

# Cody's Collection of Popular SAS® Programming Tasks and How to Tackle Them



Ron Cody



From *Cody's Collection of Popular SAS® Programming Tasks and How to Tackle Them*. Full book available for purchase [here](#).

## Contents

<b>List of Programs .....</b>	<b>ix</b>
<b>About This Book.....</b>	<b>xv</b>
<b>About The Author.....</b>	<b>xix</b>
<b>Acknowledgments .....</b>	<b>xxi</b>
<b>Chapter 1 Tasks Involving Conversion: Character to Numeric, Specific Values to Missing, and Changing Case .....</b>	<b>1</b>
Introduction.....	1
Task: Converting character values to numeric values.....	2
Task: Converting character values to numeric values using a macro .....	3
Task: Converting a specific value such as 999 to a missing value for all numeric variables in a SAS data set.....	5
Task: Converting a specific value such as 'NA' to a missing value for all character variables in a SAS data set .....	7
Task: Changing all character values to either uppercase, lowercase, or proper case.....	8
Task: Reading a numeric value that contains units such as Lbs. or Kgs. in the value .....	9
Task: Solving part of the previous task using a Perl regular expression.....	10
Conclusion .....	11
<b>Chapter 2 Grouping Data .....</b>	<b>13</b>
Introduction.....	13
Task: Grouping values using IF-THEN-ELSE statements.....	13
Task: Grouping values using user-defined formats .....	14

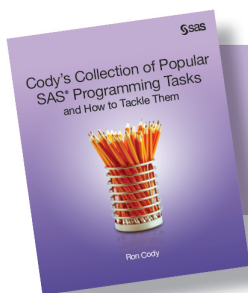
<b>Task: Creating groups using PROC RANK .....</b>	<b>15</b>
<b>Conclusion .....</b>	<b>19</b>
<b>Chapter 3 Summarizing Data .....</b>	<b>21</b>
<b>Introduction .....</b>	<b>21</b>
<b>Task: Using PROC MEANS to create a data set containing summary information .....</b>	<b>22</b>
<b>Task: Computing the mean of a variable broken down by values of another variable: Using a BY variable .....</b>	<b>23</b>
<b>Task: Computing the mean of a variable broken down by values of another variable: Using a CLASS statement .....</b>	<b>24</b>
<b>Task: Have PROC MEANS name the variables in the output data set automatically (the AUTONAME option) .....</b>	<b>25</b>
<b>Task: Creating multiple output data sets from PROC MEANS, each with a different combination of CLASS variables .....</b>	<b>26</b>
<b>Task: Combining summary information (a single mean) with detail data: Using a conditional SET statement .....</b>	<b>29</b>
<b>Task: Combining summary information (a single mean) with detail data: Using PROC SQL .....</b>	<b>31</b>
<b>Task: Combining summary information (a single mean) with detail data: Using PROC SQL without using PROC MEANS .....</b>	<b>32</b>
<b>Task: Combining summary information (a single mean) with detail data: Using a macro variable .....</b>	<b>33</b>
<b>Task: Combining summary data with detail data—for each category of a BY variable .....</b>	<b>34</b>
<b>Conclusion .....</b>	<b>36</b>
<b>Chapter 4 Combining and Updating SAS Data Sets .....</b>	<b>37</b>
<b>Introduction .....</b>	<b>37</b>
<b>Task: Concatenating two SAS data sets—Using a SET statement .....</b>	<b>38</b>
<b>Task: Concatenating two SAS data sets—Using PROC APPEND .....</b>	<b>40</b>
<b>Task: Concatenating two SAS data sets with character variables of different lengths .....</b>	<b>40</b>
<b>Task: Concatenating two SAS data sets that contain character variables of different lengths and controlling the length of the character variables .....</b>	<b>42</b>

<b>Task: Developing a macro to concatenate two SAS data sets that contain character variables of different lengths.....</b>	<b>43</b>
<b>Task: Updating a SAS data set using a transaction data set.....</b>	<b>47</b>
<b>Task: Using a MODIFY statement to update a master file from a transaction file .....</b>	<b>50</b>
<b>Task: Updating several variables using a transaction file created with an INPUT method called named input.....</b>	<b>50</b>
<b>Task: Matching names from two SAS data sets where the names may not be spelled the same (fuzzy merge).....</b>	<b>53</b>
<b>Conclusion .....</b>	<b>56</b>
<b>Chapter 5 Creating Formats from SAS Data Sets .....</b>	<b>57</b>
<b>Introduction.....</b>	<b>57</b>
<b>Task: Using a SAS data set to create a format (by creating a control data set).....</b>	<b>57</b>
<b>Task: Adding new format values to an existing format .....</b>	<b>62</b>
<b>Conclusion .....</b>	<b>63</b>
<b>Chapter 6 Table Lookup Techniques .....</b>	<b>65</b>
<b>Introduction.....</b>	<b>65</b>
<b>Task: Performing a one-way table lookup using a MERGE statement .....</b>	<b>65</b>
<b>Task: Performing a one-way table lookup using user-defined informats .....</b>	<b>67</b>
<b>Task: Creating an INFORMAT using a control data set.....</b>	<b>69</b>
<b>Task: Performing a one-way table lookup using a temporary array .....</b>	<b>70</b>
<b>Task: Performing a two-way table lookup using a temporary array .....</b>	<b>71</b>
<b>Conclusion .....</b>	<b>73</b>
<b>Chapter 7 Restructuring (Transposing) SAS Data Sets.....</b>	<b>75</b>
<b>Introduction.....</b>	<b>75</b>
<b>Task: Converting a data set with one observation per subject into one with multiple observations per subject (using a DATA step) .....</b>	<b>76</b>
<b>Task: Converting a data set with one observation per subject into one with multiple observations per subject (using PROC TRANSPOSE).....</b>	<b>77</b>
<b>Task: Converting a data set with multiple observations per subject into one with one observation per subject (using a DATA step).....</b>	<b>79</b>

