPU Time. The User CPU Time is the time spent by the processor to execute user-written code which can be determined using the FULLSTIMER option in SAS. IFC/IFN were compared with 3 traditional methods - IF-THEN-ELSE, SELECT-WHEN and SQL-CASE.

The comparative runs were performed on a personal workstation with Intel®Core™ i7-4770 with 3.4GHz processor, 8GB installed memory and 64-bit Windows 7 operating system. The number of variables were kept constant at 835 (included both character and numeric variables) and user CPU time was recorded for varying number of observations. The runs were performed on two types of data sets, one with variable of interest, missing's included and the other with missing's excluded. Figure 1 shows the comparison of user CPU time with variable of interest (missing's included) with varying number of records for IFN and IFC functions.

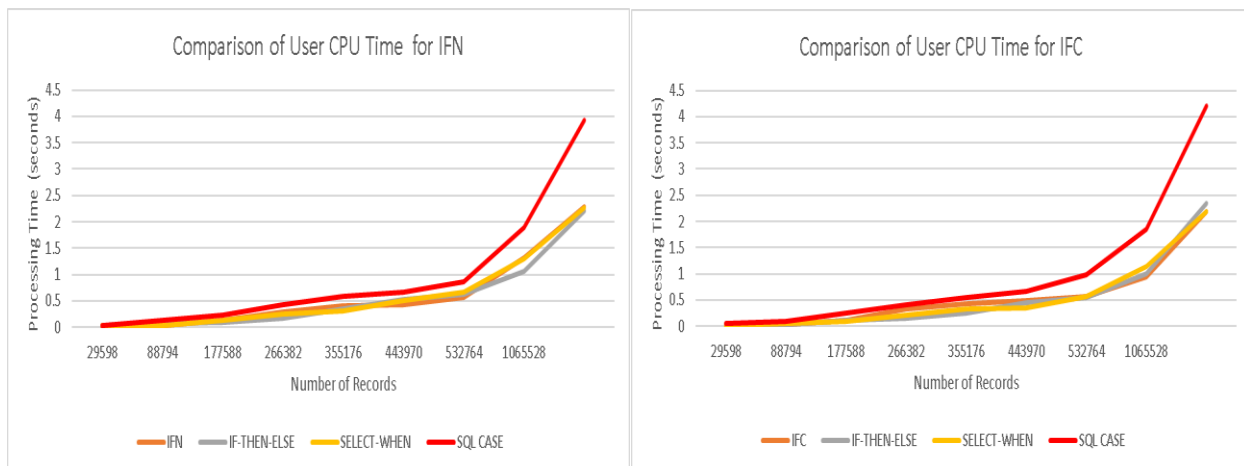


Figure 1. Comparison of User CPU Time between IFN/IFC and Traditional Approaches (Missing Values Included)

From Figure 1, for both IFN and IFC functions (missing values included), the user CPU time tends to be at par or better than some of the traditional approaches. Figure 2 shows the same trend for the user CPU time for the IFN/IFC functions (missing values excluded). With respect to memory allocated for the various conditional approaches, IFN/IFC had the lowest usage (~2095K). The comparisons of user CPU time and memory usage with the traditional approaches suggests that a programmer can use the IFN/IFC function as an additional tool for elegant coding without compromising on efficiency.

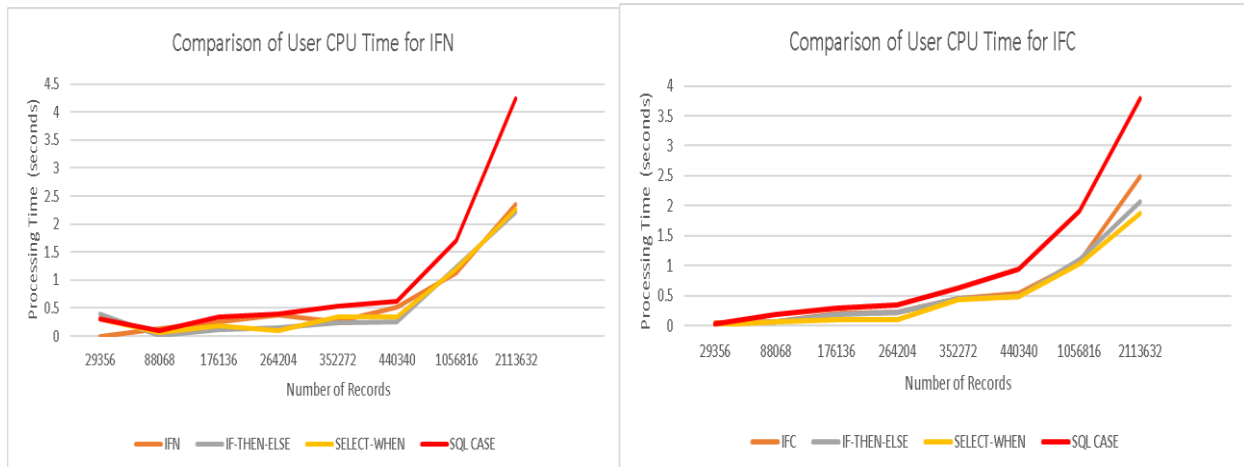


Figure 2. Comparison of User CPU Time between IFN/IFC and Traditional Approaches (Missing Values Excluded)

CONCLUSION

The paper presented various applications of IFN and IFC functions which can aid in elegant and compact coding. User CPU Time and memory comparisons to some of the traditional approaches also suggests that these functions can be applied without compromising much on efficiency.

REFERENCES

- SAS Online Documents. Available at <http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/de-fault/viewer.htm#a002604570.htm>
- SAS Online Documents. Available at <https://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/de-fault/viewer.htm#a002604573.htm>
- SAS Online Documents. Available at <http://support.sas.com/rnd/scalability/tools/fullstim/>

CONTACT INFORMATION

Your comments/questions/criticisms are valued and encouraged. Please contact the authors at:

Jinson J. Erinjeri
The Emmes Corporation
401 N Washington St.
Rockville, MD 20850
E-mail: jerinjeri@emmes.com

Saritha Bathi
Celgene, Corporation.
86 Morris Avenue
Summit, NJ 07901
E-mail: Sabathi@celgene.com