ABSTRACT

At times, values being compared by the SAS® System COMPARE procedure are truncated in the output (.lst) because they are too long and the differences are not shown. While the COMPARE procedure can produce various data sets, that approach can be cumbersome. Instead, the programmer might want to display the discrepant values in the log or redirect the log to a permanent file for a quick view. The goal of this paper is to introduce a SAS macro that quickly displays these overly long values in the log for quick comparison.

INTRODUCTION

Independent Validation usually involves a 100% match on the data set of interest, whether it is a CDISC (Clinical Data Interchange Standards Consortium) SDTM (Study Data Tabulation Model) domain, a CDISC ADaM (Analysis Data Model) data set, or a VALDS (Validation Data Sets from Tables, Figures, and Listings programs, also known as TFLs). For Validation programming using the SAS® System, the COMPARE procedure is nearly ubiquitous and one may not infrequently see the abbreviation DP-PC (Double Programming-Proc COMPARE) in SOPs (standard operating procedures) and WIs (work instructions).

Depending on the Validation Plan and applicable SOPs/WIs, the Validation programmer may need to exactly match:

- Variable names
- Variable attributes (type, label, and length)
- Variable Order within the data set (essential for SDTM domains)
- Variable Values

This paper pertains to matching of values in the Validation process. With an exact match of the Main and Validation data sets, the results, i.e. the COMPARE output in the .lst file or Output window of an interactive SAS session, are the minimal possible and easy to interpret. When discrepancies occur, investigating the origins of them can be complex and challenging, i.e., time consuming, especially if the Main and Validation data sets are not correctly sorted on the key variables. A further complication occurs when values are too long to be fully displayed in the COMPARE output, in which case they may appear to match exactly to the point of truncation. The goal of this paper is to introduce a SAS macro that quickly displays these overly long values fully in the log or a file for quick comparison, with flexibility in the format of the display, and the option to show a byte-by-byte comparison, including showing the hexadecimal (the SAS HEX. format) valued to emphasize discrepant non-printable bytes. The second goal of this paper is to introduce another SAS macro to display values with leading spaces, an issue that is notoriously hard to spot in COMPARE procedure output. The third goal of this paper is to discuss additions to values solely to create formatting, modifications such as leading spaces, versus using logic and styles to achieve selective formatting with the least complex value possible, i.e., to expedite Validation and minimize the data and comparisons required.

COMPARE RESULTS

The following data steps and COMPARE procedure generates the output in Display 1:
data one ;
  x   = 8 ;
  id1 = 1 ;
  id2 = "LONG INDENTIFICATION A" ;
run ;

data two ;
  set one ;
  x   = 9 ;
run ;

proc compare
  data = one
  comp = one
  listall
  ;
run ;

The desired scenario of an exact match is assured because the base data set and the comparison data set are the same (WORK.ONE).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Created</th>
<th>Modified</th>
<th>NVar</th>
<th>NObs</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK.ONE</td>
<td>05FEB23:14:46:00</td>
<td>05FEB23:14:46:00</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>WORK.ONE</td>
<td>05FEB23:14:46:00</td>
<td>05FEB23:14:46:00</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Number of Variables in Common: 3.
Number of ID Variables: 2.

Observation Summary

<table>
<thead>
<tr>
<th>First Obs</th>
<th>Base</th>
<th>Compare</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>id1=1 id2=&quot;LONG INDENTIFICATION A&quot;</td>
</tr>
<tr>
<td>Last Obs</td>
<td>1</td>
<td>1</td>
<td>id1=1 id2=&quot;LONG INDENTIFICATION A&quot;</td>
</tr>
</tbody>
</table>

Number of Observations in Common: 1.
Number of Observations Read from WORK.ONE: 1.

NOTE: No unequal values were found. All values compared are exactly equal.

Display 1. COMPARE procedure output for exact matches.
Note that values of the ID2 variables (lines 23 and 24) show the entire length: “LONG INDENTIFICATION A”. Contrast this with Display 2, specifically, line 60, which displays only the truncated value: “LONG INDENTIFICATION” for ID2.

```plaintext
The COMPARE Procedure
Comparison of WORK.ONE with WORK.TWO
(Method=EXACT)

Data Set Summary

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Created</th>
<th>Modified</th>
<th>NVar</th>
<th>NObs</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORK.ONE</td>
<td>05FEB23:14:46:00</td>
<td>05FEB23:14:46:00</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>WORK.TWO</td>
<td>05FEB23:14:46:00</td>
<td>05FEB23:14:46:00</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Variables Summary

Number of Variables in Common: 3.
Number of ID Variables: 2.

Observation Summary

First Obs 1 1 id1=1 id2=LONG INDENTIFICATION A
First Unequal 1 1 id1=1 id2=LONG INDENTIFICATION A
Last Obs 1 1 id1=1 id2=LONG INDENTIFICATION A

Number of Observations in Common: 1.
Total Number of Observations Read from WORK.ONE: 1.
Total Number of Observations Read from WORK.TWO: 1.
Number of Observations with Some Compared Variables Unequal: 1.
Number of Observations with All Compared Variables Equal: 0.

Values Comparison Summary

Number of Variables Compared with All Observations Equal: 0.
Number of Variables Compared with Some Observations Unequal: 1.
Maximum Difference: 1.

The COMPARE Procedure
Comparison of WORK.ONE with WORK.TWO
(Method=EXACT)

All Variables Compared have Unequal Values

Variable  Type  Len  Ndif  MaxDif
x    NUM     8     1 1.000

Value Comparison Results for Variables

<table>
<thead>
<tr>
<th>id1</th>
<th>id2</th>
<th>Base</th>
<th>Compare</th>
<th>Diff.</th>
<th>% Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>x</td>
<td>x</td>
<td>1.0000</td>
<td>12.5000</td>
</tr>
</tbody>
</table>

Display 2. COMPARE procedure with a mismatch and truncation of an ID variable value.
When one searches for the observations of interest, one might not find the observations, but instead received:

