# Simplifying Complex Character Comparisons by Using the IN Operator and the Colon (:) Operator Modifier

**Paul Grant, Private Healthcare Systems, Inc.**

## Abstract

Complex character comparisons - comparisons of character values in a data set against a number of character constants which vary in length - are easier specified than coded. For example, how would you select the records of customers whose last names begin with 'Mc' or 'Mac'? How would you select the records of customers who live in ZIP codes beginning with ‘010’ through ‘0131’? This paper will show you how to use the IN operator and the colon (**:**) operator modifier to code these comparisons simply and clearly.

## A Sample Comparison

To illustrate this discussion, let’s use a more realistic version of the ZIP code comparison. Suppose we have to select the records for all the customers located in ZIP codes which begin with:

```
‘010’
‘011’
‘012’
‘0130’
‘0131’
‘0133’
‘0134’
‘0138’
```

The difficulty is that we want to select some records based on the first *three* bytes of the ZIP code, others based on the first *four* bytes.

## Some Cluttered Solutions

One frequently used way of selecting the correct records is to use the SUBSTR (Substring) function to truncate the values of the variable:

```
Data Select ;

   set Customer ;

      if substr( ZIP_code, 1, 3 ) = ‘010’ or
         substr( ZIP_code, 1, 3 ) = ‘011’ or
         substr( ZIP_code, 1, 3 ) = ‘012’ or
         substr( ZIP_code, 1, 4 ) = ‘0130’ or
         substr( ZIP_code, 1, 4 ) = ‘0131’ or
         substr( ZIP_code, 1, 4 ) = ‘0133’ or
         substr( ZIP_code, 1, 4 ) = ‘0134’ or
         substr( ZIP_code, 1, 4 ) = ‘0138’ ;
```

This approach certainly does the job but the code is verbose and cluttered (and probably inefficient as well). This condition only worsens as the complexity of the comparison increases.

We could improve this code by executing the SUBSTR function calls before the comparison statement and assigning the results to ‘throw-away’ variables. We could also replace the ‘=’ operator with the IN operator:

```
Data Select ;

   set Customer ;

      zip_3 = substr( ZIP_code, 1, 3 ) ;
      zip_4 = substr( ZIP_code, 1, 4 ) ;

      if Zip_3 in( ‘010’,
                   ‘011’,
                   ‘012’ ) or
        Zip_4 in ( ‘0130’,
                   ‘0131’,
                   ‘0133’,
                   ‘0134,
                   ‘0138’ ) ;

      drop Zip_3 Zip_4 ;
```

Taking this a step further, we could replace the SUBSTR functions with a LENGTH statement (a non-executable statement) and simple assignment statements:

```
Data Select ;

   set Customer ;

      length  Zip_3  $ 3
              Zip_4  $ 4 ;

      zip_3 = ZIP_code ;
      zip_4 = ZIP_code ;

      if Zip_3 in( ‘010’,   ….code continues…
```

The LENGTH statement creates two variables, 3 and 4 bytes long respectively. The assignment statements fills these variables with the first 3 and 4 bytes of ZIP_code. In other words, the SUBSTR function is redundant if you want to work with the *first* few bytes of a character value.

## Two Versatile Techniques

But there’s a much more elegant way to code this type of comparison. It employs two very versatile techniques available in the SAS® System. The first technique is to use the colon (**:**) operator modifier to truncate the length of the longer value. This eliminates the need for the SUBSTR function and/or creating additional variables.

In the SAS System, character values must be adjusted to the same length before they can be compared. When the SAS System compares character values *without* the colon (**:**) operator modifier, it pads the shorter value with blanks to the length of the longer value before making the comparison. When it compares character values *with* the colon (**:**) operator modifier, it truncates the longer value to the length of the shorter value. Thus:

‘010’ = ZIP_code *compares as* ‘010~~*bb*~~’ = ‘01034’

and

‘010’ =**:** ZIP_code *compares as* ‘010’ = ‘010’

The second technique is the undocumented ability of the IN operator to accept character constants of different lengths. For example, the statement:

```
if language in( ‘SAS’, ‘C’ ) ;
```

will be true for the values ‘SAS’ and ‘C’ but not for the values ‘COBOL’ or ‘PL/1.’ This capability is not mentioned in the SAS Language Reference Manual and all the examples in that manual have lists of character constants which have the same length.

## An Elegant Solution

Now, let’s apply these two techniques to the task at hand. In the following code, the IN operator contains a list of the various length ZIP_code prefixes we want to select and the colon (**:**) operator modifier truncates the values of ZIP_code to the appropriate lengths. Notice how similar this statement is to the simple list of ZIP code prefixes we used at the beginning of the paper.

```
Data Select ;

  set Customer ;

     if ZIP_code in:( ‘010’,
                      ‘011’,
                      ‘012’,
                      ‘0130’,
                      ‘0131’,
                      ‘0133’,
                      ‘0134’,
                      ‘0138’ ) ;
```

This simple and uncluttered code makes it easy for persons with limited knowledge of SAS to read the program and validate that it meets specifications. This is important!

These techniques are also quite versatile. For instance, since the first four ZIP_code prefixes we want to select form a continuous range, we can also code this IF statement as:

```
Data Select ;

  set Customer ;

     if ‘010’ <=: ZIP_code <=: ‘0131’    or
       ZIP_code in:(‘0133’,
                    ‘0134’,
                    ‘0138’ ) ;
```

The important thing to notice in this example is that the character constants used to specify the range are of different lengths. The first is 3 bytes long; the other, 4 bytes long.

## Conclusion

Using the IN operator with a list of character constants which vary in length and using the colon (**:**) operator modifier to truncate values being compared are two techniques which can greatly simplify the code you use to perform complex character comparisons. I’ve found them handy in dealing with ZIP codes, names, clinical coding schemes, and a variety of other classification values.

## References

SAS Institute Inc. (1990), *SAS Language: Reference, Version 6, First Edition*, Cary, NC: SAS Institute Inc., 121-125 and 517-519

## Your Thoughts

I’m interested in your comments and suggestions on the techniques described in this paper. You can contact me at:

Paul Grant
Private Healthcare Systems, Inc.
1100 Winter Street
Waltham, MA  02154
Tel:  781-895-7614
e-mail:  0005733569@mcimail.com