Overview

The SORT procedure sorts observations in a SAS data set by one or more character or numeric variables, either replacing the original data set or creating a new, sorted data set. PROC SORT by itself produces no printed output.

Output 33.1 on page 1005 shows the results of sorting a data set with the most basic form of a PROC SORT step. In this example, PROC SORT replaces the original data set, sorted alphabetically by last name, with a data set that is sorted by employee identification number. The statements that produce the output follow:
Output 33.1  Observations Sorted by the Values of One Variable

<table>
<thead>
<tr>
<th>obs</th>
<th>name</th>
<th>idnumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>belloit</td>
<td>1988</td>
</tr>
<tr>
<td>2</td>
<td>wesley</td>
<td>2092</td>
</tr>
<tr>
<td>3</td>
<td>lemeux</td>
<td>4210</td>
</tr>
<tr>
<td>4</td>
<td>arnsbarger</td>
<td>5466</td>
</tr>
<tr>
<td>5</td>
<td>pierce</td>
<td>5779</td>
</tr>
<tr>
<td>6</td>
<td>capshaw</td>
<td>7338</td>
</tr>
</tbody>
</table>

Output 33.2 on page 1006 shows the results of a more complicated sort by three variables. The businesses in this example are sorted by town, then by debt from highest amount to lowest amount, then by account number. For an explanation of the program that produces this output, see Example 2 on page 1016.

Output 33.2  Observations Sorted by the Values of Multiple Variables

<table>
<thead>
<tr>
<th>obs</th>
<th>company</th>
<th>town</th>
<th>debt</th>
<th>account number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>paul's pizza</td>
<td>apex</td>
<td>83.00</td>
<td>1019</td>
</tr>
<tr>
<td>2</td>
<td>peter's auto parts</td>
<td>apex</td>
<td>65.79</td>
<td>7288</td>
</tr>
<tr>
<td>3</td>
<td>watson tabor travel</td>
<td>apex</td>
<td>37.95</td>
<td>3131</td>
</tr>
<tr>
<td>4</td>
<td>tina's pet shop</td>
<td>apex</td>
<td>37.95</td>
<td>5108</td>
</tr>
<tr>
<td>5</td>
<td>apex catering</td>
<td>apex</td>
<td>37.95</td>
<td>9923</td>
</tr>
<tr>
<td>6</td>
<td>deluxe hardware</td>
<td>garner</td>
<td>467.12</td>
<td>8941</td>
</tr>
<tr>
<td>7</td>
<td>boyd &amp; sons accounting</td>
<td>garner</td>
<td>312.49</td>
<td>4762</td>
</tr>
<tr>
<td>8</td>
<td>world wide electronics</td>
<td>garner</td>
<td>119.95</td>
<td>1122</td>
</tr>
<tr>
<td>9</td>
<td>elway piano and organ</td>
<td>garner</td>
<td>65.79</td>
<td>5217</td>
</tr>
<tr>
<td>10</td>
<td>ice cream delight</td>
<td>holly springs</td>
<td>299.98</td>
<td>2310</td>
</tr>
<tr>
<td>11</td>
<td>tim's burger stand</td>
<td>holly springs</td>
<td>119.95</td>
<td>6335</td>
</tr>
<tr>
<td>12</td>
<td>strickland industries</td>
<td>morrisville</td>
<td>657.22</td>
<td>1675</td>
</tr>
<tr>
<td>13</td>
<td>pauline's antiques</td>
<td>morrisville</td>
<td>302.05</td>
<td>9112</td>
</tr>
<tr>
<td>14</td>
<td>bob's beds</td>
<td>morrisville</td>
<td>119.95</td>
<td>4998</td>
</tr>
</tbody>
</table>

Note: The sorting capabilities that are described in this chapter are available on all operating environments. In addition, if you use the HOST value of the SAS system option SORTPGM=, you may be able to use other sorting options available only in your operating environment. Refer to the SAS documentation for your operating environment for information on other sorting capabilities. For more information about the SAS system option SORTPGM=, see the chapter on SAS system options in SAS Language Reference: Dictionary.
The **SORT** Procedure

**PROC SORT Statement**

**PROC SORT** <option(s)> <collating-sequence-option>;

**BY** <DESCENDING> variable-1 <…<DESCENDING> variable-n>;

---

**Procedure Syntax**

**Requirements:** BY statement

**Reminder:** You can use the ATTRIB, FORMAT, LABEL, and WHERE statements. See Chapter 3, "Statements with the Same Function in Multiple Procedures," for details. You can also use any global statements as well. See Chapter 2, "Fundamental Concepts for Using Base SAS Procedures," for a list.