<b>Task: Converting a data set with multiple observations per subject into one with one observation per subject (using PROC TRANSPOSE) .....</b>	<b>80</b>
<b>Conclusion .....</b>	<b>81</b>
<b>Chapter 8 Tasks Involving Dates .....</b>	<b>83</b>
<b>Introduction.....</b>	<b>83</b>
<b>Task: Computing a person's age, given his or her date of birth.....</b>	<b>83</b>
<b>Task: Computing a SAS date given a month, day, and year (even if the day value is missing).....</b>	<b>84</b>
<b>Conclusion .....</b>	<b>85</b>
<b>Chapter 9 Data Cleaning Tasks .....</b>	<b>87</b>
<b>Introduction.....</b>	<b>87</b>
<b>Task: Looking for possible data errors using a given range .....</b>	<b>88</b>
<b>Task: Demonstrating a macro to report on outliers using fixed ranges.....</b>	<b>89</b>
<b>Task: Demonstrating a macro that performs automatic outlier detection.....</b>	<b>92</b>
<b>How the macro works.....</b>	<b>94</b>
<b>Conclusion .....</b>	<b>96</b>
<b>Chapter 10 Reading Data with User-Defined INFORMATS .....</b>	<b>97</b>
<b>Introduction.....</b>	<b>97</b>
<b>Task: Reading a combination of character and numeric data.....</b>	<b>97</b>
<b>Conclusion .....</b>	<b>100</b>
<b>Chapter 11 Tasks Involving Multiple Observations per Subject .....</b>	<b>101</b>
<b>Introduction.....</b>	<b>101</b>
<b>Task: Using PROC SORT to detect duplicate BY values or duplicate observations (records) .....</b>	<b>102</b>
<b>Task: Extracting the first and last observation in a BY group.....</b>	<b>106</b>
<b>Task: Detecting duplicate BY values using a DATA step .....</b>	<b>108</b>
<b>Task: Identifying observations with exactly 'n' observations per subject.....</b>	<b>109</b>
<b>Task: Computing differences between observations (for each subject) .....</b>	<b>110</b>

<b>Task: Computing the difference between the first and last observation for each subject .....</b>	<b>112</b>
<b>Conclusion .....</b>	<b>114</b>
<b>Chapter 12 Miscellaneous Tasks .....</b>	<b>115</b>
<b>Introduction .....</b>	<b>116</b>
<b>Task: Determining the number of observations in a SAS data set (using the NOBS= SET option) .....</b>	<b>116</b>
<b>Task: Determining the number of observations in a SAS data set and assigning the value to a macro variable .....</b>	<b>117</b>
<b>Task: Determining the number of observations in a SAS data set (using library tables) .....</b>	<b>118</b>
<b>Task: Determining the number of observations in a SAS data set (using SAS functions) .....</b>	<b>119</b>
<b>Task: Counting the number of a specific response in a list of variables .....</b>	<b>120</b>
<b>Task: Computing a moving average .....</b>	<b>122</b>
<b>Task: Presenting a macro to compute a moving average .....</b>	<b>124</b>
<b>Task: Replacing the first eight digits of a credit card number with asterisks .....</b>	<b>126</b>
<b>Task: Sorting within an observation (using the ORDINAL function) .....</b>	<b>127</b>
<b>Task: Sorting within an observation (using CALL SORTN) .....</b>	<b>128</b>
<b>Task: Computing the average of the 'n' highest scores .....</b>	<b>129</b>
<b>Task: Extracting the first and last name (and possibly a middle name) from a variable containing the first and last name (and possibly a middle name) in a single variable .....</b>	<b>130</b>
<b>Index .....</b>	<b>133</b>

From *Cody's Collection of Popular SAS® Programming Tasks and How to Tackle Them* by Ron Cody.  
Copyright © 2012, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.



From *Cody's Collection of Popular SAS® Programming Tasks and How to Tackle Them*. Full book available for purchase [here](#).

## Chapter 1: Tasks Involving Conversion: Character to Numeric, Specific Values to Missing, and Changing Case

<b>Introduction .....</b>	<b>1</b>
<b>Task: Converting character values to numeric values .....</b>	<b>2</b>
<b>Task: Converting character values to numeric values using a macro .....</b>	<b>3</b>
<b>Task: Converting a specific value such as 999 to a missing value for all numeric variables in a SAS data set .....</b>	<b>5</b>
<b>Task: Converting a specific value such as 'NA' to a missing value for all character variables in a SAS data set .....</b>	<b>7</b>
<b>Task: Changing all character values to either uppercase, lowercase, or proper case .....</b>	<b>8</b>
<b>Task: Reading a numeric value that contains units such as Lbs. or Kgs. in the value .....</b>	<b>9</b>
<b>Task: Solving part of the previous task using a Perl regular expression .....</b>	<b>10</b>
<b>Conclusion .....</b>	<b>11</b>

---

### Introduction

This chapter contains programs to perform character-to-numeric conversion, one of the most common tasks you will face as a SAS programmer. You will see a sample program as well as a useful macro that accomplishes this goal.

Another task that you will probably face is converting a specific numeric value such as 999 or a specific text value such as 'NA' to a SAS missing value.

In this chapter, you will also see how to convert every character variable to a specific case, such as uppercase.

## 2 Cody's Collection of Popular SAS Programming Tasks and How to Tackle Them

The last task in this chapter demonstrates how to read data values that contain units, such as 100Lbs. or 50Kgs. and create a numeric variable with all of the values using the same units.