```
NOTE: No observations were selected from data set WORK.ONE.
NOTE: There were 0 observations read from the data set WORK.ONE.
WHERE (id1=1) and (id2='LONG INDENTIFICATION');
NOTE: PROCEDURE PRINT used (Total process time):
    real time         0.05 seconds
    cpu time          0.00 seconds
```

Depending on the data, one might not easily detect which (group of) observations to investigate. For instance, a possible DUT (Device Under Test), or UAT (User Acceptance Testing), program might include:

```latex
%let usub = "LONG INDENTIFICATION A1",
     "LONG INDENTIFICATION A2"
;
proc print
  data = adam.adlb
    ( keep = usubjid
      visitnum
      paramcd
      lbtestcd
      aval
      where = ( lbtestcd = "ALB"
                 and usubjid in
                 ( &usub. )
      )
    )
  ;
run;

class proc print
  data = v_adlb
    ( keep = usubjid
      visitnum
      paramcd
      lbtestcd
      aval
      where = ( lbtestcd = "ALB"
                 and usubjid in
                 ( &usub. )
      )
    )
  ;
run;

class proc print
  data = sdtm.lb
    ( keep = usubjid
      visitnum
      lbtestcd
      lbstresn
      where = ( lbtestcd = "ALB"
                 and usubjid in
                 ( &usub. )
      )
    )
  ;
run;
```
In this case, suspicion is immediate on ID2 since ID1 is numeric variable. We know both values match, so we might switch to the "in:" operator instead of "in", but that could result in voluminous output. Discerning the exact value and being able to copy and paste is preferrable.

**MAC_R_LONG_COMPARE RESULTS**

One could not copy the correct value of ID2 from the output in Display 2 for an observation of interest because the value is truncated, perhaps undetectably unless one is rather familiar with the data; which of the values is truncated might not be obvious if the ID variables include several “long” variables. The macro, MAC_R_LONG_COMPARE, overcomes this issue by rapidly displaying the values in either the log or directing the results to a file. The latter options does not pertain to most aspects of programming in clinical trials that follow CDISC guidance since the maximum length of variables is 200 bytes, which can be fully displayed in a SAS log. For issues concerning long DNA, RNA, or amino acid sequences, the reader may wish to consider BLAST®\(^1,2\) and the SAS macro previously presented\(^3,4\) noting that an updated version of MAC_U_BLAST is available upon request.

Appendix 1 presents MAC_R_LONG_COMPARE and invoking the HELP option of MAC_R_LONG_COMPARE results in a short description of the macro in the log:

```sas
%mac_r_long_compare
  ( help = Y ) ;
```

<table>
<thead>
<tr>
<th>Macro Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Name of the base data set.</td>
</tr>
<tr>
<td>comp</td>
<td>Name of the compare data set.</td>
</tr>
<tr>
<td>out</td>
<td>Name of the output data set.</td>
</tr>
<tr>
<td>var</td>
<td>Variable to compare.</td>
</tr>
<tr>
<td>id_rows</td>
<td>Whether to put one ID variable per row.</td>
</tr>
<tr>
<td>where</td>
<td>Where clause.</td>
</tr>
<tr>
<td>differences</td>
<td>Whether to use a bar/hyphen to emphasize differences.</td>
</tr>
<tr>
<td>transpose</td>
<td>The threshold of the length to transpose the values.</td>
</tr>
<tr>
<td>format</td>
<td>The format to display the values when not transposed.</td>
</tr>
<tr>
<td>byte_format</td>
<td>The format to display the individual bytes of the values when transposed.</td>
</tr>
<tr>
<td>both_formats</td>
<td>When transposed and a HEXw. format is used, whether to display both the values with a HEXw. format and with a CHAR1. format.</td>
</tr>
<tr>
<td>file</td>
<td>The path and filename to FILE (write) the results (discrepancies).</td>
</tr>
</tbody>
</table>

Display 3 presents two calls of MAC_R_LONG_COMPARE and their respective output for what should be the two most frequent uses of it. Note that the entire value of ID2 is displayed, i.e. no truncation, in contrast to the COMPARE results in Display 2 (line 60). Given that the most likely ID variables will be the keys of the SDTM domain or ADaM data sets, and thus could be five or more variables, some with long values, i.e., USUBJID, PARAM, VISIT, displaying each variable-value pair on its own row might be preferred (right panels of Display 3).
Another use of MAC_R_LONG_COMPARE is to elucidate values that appear to match in the output but one or both have one or more leading spaces. The presence of leading spaces may be obvious, but the number of leading spaces is not readily discernable. The following two data steps and COMPARE procedure produces the subset of the output in Display 4:

```plaintext
data three ;
   length x $ 30 ;
   x   = "    9 (10)" ;
   id1 = 1 ;
   id2 = "LONG INDENTIFICATION A" ;
   output ;
   x   = "   18 (20)" ;
   id1 = 2 ;
   id2 = "LONG INDENTIFICATION B" ;
   output ;
run ;

data four ;
   length x $ 30 ;
   x   = "    9 (10)" ;
   id1 = 1 ;
   id2 = "LONG INDENTIFICATION A" ;
   output ;
   x   = "   18 (20)" ;
   id1 = 2 ;
   id2 = "LONG INDENTIFICATION B" ;
   output ;
run ;
proc compare
   data = three
   comp = four
   listall ;
   id id1
   id2
run ;
```

Display 4. Partial COMPARE output demonstrating leading spaces mismatches.

<table>
<thead>
<tr>
<th>id1</th>
<th>id2</th>
<th>Base Value</th>
<th>Compare Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LONG INDENTIFICATION A</td>
<td>9 (10)</td>
<td>9 (10)</td>
</tr>
<tr>
<td>2</td>
<td>LONG INDENTIFICATION B</td>
<td>18 (20)</td>
<td>18 (20)</td>
</tr>
</tbody>
</table>

Display 3. Two typical uses of MAC_R_LONG_COMPARE.
Depending on where the mismatching observations occur in the list, one may not discern a leading space or two. With some experience, when one cannot see the mismatching characters, one suspects leading spaces, especially for VALDS’s in which a programmer may use leading spaces to format the output (PDF/RTF) or fail to use LEFT() or STRIP() when converting a numeric variable to its character version. One manual approach would be to open the data set in the SAS Explorer in an interactive session, find the observation(s) of interest, copy the value and paste it to an editor. One may need to manually increase the width of the variable in the SAS Explorer to see the entire value (and potential leading spaces); the next section discusses this in more detail. **Display 5** demonstrates a quicker approach using MAC_R_LONG_COMPARE.

<table>
<thead>
<tr>
<th>%mac_r_long_compare</th>
<th>%mac_r_long_compare</th>
</tr>
</thead>
<tbody>
<tr>
<td>( data = three</td>
<td>( data = three</td>
</tr>
<tr>
<td>, comp = four</td>
<td>, comp = four</td>
</tr>
<tr>
<td>, var = x</td>
<td>, var = x</td>
</tr>
<tr>
<td>, id = id1</td>
<td>, id = id1</td>
</tr>
<tr>
<td>id2</td>
<td>id2</td>
</tr>
<tr>
<td>) ;</td>
<td>) ;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id1=1 id2=LONG INDENTIFICATION A</th>
</tr>
</thead>
<tbody>
<tr>
<td>9 (10)</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>id1=2 id2=LONG INDENTIFICATION B</td>
</tr>
<tr>
<td>18 (20)</td>
</tr>
</tbody>
</table>

**Display 5. MAC_R_LONG_COMPARE output demonstrating leading spaces mismatches.**

The limitation of this approach is that one may still need to count the number of spaces by placing the cursor in column one and moving it using the right arrow key. That these values differ by one leading space is readily obvious from this output, however. Using the value of 0 (zero) for the macro parameter TRANSPOSE forces the display to be transposed, that is, to be displayed byte-by-byte across the rows of the log (or file). **Display 6** demonstrates this effect with the number of leading spaces being easily read from the row number.
```
%mac_r_long_compare
  ( data        = three
    , comp        = four
    , var         = x
    , id          = id1
    , id2         = id2
    , transpose   = 0
  );

%mac_r_long_compare
  ( data        = three
    , comp        = four
    , var         = x
    , id          = id1
    , id2         = id2
    , differences = Y
    , transpose   = 0
  );
```

<table>
<thead>
<tr>
<th>id1</th>
<th>id2</th>
<th>LONG IDENTIFICATION A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>9 -</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id1</th>
<th>id2</th>
<th>LONG IDENTIFICATION B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Display 6. MAC R_LONG_COMPARE output demonstrating leading spaces mismatches.

Another scourge of validation using the COMPARE procedure is the presence of non-printable characters. The following two data steps and COMPARE procedure produces the subset of the output in Display 7:

```
data five ;
  length x $ 4 ;
  x = cat( "ab"
    , "0A"x
    , "c"
  ) ;
  id1 = 1 ;
  id2 = "LONG IDENTIFICATION A" ;
run ;

data six ;
  x = "abc" ;
  id1 = 1 ;
  id2 = "LONG IDENTIFICATION A" ;
runc;

proc compare
  data = five
    comp = six
    listall
  ;
runc;
```
Variables with Unequal Values

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Len1</th>
<th>Len2</th>
<th>Ndif</th>
<th>MaxDif</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>CHAR</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Value Comparison Results for Variables

<table>
<thead>
<tr>
<th>Obs</th>
<th>Base Value</th>
<th>Compare Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>abc</td>
<td>abc</td>
</tr>
</tbody>
</table>

Display 7. Partial COMPARE output demonstrating a mismatch due to non-printable characters.

A keen eye might have noted the LENGTH statement in the data step that created data set FIVE and the difference between LEN1 and LEN2 in the COMPARE results, but that is likely an artifact of this contrived example; also including that LENGTH statement in the data step that created data set SIX would not affect this example. Further in independent Validation, one does not view the code or log of the peer program. While the Validation Plan and/or applicable SOP/WI might require a match on the lengths of variables, one can conclude that other values of X in data set FIVE could not have been length four because that would obviously create more mismatches since no value of X in data set SIX could have been length four, ignoring the fact that both data sets contain but one observation for this example. Those who have experienced this extreme consternation (and the pressure it creates with regards to deadlines), may have resorted to examining the hexadecimal representation of the values, and reasonably display them byte-by-byte using a DO-LOOP and SUBSTR(). This is the approach used to create the output in Display 8.

Display 8. A side-by-side comparison of a byte-by-byte display of the hexadecimal representation of each character.

From this output, it is easy to see that the values mismatches starting at byte 3. The mismatch is even more obvious in the bottom right panel of Display 8 because of the “deformity” caused by the spacing when the original values are also displayed. “0A”x is, again, “0 length”. The values in Display 7 both appear to be “abc”, but note that the Compare Value is offset one byte to the left of the variable x and the underline (both of the latter start under the “o” of Compare, but “a” of “abc” starts under the “C” of
“Compare”. Note that if one copies the value from the log or from the data set opened in the SAS Explorer and pastes it to the SAS Enhanced Editor, the “c” of “abc” will be on the next line:

```
ab
  c
```

which is expected since "0A"x is the line feed byte. One could (re)write the macro to COMPRESS(), for instance, the non-printable character using the third argument “KW” (Keep Written), to avoid the deformity, but the emphasis is appealing to the author and his eyes and one can copy the value origin byte from the log, which has some other utilities.

Another issue with the output in the bottom panels of Display 8 might catch a keen eye. The length of the variable X is 3 (three) bytes in data set SIX. Why does the sequence of bytes show that the fourth value is "20"x, a space (a byte, that does not exist in the value)? Would not displaying no value be a better representation? Indeed, but this is supposed to be a “quick and dirty” macro, a service macro. The discussion of the code below will elucidate the cause and provide a justification of the reason. Under no circumstances would this macro be used in Production (formal) execution or be part of a final Validation program, but it should be retained in a (dated) DUT/UAT program, especially for the astute Validation programmer who retains such programs as alternative or supplemental documentation to the Validation process beyond an issue tracker. Note that upon correction of the issue(s) that created a value with non-printable characters, Validation becomes more rapid, but the DUT/UAT program/code investigating mismatches can be “versioned” to document the original issue and it can be re-used for the next tasks.

**MAC_R_LEAD_SPACES**

Appendix 2 presents the macro, MAC_R_LEAD_SPACES, which displays any value with a leading space after replacing spaces with underscores and, optionally, removes those leading spaces with the STRIP() function. The following call demonstrates a macro using data set THREE above:

```
%mac_r_lead_spaces
  ( ds     = three ,
    remove = Y ,
    list   = N );
```

The SAS log shows that two values of the variable X have leading spaces:

```
x  9 (10)
x__18 (20)
```

For the purposes of investigation, one might wished to COMPARE the values of two data sets without leading spaces. In that case, one can use the macro parameter REMOVE= and OUT= as follows:

```
%mac_r_lead_spaces
  ( ds     = three ,
    out    = tres ,
    remove = Y ,
    list   = N );
```