**PROC SORT** <option(s)> <collating-sequence-option>;

**BY** <DESCENDING> variable-1 <…<DESCENDING> variable-n>;

---

**PROC SORT Statement**

**PROC SORT** <option(s)> <collating-sequence-option>;

<table>
<thead>
<tr>
<th>To do this</th>
<th>Use this option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specify the input data set</td>
<td>DATA=</td>
</tr>
<tr>
<td>Create an output data set</td>
<td>OUT=</td>
</tr>
<tr>
<td>Specify the collating sequence</td>
<td></td>
</tr>
<tr>
<td>Specify ASCII</td>
<td>ASCII</td>
</tr>
<tr>
<td>Specify EBCDIC</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>Specify Danish</td>
<td>DANISH</td>
</tr>
<tr>
<td>Specify Finnish</td>
<td>FINNISH</td>
</tr>
<tr>
<td>Specify Norwegian</td>
<td>NORWEGIAN</td>
</tr>
<tr>
<td>Specify Swedish</td>
<td>SWEDISH</td>
</tr>
<tr>
<td>Specify a customized sequence</td>
<td>NATIONAL</td>
</tr>
<tr>
<td>Specify any of these collating sequences: ASCII, EBCDIC, DANISH, FINNISH, ITALIAN, NORWEGIAN, SPANISH, SWEDISH</td>
<td>SORTSEQ=</td>
</tr>
<tr>
<td>Specify the output order</td>
<td></td>
</tr>
<tr>
<td>Reverse the order for character variables</td>
<td>REVERSE</td>
</tr>
<tr>
<td>Maintain the order within BY groups</td>
<td>EQUALS</td>
</tr>
<tr>
<td>Allow for variation within BY groups</td>
<td>NOEQUALS</td>
</tr>
<tr>
<td>Eliminate duplicate observations</td>
<td></td>
</tr>
<tr>
<td>Delete observations with common BY values</td>
<td>NODUPKEY</td>
</tr>
<tr>
<td>Delete observations that have duplicate values</td>
<td>NODUPRECS</td>
</tr>
<tr>
<td>Specify the available memory</td>
<td>SORTSIZE=</td>
</tr>
</tbody>
</table>
To do this | Use this option
---|---
Force redundant sorting | FORCE
Reduce temporary disk usage | TAGSORT

Options

**ASCII**

sorts character variables using the ASCII collating sequence. You need this option only when you sort by ASCII on a system where EBCDIC is the native collating sequence.

**Restriction:** You can specify only one collating sequence option in a PROC SORT step.

**See also:** “Sorting Orders for Character Variables” on page 1012

**Default:** NO

**DANISH**

NORWEGIAN

sort characters according to the Danish and Norwegian national standard.

The Danish and Norwegian collating sequence is shown in Figure 33.1 on page 1011.

**Operating Environment Information:** For information about operating environment-specific behavior, see the SAS documentation for your operating environment.

**Restriction:** You can specify only one collating sequence option in a PROC SORT step.

**DATA= SAS-data-set**

identifies the input SAS data set.

**Main discussion:** “Input Data Sets” on page 18

**EBCDIC**

sorts character variables using the EBCDIC collating sequence. You need this option only when you sort by EBCDIC on a system where ASCII is the native collating sequence.

**Restriction:** You can specify only one collating sequence option in a PROC SORT step.

**See also:** “Sorting Orders for Character Variables” on page 1012

**EQUALS | NOEQUALS**

specifies the order of the observations in the output data set. For observations with identical BY-variable values, EQUALS maintains the order from the input data set in the output data set. NOEQUALS does not necessarily preserve this order in the output data set.