---

### Task: Converting character values to numeric values

#### Keywords

**Character-to-numeric conversion**

**Swap and Drop**

How many times have you been given a SAS data set with variables such as Height or Weight but, instead of being numeric variables, they are stored as character? The following example describes how to convert these character variables to numeric variables, maintaining the original variable names.

For this example, you start out with a SAS data set called Char\_values. Here is a listing:

Age	Weight	Gender	DOB
23	150	M	10/21/1983
67	220	M	09/12/2001
77	101	F	05/06/1977

If you run PROC CONTENTS on this data set, you see that Age and Weight are character variables. The following program performs the conversion:

#### Program 1.1: Converting character values to numeric values

```
*Converting character values to numeric;

data Num_Values;
  set Char_Values(rename=(Age = C_Age
                          Weight = C_Weight));
  Age = input(C_Age,best12.);
  Weight = input(C_Weight,best12.);
  drop C_;;
run;
```

The “trick” here is to rename the variables as they are read from the input data set. This allows you to use the original variable names for the resulting numeric variables. The character-to-numeric conversion is performed using the INPUT function. You don’t have to worry if the INFORMAT used in the INPUT function represents more digits than you need—unlike an INPUT statement, you can never read past the end of a character value when using the INPUT function.



Notice the variable list on the DROP statement `C_`: The colon acts as a wildcard suffix. `C_` represents all variables that begin with the characters C followed by an underscore.

The resulting data set has exactly the same variables as the original data set except the two variables Age and Weight are now numeric. A partial listing from PROC CONTENTS confirms this:

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Format
3	Age	Num	8	
2	DOB	Num	8	MMDDYY10.
1	Gender	Char	1	
4	Weight	Num	8	

---

## Task: Converting character values to numeric values using a macro

### Keywords

**Character-to-numeric conversion**

**Conversion macro**

Because character-to-numeric conversion is required in so many situations, this chapter offers you a macro that performs the conversion automatically. As in the previous program, the resulting data set uses the same variable names as in the original data set that contains the character variables. Here is the macro, followed by an explanation:

### Program 1.2: Presenting a macro to perform character-to-numeric conversion

```
*Macro to convert selected character variables to
numeric variables;
%macro char_to_num(In_dsn=, /*Name of the input data set*/
                  Out_dsn=, /*Name of the output data set*/
                  Var_list= /*List of character variables that you
                             want to convert from character to
                             numeric, separated by spaces*/);

    /*Check for null var list */
    %if &var_list ne %then %do;
    /*Count the number of variables in the list */
    %let n=%sysfunc(countw(&var_list));
    data &Out_dsn;
        set &In_dsn(rename=(
            %do i = 1 %to &n;
```

#### 4 *Cody's Collection of Popular SAS Programming Tasks and How to Tackle Them*

```
/* break up list into variable names */
  %let Var = %scan(&Var_list,&i);
/*Rename each variable name to C_ variable name */
  &Var = C_&Var
%end;
));

%do i = 1 %to &n;
  %let Var = %scan(&Var_list,&i);
  &Var = input(C_&Var,best12.);
%end;
drop C_::;
run;
%end;
%mend char_to_num;
```

The calling arguments in this macro are the names of the input and output data sets and a list of the variables that you wish to convert from character to numeric. You enter the names of each variable in this list, separated by spaces.

The first task of the macro is to rename each of the original variable names by appending the prefix `C_` to each of the names. To determine how many variable names there are in `&Var_list`, you use the `COUNTW` function. This function computes the number of words in a string. To obtain each of the variable names, you use the `%SCAN` macro function. This function works in the same way as the regular non-macro `SCAN` function. The first argument is the list of variable names. The second argument specifies which “word” you want in the string. The macro uses a `%DO` loop to extract each of the individual variable names.

The next `%DO` loop performs the character-to-numeric conversion using the `INPUT` function. Notice that the first argument of the `INPUT` function is the original variable name with the `C_` prefix added. Finally, a `DROP` statement deletes all of the `C_` variables.

To test the macro, you can use the original data set `Char_values` and enter `Age` and `Weight` as the argument of `Var_List`. Here is the code:

##### **Program 1.3: Testing the character-to-numeric conversion macro**

```
*Test the macro;
%char_to_num(In_dsn=char_values, Out_dsn=Num_values,
             Var_list=Age Weight)
```

After you run the macro, the output data set (`Num_values`) is identical to the one created by the previous program.

---

## Task: Converting a specific value such as 999 to a missing value for all numeric variables in a SAS data set

### Keywords

Numeric variables

`_numeric_`

Array

You will find numerous occasions where you need to perform an operation on all numeric (or character) variables in a SAS data set. For example, you may have a SAS data set where specific values, such as 999 or 9999, were used to represent a missing value. In the character domain, you may want to convert all character values to uppercase or convert a specific value such as 'NA' to a SAS missing value. The approach to all of these tasks is the same. You create an array of all numeric or character variables. Once you do this, you can then use a DO loop to perform any operation you desire on all of the variables in the array.

This first example converts a value of 999 to a SAS missing value for all the numeric variables in data set Demographic.