```
proc compare
  data = three
   comp = tres
    listall
  ;
  id id1
    id2
  ;
run ;
```
<table>
<thead>
<tr>
<th>id1</th>
<th>id2</th>
<th>Base Value</th>
<th>Compare Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Long IDentification</td>
<td>1</td>
<td>9 (10)</td>
</tr>
<tr>
<td></td>
<td>Long IDentification</td>
<td>2</td>
<td>18 (20)</td>
</tr>
</tbody>
</table>

The Validation programmer certainly would not want to modify a permanent data set or file created by the Main programmer but using the macro to output a new (temporary) data set for investigational purposes in a non-production program could be informative. In this case, following whatever processes the Validation plan, SOPs, and WIs instruct, the Validation programmer could update the issue tracker and send an email to the Production programmer that informally, i.e., in a non-production (unofficial) run, a match occurs on the values without leading spaces and request either that the Production programmer report the algorithm that decides how many leading spaces a value should have or that the leading spaces be removed and to re-submit the Production program for another cycle of Validation.

**INCLUDING FORMATTING IN VALUES**

To increase the accuracy of Validation and to expedite the process, creating values of variables that are the minimal complexity with the maximum clarity is essential. The author presented a poster describing the structure (variable names, order, labels, types, and formats) of a standardized VALDS for Table and Listings programs. The code in Appendix 3 presents a data step and REPORT procedure to achieve indentation, for instance, the row labels, like “Female”, under section headers, like “Sex n(%)”, are typically indented. Using actual leading spaces can lead to problems and not only in Validation, but also in formatting of the output (PDF/RTF file), for instance, if the width of the column needs to be shorted or the value wraps due to the width of the column.

Display 9 presents the data set, INDENT, created by the data step in lines 1-64 of Appendix 3. Even though more involved than manually type leading spaces, if one may prefer to use the SAS function REPEAT(), on line 38, since one can explicitly see how many spaces were inserted. In this data set, one can see that the value of COLUMN_1 in observation 3 has leading spaces, if one expands the length of the column in the SAS Explorer (as the author did for Display 9).

Display 9. The data set INDENT created by the data step in Appendix 3.

The value of ROW_1 of observation 4 demonstrates the complexity of adding inline formatting to a value and that the length of the variable needs to be much greater than its intended display length. Note that the lengths of character variables in the proposed standardized VALDS are 200 by default, since this is the CDISC and Version 5 SAS Transport file (.xpt) maximum.

Display 10 present the PDF output of this code. Note that both the Style INDENT and leading spaces approach will not likely create acceptable wrapping of long text. The Style LEFTMARGIN, however, insures that any text that wraps is also properly indented. Both values subject to formatting by the REPORT procedure do not have leading spaces. Instead, the code in the COMPUTE block on lines 118-131 achieves the selective formatting and was demonstrated by Cynthia Zender on SAS Communities.
Display 10. The PDF output created by the REPORT procedure in Appendix 3.

**MAC_R_LONG_COMPARE CODE**

The macro is standard fare. Lines 99-141 create a SQL procedure that creates a data set containing observations with mismatching values in the variable of interest via an inner join of observations that match on the ID variables. If this data set contains observations, then the macro determines the type (char or num) of the variable from the SASHELP.VCOLUMN view (lines 147-161). If that variable is character, it then determines the maximum length of the values (lines 166-172). The remainder of the macro creates a data step to display the mismatching values for the given ID variables values.

Of note, instead of using column pointers, i.e. "@10" for each item, the columns are determined by the +n pointer. With slight effort, if desired, that can be changed so that the "deformity" discussed above can be rectified. Reading the output programmatically might be one reason to make the change.

The use of regular expressions is worth explaining, for example on line 221:

```s/**/prxchange( s/(_a-z)(_a-z0-9){0,31})/$1=/i , -1 , &id. )**```

The first argument is a substitution with the "/" delimiter: "s/**/". The pattern "(_a-z)(_a-z0-9){0,31}" is within a grouping (). If a string in the value matches that pattern, then that string is retained in memory. The value in the second part of the substitution is what is substituted. In this case $1 refer to the first (and only) grouping. So the result returned by that code is that string followed by an equal sign. The pattern is by bytes, "([a-z][a-z0-9]{0,31}" matches one byte that is either an underscore or a letter. In this case the "i" following "/**/" directs that match to be case insensitive, so "a" or "A". The "(_a-z0-9)" extends this match to include the characters 0-9 (character versions of digits). The "(0,31)" following that pattern instructs the regular expression engine to match 0 to 31 bytes following the first byte. Take in total, these are the rules for a SAS variable as invoked by the SAS System Option VALIDVARNAME = V7. Note that the need for quoting is obviated since it is macro code. In SAS code, this would be:

```prxchange( "s/(_a-z)(_a-z0-9){0,31})/$1=/i" , -1 , "&id."
```

Such formatting of one argument per line allows one to more easily see the arguments, but it is also very good for 1) debugging or experimenting since the arguments can be more easily commented out using a block comment (Ctrl-/ and Shift-Ctrl-/ and 2) written by a program (that will be subsequently assigned to a human programmer for fine tuning and approval). The second argument, "-1", instructs the regular expression engine to repeat the pattern substitution indefinitely.