**Default:** EQUALS

**Interaction:** When you use NODUPRECS to remove consecutive duplicate observations in the output data set, the choice of EQUALS or NOEQUALS can have an effect on which observations are removed.

**Tip:** Using NOEQUALS can save CPU time and memory.

**FINNISH**

**SWEDISH**

sort characters according to the Finnish and Swedish national standard. The Finnish and Swedish collating sequence is shown in Figure 33.1 on page 1011.
**Operating Environment Information:** For information about operating environment-specific behavior, see the SAS documentation for your operating environment.

**Restriction:** You can specify only one collating sequence option in a PROC SORT step.

**FORCE** sorts and replaces an indexed or subsetted data set when the OUT= option is not specified. Without the FORCE option, PROC SORT does not sort and replace an indexed data set because sorting destroys user-created indexes for the data set. When you specify FORCE, PROC SORT sorts and replaces the data set and destroys all user-created indexes for the data set. Indexes that were created or required by integrity constraints are preserved.

**Tip:** Since, by default, PROC SORT does not sort a data set according to how it is already sorted, you can use FORCE to override this behavior. This might be necessary if the SAS System cannot verify the sort specification in the data set option SORTEDBY=. For information about SORTEDBY=, see the section on SAS system options in **SAS Language Reference: Dictionary**.

**Restriction:** You cannot use PROC SORT with the FORCE option and without the OUT= option on data sets that were created with the Version 5 compatibility engine or with a sequential engine such as a tape format engine.

**NATIONAL** sorts character variables using an alternate collating sequence, as defined by your installation, to reflect a country's National Use Differences. To use this option, your site must have a customized national sort sequence defined. Check with the SAS Installation Representative at your site to determine if a customized national sort sequence is available.

**Restriction:** You can specify only one collating sequence option in a PROC SORT step.

**NODUPKEY** checks for and eliminates observations with duplicate BY values. If you specify this option, PROC SORT compares all BY values for each observation to those for the previous observation written to the output data set. If an exact match is found, the observation is not written to the output data set.

**Operating Environment Information:** If you use the VMS operating environment sort, the observation that is written to the output data set is not always the first observation of the BY group.

**See also:** NODUPRECS

**Featured in:** Example 3 on page 1018

**NODUPRECS** checks for and eliminates duplicate observations. If you specify this option, PROC SORT compares all variable values for each observation to those for the previous observation that was written to the output data set. If an exact match is found, the observation is not written to the output data set.

**Alias:** NODUP

**Interaction:** When you are removing consecutive duplicate observations in the output data set with NODUPRECS, the choice of EQUALS or NOEQUALS can have an effect on which observations are removed.

**Interaction:** The action of NODUPRECS is directly related to the setting of the SORTDUP data set option. When SORTDUP= is set to LOGICAL, NODUPRECS removes only the duplicate variables that are present in the input data set after a
DROP or KEEP operation. Setting SORTDUP=LOGICAL increases the number of duplicate records that are removed because it eliminates variables before record comparisons takes place. Also, setting SORTDUP=LOGICAL can improve performance because dropping variables before sorting reduces the amount of memory required to perform the sort. When SORTDUP= is set to PHYSICAL, NODUPRECS removes all duplicate variables in the data set, regardless if they have been kept or dropped. For more information about the data set option SORTDUP=, see SAS Language Reference: Dictionary.

Tip: Because NODUPRECS checks only consecutive observations, some nonconsecutive duplicate observations may remain in the output data set. You can remove all duplicates with this option by sorting on all variables.

See also: NODUPKEY

NOEQUALS
See EQUALS | NOEQUALS.

NORWEGIAN
See DANISH.

OUT=SAS-data-set
names the output data set. If SAS-data-set does not exist, PROC SORT creates it.

Default: Without OUT=, PROC SORT overwrites the original data set.

Tip: You can use data set options with OUT=.

Featured in: Example 1 on page 1014

REVERSE
sorts character variables using a collating sequence that is reversed from the normal collating sequence.

Interaction: Using REVERSE with the DESCENDING option in the BY statement restores the sequence to the normal order.

See also: The DESCENDING option in the BY statement. The difference is that the DESCENDING option can be used with both character and numeric variables.