A listing of data set Demographic is shown here:

Subj	Score	Weight	Heart_Rate	DOB	Gender	Party
1	70	999	76	04NOV1955	Male	NA
2	26	160	62	08APR1955	NA	NA
3	71	195	71	20JUL1955	male	na
4	40	132	74	08JAN1955	Male	Republican
5	999	181	62	15AUG1951	Female	Democrat
6	62	71	52	24JAN1950	Male	democrat
7	24	136	72	26NOV1950	Female	democrat
8	5	174	71	08NOV1950	Female	democrat
9	5	172	47	28DEC1951	Male	Democrat
10	94	173	999	06MAY1953	Male	republican
11	99	170	63	27FEB1950	na	NA
12	10	133	63	18MAR1954	Male	democrat
13	6	131	60	26MAR1951	Female	republican

## 6 Cody's Collection of Popular SAS Programming Tasks and How to Tackle Them

Subj	Score	Weight	Heart_Rate	DOB	Gender	Party
14	999	140	79	01OCT1950	NA	na
15	999	124	999	12OCT1950	NA	na
16	44	194	72	31DEC1952	Female	republican
17	62	196	68	09MAR1951	Female	democrat
18	57	133	72	15SEP1951	Female	Democrat
19	45	137	86	16NOV1951	NA	Republican
20	90	170	80	01OCT1951	Female	Republican

You will use this data set for several of the tasks in this chapter. For this example, notice that there are several values of 999 for the variables Score, Weight, and Heart\_Rate.

Here is the code that performs the conversion:

### Program 1.4: Converting a specific value such as 999 to a missing value for all numeric variables in a SAS data set

```
*Converting a specific value such as 999 to a missing value for
all numeric variables in a SAS data set;

data Num_missing;
  set Demographic;
  array Nums[*] _numeric_;
  do i = 1 to dim(Nums);
    if Nums[i] = 999 then Nums[i] = .;
  end;
  drop i;
run;
```

The key to this program, as well as several programs to follow, is to create an array using the keyword `_NUMERIC_`. When used in a DATA step, `_NUMERIC_` represents all the numeric variables that have been defined up to that point in the DATA step. Since the ARRAY statement follows the SET statement, the Nums array contains all of the numeric variables in data set Demographic (Subj, Score, Heart\_Rate, and DOB). To make this important point clear, had you placed the ARRAY statement before the SET statement, the array Nums would not contain any variables.

You certainly do not want to have to count all the numeric variables in a large data set. Therefore, you use an asterisk in the brackets following the array name. When you do this, SAS will count the number of variables for you. But, what value do you use in the DO loop? You can use the DIM (dimension) function to determine how many variables are in the array. Your work is almost finished. All you need to do now is to check for values of 999 and convert them to a SAS numeric missing value. Don't forget to drop the DO loop counter.

The first five observations in data set Num\_missing are shown next, to demonstrate that the program worked as expected:

Subj	Score	Weight	Heart_Rate	DOB	Gender	Party
1	70	.	76	04NOV1955	Male	NA
2	26	160	62	08APR1955	NA	NA
3	71	195	71	20JUL1955	male	na
4	40	132	74	08JAN1955	Male	Republican
5	.	181	62	15AUG1951	Female	Democrat

---

## Task: Converting a specific value such as 'NA' to a missing value for all character variables in a SAS data set

### Keywords

Character variables

character\_Array

This task is similar to the previous task. The difference is that you want to convert a specified character value to a SAS character missing value. All you need to do is use the SAS keyword `_CHARACTER_` to create an array of all character variables. Here is the program:

### Program 1.5: Converting a specific value such as 'NA' to a missing value for all character variables in a SAS data set

```
*Converting a specific value such as "NA" to a missing value for all
character variables in a SAS data set;
data Char_missing;
  set Demographic;
  array Chars[*] _character_;
  do i = 1 to dim(Chars);
    if Chars[i] in ('NA' 'na') then Chars[i] = ' ';
  end;
  drop i;
run;
```

Array Chars contains all the character variables in data set Demographic (in this case, Gender and Party). As in the previous task, the DIM function returns the number of variables in the array. To make the program more general, it looks for uppercase or lowercase values of 'NA'. Here is a listing of the first five observations in data set Char\_missing:

Subj	Score	Weight	Heart_Rate	DOB	Gender	Party
1	70	999	76	04NOV1955	Male	
2	26	160	62	08APR1955		
3	71	195	71	20JUL1955	male	
4	40	132	74	08JAN1955	Male	Republican
5	999	181	62	15AUG1951	Female	Democrat

---

## Task: Changing all character values to either uppercase, lowercase, or proper case

### Keywords

Uppercase

Lowercase

Proper case

\_character\_

In a similar manner to the previous program, you can use an array of all your character variables to convert them all to a unified case: uppercase, lowercase, or proper case. Please refer to the previous program if you would like an explanation of this program. As you can see, this program is converting all the character values in the Demographic data set to uppercase. The two other functions that convert character values to lowercase or proper case are LOWCASE and PROPCASE, respectively. Here is the program:

### Program 1.6: Changing case for all character variables in a SAS data set

```
*Converting all character values to uppercase (or lower- or proper-
case);
data Upper;
  set Demographic;
  array Chars[*] _character_;
  do i = 1 to dim(Chars);
    Chars[i] = upcase(Chars[i]);
  end;
  drop i;
run;
```

If the character variables you are dealing with represent names and addresses, after you have converted all the values to a consistent case, you may want to take the additional step and use the COMPBL function to convert all multiple blanks to a single blank, to help standardize the names and addresses.

---

## Task: Reading a numeric value that contains units such as Lbs. or Kgs. in the value

### Keywords

**Character-to-numeric conversion**

**Removing units from a value**

**Extracting digits from a string**

**COMPRESS function**

**SCAN function**

Data set Units contains a character variable called Weight that includes units such as Lbs. and Kgs. (pounds and kilograms). To add insult to injury, the variable Height also contains units and it is expressed in feet and inches (sometimes the inches value is missing (when the inches value is zero). A listing of data set Units is shown here:

Subj	Weight	Height
001	80kgs	5ft 3in
002	190lbs	6' 1"
003	70KG.	5ft 11in
004	177LbS.	5' 11"
005	100kgs	6ft

Notice that the Weight units are not always in the same case and some of the units end in periods. For Height, the abbreviation 'ft' or 'in' is used; sometimes a single quote and double quote represent feet and inches.