Another viable way to write this is using the "#" delimiter: "s##". When the target or substitution text contains "/", this may be easier to code (and read):
data _null_;  
x = "1/2" ;  
y = prxchange( "s/// divided by /" , 1 , x ) ;  
z = prxchange( "s#/# divided by #" , 1 , x ) ;  
put x / y / z ;  
run ;  

1 divided by 2  
1 divided by 2  

NOTE: DATA statement used (Total process time):  
real time           0.20 seconds  
cpu time            0.01 seconds  

Note the need to escape the "/" when it is the delimiter of the substitution or match, i.e., "s///", but not when the delimiter is "#", i.e., "s###". Finally, on lines 222-226, the PRXCHANGE() is worth explaining. The third argument is also a to this PRXCHANGE() call is also a PRXCHANGE() call. The following code demonstrates:  

%let id = id1  
id2  
%put %sysfunc( prxchange( s/([_a-z][_a-z0-9]{0,31})/$/1=\//i , -1 , &id. )) ;  
id1=/  
id2=/  
%put %sysfunc( prxchange( s/\// , 1  

%sysfunc( prxchange( s/([_a-z][_a-z0-9]{0,31})/$/1=\//i , -1 , &id. ))  

id1=/  
id2=  

The inner PRXCHANGE() appends an “=/>” to each variable in the list of variables provided in the ID= macro parameter variable. The outer PRXCHANGE() removes the “/” if it is the last byte in the string, indicated by “$” in the pattern. Effectively, substitution removes the “/” from the last variable in that list and only from that variable.

CONCLUSION

Validation is a central, indispensable task in clinical trial programming that can be a bottleneck in the delivery process. The SAS COMPARE procedure is an essential tool in Validation when the SAS System is used. When values of the ID variables, variables being compared, or both are long, their values in the COMPARE output (.lst) may be truncated, complicating either the identification of the observation (in the case of ID variables) or the nature of the mismatch (in the case of the compared variable) and hampering subsequent investigations into the origins of the mismatch. In fact, despite being in the COMPARE output because the value of compared variable mismatched, the values may appear to match when the mismatch occurs "downstream" of the point of truncation.

This paper contributes to the accuracy and expeditiousness of Validation by presenting two macros to aid the programmers in identification of observations that mismatch on the variable of concern. MAC_R_LONG_COMPARE matches observations from two data sets on ID variables, but mismatch on the variable of interest and displays the full values of those variables rapidly, with minimal coding. The
The paper demonstrated how to use the macro when the long values of the variables were truncated in the COMPARE results; such results can either show the pattern or can be used for further investigations. The paper demonstrated mismatches due to varying numbers of leading spaces, often used to achieve desired formatting in the TFL output (PDF/RTF) files and showed how to use the macros to investigate and determine the number of leading spaces without manually counting. In contrast to a method such as

\[
\text{Length}(X) - \text{Length(Left}(X))
\]

the macro quantitates and visualizes the leading spaces. The paper demonstrated how to MAC_R_LONG_COMPARE if non-printable characters were “embedded”, but hard to see or identify, allowing the user to see the hexadecimal and original value in a byte-by-byte display.

The macro MAC_R_LEAD_SPACES displays values with leading spaces and the variables to which they pertain from a single data set. Seeing multiple variable/value pairs in one frame with a clearly defined starting point may assist the programmer in inferring a pattern, if any. Optionally, MAC_R_LEAD_SPACES can remove leading spaces so that an informal comparison of values without leading spaces can inform the programmers or guide further investigations if leading spaces are not the cause of mismatches. Usually, once the programmers know that the values without leading spaces match, arriving at the same algorithm to independently insert the same number of leading spaces occurs rapidly.

Finally, the paper contributes to an important discussion of whether values (in VALDS’s) should contain modifications to achieve formatting in TFL output or whether a more parsimonious approach would be to use programming logic in the REPORT procedure to achieve the desired formatting. The paper demonstrated indentation or changes to the “left margin”, but this could easily be adopted to other attributes or paging, for instance, changing font or background color or font emphasis (bold or italics, for instance). Substantially, some modifications may be distinct to a given programming language, package, or output destination, but “raw” values should be invariant, i.e., programming language-agnostic, for instance.

The macros and code in this paper are provided “as is”. The author and his employers assume no responsibility for their use, but do appreciate notification of errors, bugs, corrections, or suggestions.