SORTSEQ= collating-sequence
specifies the collating sequence. The value of collating-sequence can be any one of the individual options in the PROC SORT statement that specify a collating sequence, or the value can be the name of a translation table, either a default translation table or one that you have created in the TRANTAB procedure. For an example of using PROC TRANTAB and PROC SORT with SORTSEQ=, see Example 6 on page 1311. The available translation tables are

- Danish
- Finnish
- Italian
- Norwegian
- Spanish
- Swedish

To see how the alphanumeric characters in each language will sort, refer to Figure 33.1 on page 1011.

Restriction: You can specify only one collating sequence, either by SORTSEQ= or by one of the individual options that are available in the PROC SORT statement.
**SORTSIZE=memory-specification**

specifies the maximum amount of memory that is available to PROC SORT.

*memory-specification* is one of the following:

- **MAX**
  - specifies that all available memory can be used.

- **n**
  - specifies the amount of memory in bytes, where *n* is a real number.

- **nK**
  - specifies the amount of memory in kilobytes, where *n* is a real number.

- **nM**
  - specifies the amount of memory in megabytes, where *n* is a real number.

- **nG**
  - specifies the amount of memory in gigabytes, where *n* is a real number.

Specifying the SORTSIZE= option in the PROC SORT statement temporarily overrides the SAS system option SORTSIZE=. For information about the system option, see the section on SAS system options in *SAS Language Reference: Dictionary*.

**Operating Environment Information:** Some system sort utilities may treat this option differently. Refer to the SAS documentation for your operating environment.

**Default:** the value of the SAS system option SORTSIZE=

**Tip:** This option can help improve sort performance by restricting the virtual memory paging that the operating environment controls. If PROC SORT needs more memory, it uses a temporary utility file. As a general rule, the value of SORTSIZE should not exceed the amount of physical memory that will be available to the sorting process.

**SWEDISH**

See FINNISH.

**TAGSORT**

stores only the BY variables and the observation numbers in temporary files. The BY variables and the observation numbers are called tags. At the completion of the sorting process, PROC SORT uses the tags to retrieve records from the input data set in sorted order.

**Tip:** When the total length of BY variables is small compared with the record length, TAGSORT reduces temporary disk usage considerably. However, processing time may be much higher.
BY Statement

Specifies the sorting variables.

Featured in: Example 1 on page 1014, Example 2 on page 1016, and Example 3 on page 1018

**BY <DESCENDING> variable-1 <...<DESCENDING> variable-n>;**

Required Arguments

**variable**

specifies the variable by which PROC SORT sorts the observations. PROC SORT first arranges the data set by the values in ascending order, by default, of the first BY variable. PROC SORT then arranges any observations that have the same value of the first BY variable by the values in ascending order of the second BY variable. This sorting continues for every specified BY variable.

Option

**DESCENDING**

reverses the sort order for the variable that immediately follows in the statement so that observations are sorted from the largest value to the smallest value.

.Featured in: Example 2 on page 1016

Concepts

**Sorting Orders for Numeric Variables**

For numeric variables, the smallest-to-largest comparison sequence is

1 SAS System missing values (shown as a period or special missing value)
2 negative numeric values
3 zero
4 positive numeric values.

**Sorting Orders for Character Variables**

PROC SORT uses either the EBCDIC or the ASCII collating sequence when it compares character values, depending on the environment under which the procedure is running.

**EBCDIC Order**

The operating environments that use the EBCDIC collating sequence include CMS and OS/390.
The sorting order of the English-language EBCDIC sequence is

blank . < ( + | & $ * ; . / , % _ > ?: # @ '='

abcdefghijklmnopqrstuvwxyz
{ABCDEFGHIJKLMNOPQRSTUVWXYZ
UVWXYZ
0123456789

The main features of the EBCDIC sequence are that lowercase letters are sorted before uppercase letters, and uppercase letters are sorted before digits. Note also that some special characters interrupt the alphabetic sequences. The blank is the smallest displayable character.