You would like to create two new variables (Weight\_Lbs and Height\_Inches) that are numeric variables and are equal to the weight in pounds and the height in inches, respectively. Here is the program:

### Program 1.7: Reading data values that contain units

```
*Reading data values that contain units;
data No_Units;
  set Units;
  Weight_Lbs = input(compress(Weight,,'kd'),12.);
  if findc(Weight,'k','i') then Weight_Lbs = Weight_Lbs*2.2;
  Height = compress(Height,,'kds');
  Feet = input(scan(Height,1,' '),12.);
  Inches = input(scan(Height,2,' '),12.);
```

## 10 Cody's Collection of Popular SAS Programming Tasks and How to Tackle Them

```
if missing(Inches) then Inches = 0;  
Height_Inches = 12*Feet + Inches;  
drop Feet Inches;  
run;
```

You start by extracting the digits from Weight using the COMPRESS function with the modifiers 'kd' (keep digits). It is important to include two commas following the first argument of the COMPRESS function so that the function interprets 'kd' as modifiers and not the second argument to the COMPRESS function that is used to list the characters you want to compress from a string. Since the result of the COMPRESS function is a character value, you use the INPUT function to perform the character-to-numeric conversion. All you need to do is test the original variable (Weight) to see if it contains a 'K' in uppercase or lowercase. Use the FINDC function with the 'i' modifier (ignore case) to do this. If you find a 'K', you multiply by 2.2 to convert from kilograms to pounds.

The Height variable presents more of a challenge. You first use the COMPRESS function with three modifiers, 'kds' (keep digits and space characters). The variable Height now contains two sets of digits (or only a single digit if there are zero inches) and can use the SCAN function to extract the feet and inch values. The SCAN function returns a missing value for Inches if Height only contains a single number (feet). You can now add 12 times the feet plus the number of inches to obtain the height in inches. Here is the listing of the data set No\_Units:

Subj	Weight	Height	Weight_Lbs	Height_Inches
001	80kgs	5 3	176	63
002	190lbs	6 1	190	73
003	70KG.	5 11	154	71
004	177LbS.	5 11	177	71
005	100kgs	6	220	72

Solving this task without the COMPRESS and SCAN functions would certainly be a challenge—with these functions, it's a snap.

---

### Task: Solving part of the previous task using a Perl regular expression

#### Keywords

Removing units from a value

Extracting digits from a string

Perl regular expression



My younger son, who is a wizard at programming, suggested I solve this problem using a Perl regular expression. This solution is not simpler than the previous solution, but it demonstrates the versatility of regular expressions.

You start by using PRXPARSE to compile the regular expression:

```
/^(\d+) (\D) /
```

This regex (this is what Perl programmers call regular expressions) is looking for one or more digits followed by a non-digit. The ^ in the beginning of the expression says to start the search at the beginning of the string. The digit and non-digit values will be placed in capture buffers because each of these expressions is in a set of parentheses. You use the PRXMATCH function to search for the pattern of a number followed by a non-number. The PRXPOSN function extracts the values in each of the capture buffers. The INPUT function performs the character-to-numeric conversion as in the previous task.

If the value in the second capture buffer is a 'K', you perform the kilogram to pound conversion.

**Program 1.8: Using a Perl regular expression to extract the digit and units part of a character value**

```
*Solution using Perl Regular expressions;
data No_Units;
  set Units(drop=Height);
  if _n_ = 1 then do;
    Regex = "/^(\d+) (\D) /";
    re = prxparse(Regex);
  end;
  retain re;
  if prxmatch(re,Weight) then do;
    Weight_Lbs = input(prxposn(re,1,Weight),8.);
    Units = prxposn(re,2,Weight);
    if upcase(Units) = 'K' then Weight_Lbs = Weight_Lbs*2.2;
  end;
  keep Subj Weight Weight_Lbs;
run;
```

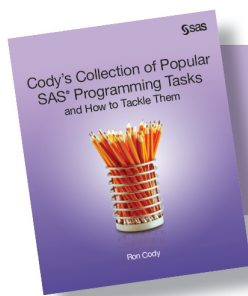
The resulting data set contains values for Weight\_Lbs that are identical to the values in the previous task.

---

## Conclusion

It is quite likely that you will need to perform one or more of the tasks described in this chapter on a regular basis. Since the character-to-numeric conversion is one of the most common tasks, you may choose to store the conversion macro in your macro library.

Also keep in mind that using the special keywords `_NUMERIC_` and `_CHARACTER_` to define an array can save you immense time when you need to perform an operation on all character or numeric variables in a data set.



From *Cody's Collection of Popular SAS® Programming Tasks and How to Tackle Them*. Full book available for purchase [here](#).