REFERENCES

5 KRV Holder, PHUSE Connect 2023, Orlando, FL.
Appendix 1

```sas
%macro mac_r_long_compare
  ( data         = , comp         = , out          = _lc , var          = , id          = , id_rows      = N , where        = 1 , differences = N , transpose    = 256 , format       = %str() , byte_format  = $char1. , both_formats = Y , file         = %str() , help         = N ) ;

%if &help. = Y %then
  %do ;
  %put _________________________________________________________________________________________________ ;
  %put Purpose of program:        This macro displays values of mismatching variables that cannot be ;
  %put %str(displayed)          fully by the COMPARE procedure output due to truncation. ;
  %put _____________ ___________________________________________________________________________ ;
  %put data                      = Name of the base data set.                                                ;
  %put comp                      = Name of the compare data set.                                             ;
  %put out                       = Name of the output data set.                                              ;
  %put %str( Default: __lc )                                                      ;
  %put var                       = Variable to compare.                                                      ;
  %put id                        = ID variables.                                                             ;
  %put id_rows                   = Whether to put one ID variable per row.                                   ;
  %put %str( Default: N )                                                      ;
  %put where                     = Where clause.                                                             ;
  %put %str( Default: 1 )                                                      ;
  %put differences               = Whether to use a bar/hyphen to emphasize differences.                     ;
  %put %str( Default: N )                                                      ;
  %put transpose                 = The threshold of the length to transpose the values.                     ;
  %put %str( Default: 256 )                                                      ;
  %put format                    = The format to display the values when not transposed.                     ;
  %put %str( Default: %nrstr(%%)str() )                                                      ;
  %put byte_format               = The format to display the individual bytes of the values when transposed. ;
  %put %str( Default: $char1. )                                                      ;
  %put both_formats              = When transposed and a HEXw. format is used, whether to display both the ;
  %put %str( Default: Y )                                                      ;
  %put file                      = %str() ;
  %put %str( Default: Y )                                                      ;
  %put help                      = N ;
  %end ;
  %do ;
  %put Vput Purpose of program:        This macro displays values of mismatching variables that cannot be ;
  %put %str(displayed)          fully by the COMPARE procedure output due to truncation. ;
  %put _____________ ___________________________________________________________________________ ;
  %put data                      = Name of the base data set.                                                ;
  %put comp                      = Name of the compare data set.                                             ;
  %put out                       = Name of the output data set.                                              ;
  %put %str( Default: __lc )                                                      ;
  %put var                       = Variable to compare.                                                      ;
  %put id                        = ID variables.                                                             ;
  %put id_rows                   = Whether to put one ID variable per row.                                   ;
  %put %str( Default: N )                                                      ;
  %put where                     = Where clause.                                                             ;
  %put %str( Default: 1 )                                                      ;
  %put differences               = Whether to use a bar/hyphen to emphasize differences.                     ;
  %put %str( Default: N )                                                      ;
  %put transpose                 = The threshold of the length to transpose the values.                     ;
  %put %str( Default: 256 )                                                      ;
  %put format                    = The format to display the values when not transposed.                     ;
  %put %str( Default: %nrstr(%%)str() )                                                      ;
  %put byte_format               = The format to display the individual bytes of the values when transposed. ;
  %put %str( Default: $char1. )                                                      ;
  %put both_formats              = When transposed and a HEXw. format is used, whether to display both the ;
  %put %str( Default: Y )                                                      ;
  %put file                      = %str() ;
```
%put file = The path and filename to FILE (write) the results [discrepancies].;
%put %str( )Default: %nrstr(%%)str() ;
%put End of help;
%goto END ;
%end ;

proc sql;
create table &out. as
select a.&var. as v1
, b.&var. as v2
%let i = 1 ;
%let d = %scan( &id. , &i. , %str( )) ;
%do %while ( &d. ne ) ;
%let i = %eval( &i. + 1 ) ;
%let d = %scan( &id. , &i. , %str( )) ;
%end ;
from %data. as a
, %comp. as b
where %let i = 1 ;
%let d = %scan( &id. , &i. , %str( )) ;
%do %while ( &d. ne ) ;
%if &i. > 1 %then and ;
a.&d. = b.&d.
%let i = %eval( &i. + 1 ) ;
%let d = %scan( &id. , &i. , %str( )) ;
%end ;
a.&var. ne b.&var.
and %where.
order by %let i = 1 ;
%let d = %scan( &id. , &i. , %str( )) ;
%do %while ( &d. ne ) ;
%if &i. > 1 %then , ;
a.&d.
%let i = %eval( &i. + 1 ) ;
%let d = %scan( &id. , &i. , %str( )) ;
%end ;
;
quit ;
%if &sqlobs. > 0
  %then
    %do ;
    proc sql
    noprint ;
    select type
    into : type
    from sashelp.vcolumn
    where %if %sysfunc( index( &out. , . ))
      %then upcase( libname ) = "%sysfunc( upcase( %sysfunc( scan( &out. , 1 , . ))))"
      and upcase( memname ) = "%sysfunc( upcase( %sysfunc( scan( &out. , 2 , . ))))"
    ;
    %else
      upcase( libname ) = "WORK"
      and upcase( memname ) = "%sysfunc( upcase( %out. ))"
    ;
    and upcase( name ) = "V1"
    ;
    quit ;
%end ;
%if &type. = char
  %then
    %do ;
    proc sql
    noprint ;
    select strip( put( max( length( v1 )) max max( length( v2 )) , 8.0 ))
    into : length separated by ""
    from &out.
    ;
    quit ;
%end ;
%if &format. = %str()
  %then
    %let format = $char&length.. ;
%end ;
%else %let length = ;
data _null_;
%if %nrbquote(&file.) ne %str()
  %then
    file "&file." %str(;) ;
set &out. ;
if _n_ > 1 then put /;