**ASCII Order**

The operating environments that use the ASCII collating sequence include

- Macintosh
- PC DOS
- MS-DOS
- UNIX and its derivatives
- OpenVMS
- Windows
- OS/2

From the smallest to largest displayable character, the English-language ASCII sequence is

blank ! " # $ % & ’ ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 ; ; ; < = > ? @

ABCDEFGHIJKLMNOPQRSTUVWXYZ\[\]
abcdefghijklmnopqrstuvwxyz\{\~

The main features of the ASCII sequence are that digits are sorted before uppercase letters, and uppercase letters are sorted before lowercase letters. The blank is the smallest displayable character.

**Stored Sort Information**

PROC SORT records the BY variables, collating sequence, and character set that it uses to sort the data set. This information is stored with the data set to help avoid unnecessary sorts.

Before PROC SORT sorts a data set, it checks the stored sort information. If you try to sort a data set the way that it is currently sorted, PROC SORT does not perform the sort and writes a message to the log to that effect. To override this behavior, use the FORCE option. If you try to sort a data set the way that it is currently sorted and you specify an OUT= data set, PROC SORT simply makes a copy of the DATA= data set.

To override the sort information that PROC SORT stores, use the _NULL_ value with the SORTEDBY= data set option. For information about SORTEDBY=, see the section on data set options in *SAS Language Reference: Dictionary*.

If you want to change the sort information for an existing data set, use the SORTEDBY= data set option in the MODIFY statement in the DATASETS procedure.
To access the sort information that is stored with a data set, use the CONTENTS statement in PROC DATASETS. For details, see Chapter 14, “The DATASETS Procedure,” on page 329.

### Integrity Constraints

Sorting the data set in place without OUT= preserves both referential and general integrity constraints, as well as any indexes that they may require. A sort using the OUT= option will not preserve any integrity constraints or indexes. For more information on integrity constraints, see the section on SAS files in *SAS Language Reference: Concepts*.

### Results

#### Procedure Output

PROC SORT produces only an output data set. To see the output data set, you can use PROC PRINT, PROC REPORT, or another of the many available methods of printing in the SAS System.

#### Output Data Set

When you specify the OUT= option, PROC SORT creates a new data set that contains the sorted observations. Without OUT=, PROC SORT replaces the original data set with the sorted observations as soon as the procedure executes without errors. Even when a data set is replaced, there must be at least enough space in the data library for a second copy of the original data set.

You can also sort compressed data sets. If you specify a compressed data set as the input data set and omit the OUT= option, the input data set is sorted and remains compressed. If you specify an OUT= data set, the resulting data set is compressed only if you choose a compression method with the COMPRESS= data set option. For more information about the data set option COMPRESS=, see the section on SAS data set options in *SAS Language Reference: Dictionary*.

*Note:* If the SAS system option NOREPLACE is in effect, you cannot replace the original data set with the sorted version. You must either use the OUT= option or specify the SAS system option REPLACE in an OPTIONS statement. △

### Examples

#### Example 1: Sorting by the Values of Multiple Variables

*Procedure features:*

PROC SORT statement option:
OUT=
BY statement
Other features:
   PROC PRINT

This example
  □ sorts the observations by the values of two variables
  □ creates an output data set for the sorted observations
  □ prints the results.

Program

options nodate pageno=1 linesize=80 pagesize=60;

The data set ACCOUNT contains the name of each business that owes money, the amount of
money that it owes on its account, the account number, and the town where the business is
located.

data account;
   input Company $ 1-22 Debt 25-30 AccountNumber 33-36
       Town $ 39-51;
datalines;
Paul’s Pizza          83.00  1019  Apex
World Wide Electronics 119.95  1122  Garner
Strickland Industries   657.22  1675  Morrisville
Ice Cream Delight     299.98  2310  Holly Springs
Watson Tabor Travel    37.95  3131  Apex
Boyd & Sons Accounting  312.49  4762  Garner
Bob’s Beds              119.95  4998  Morrisville
Tina’s Pet Shop        37.95  5108  Apex
Elway Piano and Organ    65.79  5217  Garner
Tim’s Burger Stand     119.95  6335  Holly Springs
Peter’s Auto Parts     65.79  7288  Apex
Deluxe Hardware       467.12  8941  Garner
Pauline’s Antiques     302.05  9112  Morrisville
Apex Catering           37.95  9923  Apex
;

OUT= creates a new data set for the sorted observations.

proc sort data=account out=bytown;