# Index

## A

age computation using birth date 83–84  
\_ALL\_ sort option 105–106  
APPEND procedure 37, 40, 41–42, 56, 90  
arrays, temporary 70–73  
ARRAY statement 6, 7, 70, 128  
ATTRN function 120  
Auto\_Outlier macro 92–96  
automatic outlier detection 92–96  
AUTONAME option of MEANS procedure 25–26  
average, computing moving 122–123, 124–125  
average of 'n' highest scores, computing 129

## B

blanks, converting multiple to single 8  
BY group, extracting first or last observation 106–107  
BY statement 35, 49, 78, 106–107  
BY values, detecting duplicate 102–106, 108  
BY variables 23, 34–35, 105–106, 109–110

## C

CALL SORTN call routine 128–129  
CALL SYMPUTX statement 33, 117–118  
Cartesian product 31, 53–54  
case, changing character values 5, 8, 55, 121  
CATS function 121, 126  
\_CHARACTER\_ keyword 7–8, 11  
character data  
  arrays of 7  
  converting case 5, 8, 55, 121  
  converting for all variables in data set 5  
  converting to numeric values 2–4, 5–7, 9–10, 11, 68  
  converting to specific value 5, 7–8  
  masking 126  
  reading with informats 98–100  
  reading with traditional approach 97–98  
  replacing in string 126

character variables, creating with PUT function 67

CHARTYPE option of MEANS procedure 26–29

CLASS statement 24, 34–35

CLOSE function 120

CNTLIN= option in FORMAT procedure 59–60, 70

CNTLOUT= option in FORMAT procedure 62

COALESCE function 85

COMPBL function 8

COMPRESS function, to extract Weight digits 10

concatenating data sets 37, 38–40, 42–47

conditional SET statement 29–31, 34

CONTENTS procedure 39, 43, 44

control data sets 57–61, 62–63, 69–70

converting characters

  all variables in data set 5

  to desired case 5, 8, 55, 121

  to numeric values 2–4, 5–7, 9–10, 11, 68

  to specific value 5, 7–8

converting macro variables, to numeric values 118

converting numeric variables

  all variables in data set 5–7

  from character values 2–4, 5–7, 9–10, 11, 68

  from macro variables 118

COUNTC function 121–122

counting numeric variables 6

counting specific values in list of variables 120–122

COUNTW function 4

credit card number, masking 126

## D

DATA= attribute for TRANSPOSE procedure 78

data cleaning 50–52, 56, 89–91, 92–96

*See also* data errors

data conversion  
  all numeric variables in data set 5–7  
  character to numeric 2–4, 5–7, 9–10, 11, 68  
  macro variables to numeric 118  
  to desired case 5, 8, 55, 121  
  to specific value 5, 7–8

data errors 38, 87, 88, 89–91

data sets  
  adding data to existing 37  
  concatenating with APPEND procedure 42  
  concatenating with different character  
    variable lengths 37, 38  
  concatenating with different character  
    variable lengths using APPEND  
    procedure 40–42  
  concatenating with different character  
    variable lengths using macro 43–47  
  concatenating with different character  
    variable lengths using SET and LENGTH  
    statements 42–43  
  concatenating with SET statement 37, 38–40,  
    42–43  
  control 57–61, 62–63, 69–70  
  correcting data errors 38  
  restructuring 76–77, 79–80, 81  
  updating using transaction data set 47–49  
  updating with new values 38

DATA step  
  concatenating data sets with different variable  
    lengths 45  
  \_NUMERIC\_ keyword 6, 11  
  restructuring data sets 76–77, 79–80, 81  
  stopping 118  
  summarized data, combining with detail data  
    34  
  summarizing data 30, 31

dates 83–85

deciles, dividing values into 15  
  *See also* RANK procedure

Delobs variable in library tables 119

differences between observations, computing  
  110–112

DIF function 111

digits, extracting from strings 9–10, 11

DIM (dimension) function, use in DO loop 6, 7

DO loops 4, 6, 7

DROP= data set option 79

DROP= option in MEANS procedure 28

DROP statement, use of colon in variable list 2,  
  3

duplicate BY values, detecting 102–106, 108

duplicates 102–106, 108

dynamic naming of variables 128

## E

END= option in MERGE statement 45

END= option in SET statement 61, 62

End variable in control data set 58, 69

enhanced numeric informat 99

ERROR MERGENOBY option 44, 45

errors macro for range checking 89–91

%EVAL function 118

EXIST function 120

extracting digits from string 9–10, 11

## F

file I/O functions 119–120

FINDC function 10

FIRSTOBS= data set option 118

first observation 106–107, 112–113

First. variables 106–107, 108

Fmtname variable in control data set 58, 59

FORCE option, use when combining data sets  
  with different variable lengths 41–42

FORMAT procedure 59–63, 70, 98–100

formats  
  *See also* informats  
  adding formats to 62–63  
  creating 57, 68, 99  
  grouping values 14–15  
  using control data set 57–61, 69–70

\_FREQ\_ variable of MEANS procedure 23, 26

fuzzy matches, between two data sets 38, 53–56

fuzzy merge 53–56

## G

grand mean 22, 24, 27, 29–31

grouping values 13–17, 19

GROUPS= option in RANK procedure 16, 17–18

## H

Height variable in Units data set 9–10  
HISTOGRAM statement 96  
HLO variable in control data set 58, 60–61, 62

## I

if-then-else statements, grouping values 13–14  
%INCLUDE statement, for concatenating data sets with different variable lengths 43  
inexact matching 38, 53–56  
informats 57–61, 67–70, 98–100  
INFORMAT statement, in INPUT statement 51  
INPUT function  
    creating informat 68  
    informats in 67  
    named input method 50–52, 56  
    reading past end of character, inability 2  
    to convert character values to numeric 10, 11  
input/output (I/O) functions 119–120  
INVALUE statement, to create informat 68, 99

## J

Jordan, Mark 33, 85  
JUST option, in INVALUE statement 99

## K

KEEP= option in RANK procedure 17  
KEEP= option of MEANS procedure 22, 23  
kilograms, converting to pounds 10