%if differences. = Y
and stype. = char
%then
  do;
    length diff $ slength.
    byte $ 1;
    difff = " ";
    l1 = length( v1 )
    l2 = length( v2 )
    min = min( l1 , l2 )
    max = max( l1 , l2 )
    do _n_ = 1 to min;
      if substr( v1 , _n_ , 1 ) = substr( v2 , _n_ , 1 )
      then byte = " "
      else byte = "|"
      substr( diff , _n_ , 1 ) = byte;
    end;
    if l1 ne l2
    then substr( diff , min + 1 , max - min ) = repeat( "|" , max - min - 1 )
  %end;
%if differences. = Y
and stype. = char
%then
  put diff &format.;
%end;

%else
  do;
    length v1_byte
    %if differences. = Y
    and stype. = char
    %then
diff_byte = substr( diff , _n_ , 1 )
    if diff_byte = "|" then diff_byte = "-"
    %end;
    if _n_ <= length( v2 ) then v2_byte = substr( v2 , _n_ , 1 )
    else v2_byte = " ";
  end;
%end;

%else
  do;
    %if differences. = Y
    and stype. = char
    %then
      +1 diff_byte
    %end;
    +1
  end;
%end;
%if %sysfunc(prxmatch(/^hex/i, &byte_format._)) and &bothformats._ = Y
  %then v2_byte &byte_format._+2 v2_byte $char. ;
%else v2_byte &byte_format._ ;
end ;
%end ;
%end ; /* END OF sqlobs > 0 */
%/else %put A$\text{\texttt{ERT:}} No qualifying observations. ;
%/__END:
%mend mac_r_long_compare ;

Appendix 2

%macro mac_r_lead_spaces
  %do -
    , out = _null_
    , remove = N
    , list = Y
  %end ;
data &out. ;
  set &ds. ;
  array __c( * ) $ _character_ ;
  %do _n_ = 1 to dim( __c ) ;
    %if %sysfunc(prxmatch(/Y|YES/i, &list.)) > 0 %then
      %do ;
        if __c( _n_ ) ne strip( __c( _n_ ) )
          then do ;
            t = translate( __c( _n_ ) , "_" , " " ) ;
            v = vname( __c( _n_ ) ) ;
            put @1 v @35 t ;
          %if %sysfunc(prxmatch(/Y|YES/i, &remove.)) > 0 %then __c( _n_ ) = strip( __c( _n_ ) ) $ ;
        end ;
      %end ;
    %else %if %sysfunc(prxmatch(/Y|YES/i, &remove.)) > 0
      %then __c( _n_ ) = strip( __c( _n_ ) ) $ ;
    %end ;
  %end ;
%mend mac_r_lead_spaces ;
data indent ;
length row_1 $200
;  
/*********/
row_order_1 = 1;
row_1 = "Plain";
column_1 = " ";
do _n_ = 1 to 9;
column_1 = cats( column_1
  , "--------"
  , put( _n_ , 1. )
  )
;
end;
output;
/*********/
row_order_1 + 1;
row_1 = "5 Leading spaces in value"
/* The REPEAT function, though cumbersome, makes the number obvious:
   note that it creates one more replicate than the (non-negative
   integer) second argument.
*/
column_1 = cat( repeat( " ", 4 )
  , "--------"
  )
;
do _n_ = 2 to 9;
column_1 = cat( trim( column_1 )
  , "--------"
  , put( _n_ , 1. )
  )
;
end;
output;
/*********/
row_order_1 + 1;
row_1 = "{style [ textdecoration = underline Left Margin]:   `{style [ fontweight = medium color = black style = `{unicode 007B} leftmargin = 4% `{unicode 007D}}"
/* (style [ textdecoration = underline ]Left Margin):   `{style [ fontweight = medium color = black ]style = `{unicode 007B} leftmargin = 4% `{unicode 007D}}
*/
column_1 = cats( column_1
  , "--------"
  , put( _n_ , 1. )
  )
;
end;
output;
*********/
run;
ods escapechar = "`";
ods pdf
  file = "&sysfunc( getoption( SASUSER ))\trim( %scan( %sysget( sas_execfilename ) , 1 , . )).pdf"
ods listing close;
ods noresults;
title1 'Only row_1 = "5 Leading spaces in value" affects the value of ROW_1'
title2 'Left margin is preferable, if the text might wrap, since the wrapped text is "properly" aligned'
proc report
data = indent;
column row_order_1
row_1
column_1;
define row_order_1
/ order
noprint
/
```
93  define row_1
94   / order
95    style ( column ) = { cellwidth = 45% } ;
96  define column_1
97   / order
98    style ( column ) = { cellwidth = 50%
99      asis = on
100   }
101 ;
102  compute row_1 ;
103  if row_order_1 = 4
104     then call define ( _col_, "style"
105       , "style = { color          = green
106          , fontweight     = bold
107      }" ) ;
108 endcomp ;
109  compute column_1 ;
110  if row_order_1 = 2
111     then call define ( _col_, "style"
112       , "style = { indent = 4% }"
113     ) ;
114 else if row_order_1 = 4
115     then call define ( _col_, "style"
116       , "style = { leftmargin = 4% }"
117     ) ;
118 endcomp ;
119 run ;
120 ods pdf close ;
121 ods listing ;
122 ods results ;
```