The BY statement first sorts the observations alphabetically by town, then by company.

   by town company;
run;
PROC PRINT prints the data set BYTOWN.

```sas
proc print data=bytown;
```

The VAR statement specifies the variables and their order in the output.

```sas
var company town debt accountnumber;
```

```sas
title 'Customers with Past-Due Accounts';
title2 'Listed Alphabetically within Town';
```

```sas
run;
```

### Output

**Customers with Past-Due Accounts**
**Listed Alphabetically within Town**

<table>
<thead>
<tr>
<th>Obs</th>
<th>Company</th>
<th>Town</th>
<th>Debt</th>
<th>Account Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Apex Catering</td>
<td>Apex</td>
<td>37.95</td>
<td>9923</td>
</tr>
<tr>
<td>2</td>
<td>Paul’s Pizza</td>
<td>Apex</td>
<td>83.00</td>
<td>1019</td>
</tr>
<tr>
<td>3</td>
<td>Peter’s Auto Parts</td>
<td>Apex</td>
<td>65.79</td>
<td>7288</td>
</tr>
<tr>
<td>4</td>
<td>Tina’s Pet Shop</td>
<td>Apex</td>
<td>37.95</td>
<td>5108</td>
</tr>
<tr>
<td>5</td>
<td>Watson Tabor Travel</td>
<td>Apex</td>
<td>37.95</td>
<td>3131</td>
</tr>
<tr>
<td>6</td>
<td>Boyd &amp; Sons Accounting</td>
<td>Garner</td>
<td>312.49</td>
<td>4762</td>
</tr>
<tr>
<td>7</td>
<td>Deluxe Hardware</td>
<td>Garner</td>
<td>467.12</td>
<td>8941</td>
</tr>
<tr>
<td>8</td>
<td>Elway Piano and Organ</td>
<td>Garner</td>
<td>65.79</td>
<td>5217</td>
</tr>
<tr>
<td>9</td>
<td>World Wide Electronics</td>
<td>Garner</td>
<td>119.95</td>
<td>1122</td>
</tr>
<tr>
<td>10</td>
<td>Ice Cream Delight</td>
<td>Holly Springs</td>
<td>299.98</td>
<td>2310</td>
</tr>
<tr>
<td>11</td>
<td>Tim’s Burger Stand</td>
<td>Holly Springs</td>
<td>119.95</td>
<td>6335</td>
</tr>
<tr>
<td>12</td>
<td>Bob’s Beds</td>
<td>Morrisville</td>
<td>119.95</td>
<td>4998</td>
</tr>
<tr>
<td>13</td>
<td>Pauline’s Antiques</td>
<td>Morrisville</td>
<td>302.05</td>
<td>9112</td>
</tr>
<tr>
<td>14</td>
<td>Strickland Industries</td>
<td>Morrisville</td>
<td>657.22</td>
<td>1675</td>
</tr>
</tbody>
</table>

### Example 2: Reversing the Order of the Sorted Values

**Procedure features:**

- **BY statement option:** DESCENDING

**Other features**

- PROC PRINT

**Data set:** ACCOUNT on page 1015

This example
- sorts the observations by the values of three variables
- reverses the sorting order for one of the variables
- prints the results.
Program

options nodate pageno=1 linesize=80 pagesize=60;

OUT= creates a new data set for the sorted observations.

proc sort data=account out=sorted;

The BY statement first sorts the observations alphabetically by town, then by descending values of amount owed, then by ascending values of the account number.

   by town descending debt accountnumber;
   run;

PROC PRINT prints the data set SORTED.

proc print data=sorted;

The VAR statement specifies the variables and their order in the output.

   var company town debt accountnumber;
   title 'Customers with Past-Due Accounts';
   title2 'Listed by Town, Amount, Account Number';
   run;