## L

Label variable in control data set 58, 61  
LAG function 111, 113, 122, 124–125  
Langston, Rick 129  
last observation 106–107, 112–113  
Last. variables 106–107, 108  
LENGTH statement 14, 43, 45, 51, 62  
%LET statement 90, 118  
libnames format 119

library tables, determining number of observations 118–119

LOWCASE function 8

lowercase, converting characters to 8

## M

macros

    concatenate data sets with different variable lengths 43–47  
    convert character values to numeric 3–4  
    converting program to 46  
    moving average 124–125  
    MPRINT option to view code 125  
    outlier detection, 92–96  
    range checking 89–91  
    reporting 89, 90–91  
    running, example 4  
    %SCAN function 4

macro variables 33–34, 117–118

masking characters 126

master file, updating from transaction file 50–52

MDY function 84–85

mean 32–33

MEAN function 32–33, 125, 129

MEANS procedure

    automatic naming of variables in output 25–26  
    BY statement 35  
    BY variable 23  
    CLASS statement 24, 34–35  
    computing mean by values of another variable 24–25  
    creating multiple output data sets 26–29  
    options 22, 23, 24–29, 30, 35  
    OUTPUT statement 22, 25–26  
    sorting 35  
    SQL procedure as alternative 32–33  
    summarizing data 22, 23, 24–29  
    variables 23, 24, 26–27, 28  
    VAR statement 22

memname format 119

merge, fuzzy 53–56

MERGENOBY system option 44–45

MERGE statement 30, 34–36, 44–45, 49, 65–67

metadata, getting data set 118  
 MISSING function 14, 85, 122  
 missing values 5–8, 30, 127  
 MODIFY statement, updating using transaction  
   file 50, 52  
 moving average, computing 122–123, 124–125  
 MPRINT option to view macro generated code  
   125  
 multiple observations per subject 76–81, 101,  
   110–113

## N

named input method 50–52, 56  
 names, parsing 130–131  
 NLOBS argument of ATTRN function 120  
 NOBS= option in SET statement 116–118, 119  
 "n" observations per subject 109–110  
 Nobs variable in library tables 119  
 NODUPKEY option in SORT procedure  
   102–103  
 NODUPRECS option in SORT procedure  
   103–106  
 NOPRINT option of MEANS procedure 22, 30  
 NOWARN MERGENOBY option 44–45  
 number of observations per subject, detecting  
   109–110, 116–118, 119  
 \_NUMERIC\_ keyword 6, 11  
 numeric informat, enhanced 99  
 numeric values  
   converting all variables in data set 5–7  
   converting from character values 2–4, 5–7,  
     9–10, 11, 68  
   converting from macro variables 118  
   creating with informats 68  
   extracting from strings 9–10, 11  
   reading 97–100  
 numeric variables, counting 6  
 NVAR argument of ATTRN function 120  
 NWAY option of MEANS procedure 24–25, 26,  
   35

## O

observations 109–113, 116–120  
*See also* restructuring data sets

ODS OUTPUT statement 94  
 one-way table lookup, using temporary arrays  
   70–71  
 OPEN function 120  
 ORDINAL function 127–128, 129  
 OUT= attribute for TRANSPOSE procedure 78  
 OUT= keyword for MEANS procedure 22, 78  
 outliers, data 87, 88, 89, 92–96  
 OUTPUT statement 22, 25–26, 44

## P

parsing strings 130–131  
 PDV (program data vector), effect of observation  
   processing on 106  
 Perl regular expressions (regex), to extract digits  
   11  
 PREFIX= option for TRANSPOSE procedure  
   80–81  
 PRINT procedure, REPORT procedure  
   compared 60  
 program data vector (PDV), effect of observation  
   processing on 106  
 PROPCASE function 8  
 proper case, converting characters to 8  
 PRXMATCH function 11  
 PRXPARSE function 11  
 PRXPOSN function 11  
 PUT function 14–15, 67, 68

## Q

questionnaires, counting specific values in list of  
   variables 120–122

## R

range checking 88, 89  
 RANK procedure 15–18  
 RANKS statement, in RANK procedure 16, 17  
 reading data 2, 97–100  
 regex (Perl regular expressions) 11  
 removing units from value 9–10, 11  
 RENAME= data set option 59, 79  
 renaming variables 2, 3–4

report macro for reporting range errors 89,  
90–91  
REPORT procedure 60  
resources on SAS, web site 125  
restructuring data sets  
    multiple observations per subject into one  
        observation per subject 79–81  
    need for 75–76  
    one observation per subject into multiple  
        observations per subject 76–79  
RETAIN statement 59, 106, 112–113

## S

\_SAME\_ keyword in INVALUE statement 99  
SAS 9.3, version changes 84  
SAS dates, computing with MDY function  
    84–85  
SAS Global Forum 29  
SASHELP.VTABLE, getting metadata from  
    118–119  
SAS Users Group International (SUGI) 29  
SCAN function 4, 10, 130–131  
SELECT statement, for documentation 60  
SET statement  
    adding to existing data sets 37  
    APPEND procedure compared 56  
    ARRAY statement with 6  
    concatenating data sets 37, 38–40, 42–43  
    conditional 29–31, 34  
    followed by BY statement 106–107  
    options 61, 62, 116–118, 119  
SET SUMMARY statement 30, 31  
single observations per subject  
    converting from multiple observations per  
        subject data set to 79–81  
    converting to multiple observations per  
        subject data set 76–79  
SMALLEST function 127  
sorting 35, 66–67, 127–129  
SORTN call routine 128–129  
SORT procedure 35, 102–106, 108  
SPEDIS (spelling distance) function 53, 55  
SQL procedure 31–34, 53–54  
Start variable in control data set 58, 69

statistical methods, for identifying possible  
    outliers 87, 92  
STOP statement in DATA step 118  
strings 9–10, 11, 126, 130–131  
    *See also* character data  
SUBSTR function 126  
SUGI (SAS Users Group International) 29  
summarized data, combining with detail data  
    conditional SET statement 29–31  
    DATA step 34  
    macro variable 33–34  
    MERGE statement 34–36  
    SQL procedure 31–34  
summarizing values  
    in DATA step 30, 31  
    macro variable 33–34  
    MEANS procedure 22, 23, 24–29  
    SQL procedure 31–33  
    uses 22  
SUM statement 110  
swap and drop 2–3  
SYMPUTX call statement 33, 117–118

## T

table lookup 65–68, 70–73  
tables, creating with SQL procedure 31  
TABULATE procedure 21  
\_TEMPORARY\_ keyword in ARRAY statement  
    70  
temporary arrays 70–73  
transaction data set, updating data sets with  
    47–49  
transaction file, updating master file using  
    50–52  
TRANSPOSE procedure 77–81  
transposing data sets  
    *See* restructuring data sets  
trends, calculating 122–123, 124–125  
trimmed statistics for automatic outlier detection  
    92, 93  
trim value for outlier detection 96  
two-way table lookup, using temporary arrays  
    71–73

\_TYPE\_ variable in MEANS procedure 23, 24,  
26–27, 28  
Type variable in control data set 58, 59

## U

units, removing from value 9–10, 11  
Units data set 9  
UNIVARIATE procedure 21, 92, 96  
UPCASE function 8, 55, 121  
UPCASE option in INVALUE statement 99  
UPDATE statement 49, 52  
updating master file 50–52, 56  
uppercase, converting characters to 5, 8, 55, 121

## V

values, grouping 13–17, 19  
VALUE statement, to create formats, 68, 99  
variables  
    macro 33–34, 117–118  
    naming 2, 3–4, 128  
    representing groups 13–17, 19  
VAR statement in MEANS procedure 22  
VAR statement in RANK procedure 16, 17  
VAR statement in TRANSPOSE procedure 80  
VNAME function 94

## W

WARN MERGENOBY option 44, 45  
web site, SAS 125  
Weight variable in Units data set 9–10  
WHERE= data set option 44, 79

## Y

YRDIF function 84

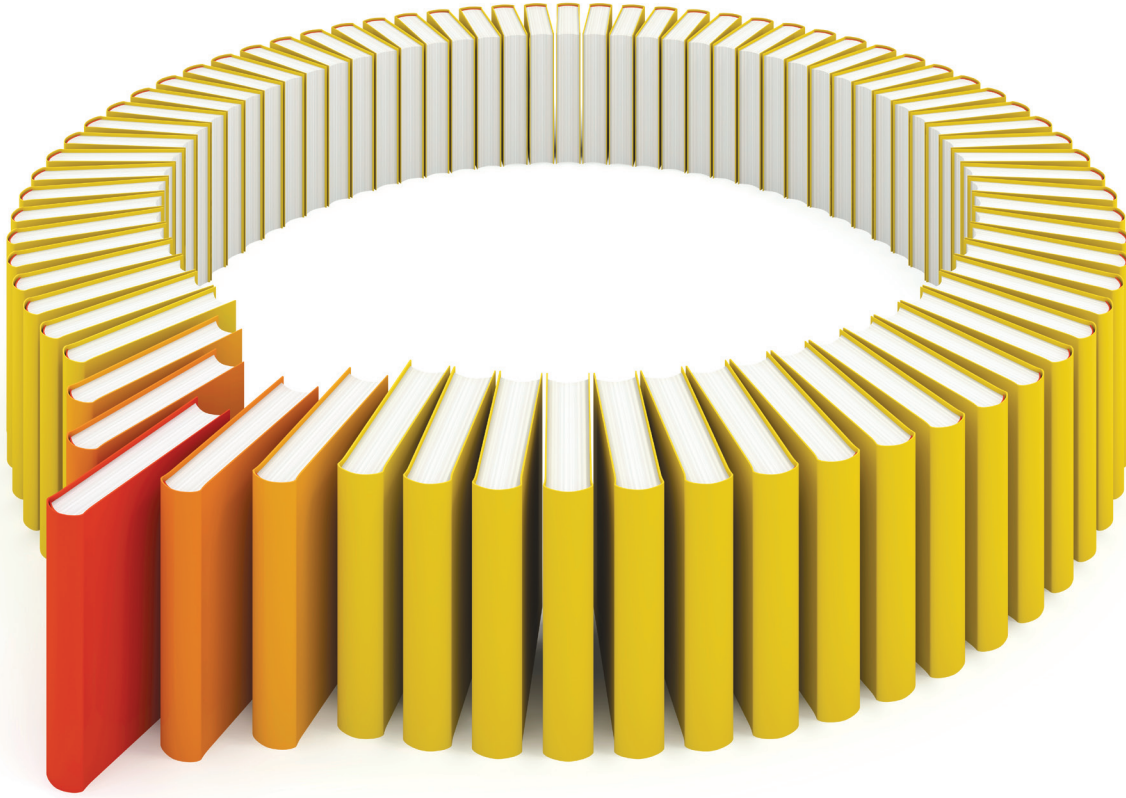


## About The Author



Ron Cody, EdD, is a retired professor from the Robert Wood Johnson Medical School who now works as a private consultant and a national instructor for SAS. A SAS user since 1977, Ron's extensive knowledge and innovative style have made him a popular presenter at local, regional, and national SAS conferences. He has authored or co-authored numerous books.

Learn more about this author by visiting his author page at [support.sas.com/cody](https://support.sas.com/cody). There you can download free chapters, access example code and data, read the latest reviews, get updates, and more.



# Gain Greater Insight into Your SAS<sup>®</sup> Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

 [support.sas.com/bookstore](http://support.sas.com/bookstore)  
for additional books and resources.

  
THE POWER TO KNOW<sup>®</sup>

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. © 2013 SAS Institute Inc. All rights reserved. S107969US.0613