Output
Note that sorting last by AccountNumber puts the businesses in Apex with a debt of $37.95 in order of account number.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Company</th>
<th>Town</th>
<th>Debt</th>
<th>Account Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Paul’s Pizza</td>
<td>Apex</td>
<td>83.00</td>
<td>1019</td>
</tr>
<tr>
<td>2</td>
<td>Peter’s Auto Parts</td>
<td>Apex</td>
<td>65.79</td>
<td>7288</td>
</tr>
<tr>
<td>3</td>
<td>Watson Tabor Travel</td>
<td>Apex</td>
<td>37.95</td>
<td>3131</td>
</tr>
<tr>
<td>4</td>
<td>Tina’s Pet Shop</td>
<td>Apex</td>
<td>37.95</td>
<td>5108</td>
</tr>
<tr>
<td>5</td>
<td>Apex Catering</td>
<td>Apex</td>
<td>37.95</td>
<td>9923</td>
</tr>
<tr>
<td>6</td>
<td>Deluxe Hardware</td>
<td>Garner</td>
<td>467.12</td>
<td>8941</td>
</tr>
<tr>
<td>7</td>
<td>Boyd &amp; Sons Accounting</td>
<td>Garner</td>
<td>312.49</td>
<td>4762</td>
</tr>
<tr>
<td>8</td>
<td>World Wide Electronics</td>
<td>Garner</td>
<td>119.95</td>
<td>1122</td>
</tr>
<tr>
<td>9</td>
<td>Elway Piano and Organ</td>
<td>Garner</td>
<td>65.79</td>
<td>5217</td>
</tr>
<tr>
<td>10</td>
<td>Ice Cream Delight</td>
<td>Holly Springs</td>
<td>299.98</td>
<td>2310</td>
</tr>
<tr>
<td>11</td>
<td>Tim’s Burger Stand</td>
<td>Holly Springs</td>
<td>119.95</td>
<td>6335</td>
</tr>
<tr>
<td>12</td>
<td>Strickland Industries</td>
<td>Morrisville</td>
<td>657.22</td>
<td>1675</td>
</tr>
<tr>
<td>13</td>
<td>Pauline’s Antiques</td>
<td>Morrisville</td>
<td>302.05</td>
<td>9112</td>
</tr>
<tr>
<td>14</td>
<td>Bob’s Beds</td>
<td>Morrisville</td>
<td>119.95</td>
<td>4998</td>
</tr>
</tbody>
</table>

Example 3: Displaying the First Observation of Each BY Group

Procedure features:
- PROC SORT statement option:
  - NODUPKEY
- BY statement

Other features:
- PROC PRINT

Data set: ACCOUNT on page 1015

In this example, PROC SORT creates an output data set that contains only the first observation of each BY group. The NODUPKEY option removes an observation from the output data set when its BY value is identical to the previous observation’s BY value. The resulting report contains one observation for each town where the businesses are located.

Program

```plaintext
options nodate pageno=1 linesize=80 pagesize=60;

NODUPKEY writes only the first observation of each BY group to the new data set TOWNS.

proc sort data=account out=towns nodupkey;
```
The BY statement sorts the observations by town.

```plaintext
by town;
run;
```

PROC PRINT prints the data set TOWNS.

```plaintext
proc print data=towns;
```

The VAR statement specifies the variables and their order in the output.

```plaintext
var town company debt accountnumber;
title 'Towns of Customers with Past-Due Accounts';
run;
```

### Output

The output data set contains only four observations, one for each town in the input data set.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Town</th>
<th>Company</th>
<th>Debt</th>
<th>Account Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Apex</td>
<td>Paul’s Pizza</td>
<td>83.00</td>
<td>1019</td>
</tr>
<tr>
<td>2</td>
<td>Garner</td>
<td>World Wide Electronics</td>
<td>119.95</td>
<td>1122</td>
</tr>
<tr>
<td>3</td>
<td>Holly Springs</td>
<td>Ice Cream Delight</td>
<td>299.98</td>
<td>2310</td>
</tr>
<tr>
<td>4</td>
<td>Morrisville</td>
<td>Strickland Industries</td>
<td>657.22</td>
<td>1675</td>
</tr>
</tbody>
</table>

SAS® Procedures Guide, Version 8
Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of the software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227–19 Commercial Computer Software-Restricted Rights (June 1987).


1st printing, October 1999
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.

IBM® and DB2® are registered trademarks or trademarks of International Business Machines Corporation. ORACLE® is a registered trademark of Oracle Corporation. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

The Institute is a private company devoted to the support and further development of its software and related services.