Beginning SAS Programming

Yufeng Guo

10 9 8 7 6 5 4 3 2

First edition: June 2015

ISBN 978-1514218990

Printed in the United States of America

Contents

Preface

> *Its syntax is downright ugly, yet its approach is pure genius. Its scope is gigantic, yet knowing only 5% of it is probably sufficient for your job. It is likely the most expensive software package you will ever use, yet it is one of the most productive data management and analysis tools ever invented for business users. It is hardly the language you fall in love at first sight, yet it grows on you over time and may become a major tool for your career. It is SAS.*

Search `www.indeed.com` and you'll find that SAS programming skills are highly sought after in today's big data world. However, learning SAS isn't easy for complete beginners primarily for three reasons.

First, the archaic syntax of SAS may frustrate many modern programmers and business users who are used to programming languages like Java and Python. The SAS DATA step was modeled after PL/I (programming language one). The SAS macro facility was modeled after the macro preprocessor of PL/I. In fact, SAS was originally written mostly in PL/I and later ported to C. PL/I had its heyday in the 1960s and is on the verge of extinction. Some even say that it's been long dead.

Fortunately, the overall language design of SAS is coherent. If you can overcome the initial syntax shock of SAS, you'll open yourself up to its immense power.

Second, getting the SAS software was, up until recently, expensive for independent learners not associated with a college or an employer. Fortunately, the SAS Institute, the company that makes the SAS software, woke to the alarm that its future could be bleak if only the privileged few could learn SAS while the open source data analysis software R and Python are free to anyone. It decided to make the SAS software in the cloud free for everyone to learn SAS. Today, an independent learner who has an internet connection can login to the SAS server in the cloud and write his first SAS program in 5 minutes.

An ambitious learner can even download and install the free SAS University Edition in virtualbox (see `http://www.sas.com/en_us/software/university-edition/download-software.html`).

Finally, SAS is a huge language to learn and newcomers often drown in the giant ocean of SAS. SAS is a software package for a full data analysis cycle, from data creation, extraction, cleaning, and transformation to advanced data analytics and data presentation. Often an overambitious newcomer attempts to learn everything in SAS but soon gets discouraged after realizing that he'll need to spend his lifetime to gain full mastery of SAS.

Fortunately, there's a strategy to cope with the enormous scope of SAS. The strategy is called the 5% principle:

> *My life is short. I don't have time to learn everything in SAS. I just need to learn 5% of it. 5% is good enough. For the remaining 95%, my answer is that I don't know and I don't care. I can always Google when I need it.*

You can think of SAS as the Microsoft Office counterpart for data management and analysis. You don't try to master the full Microsoft Office suite: Word, Excel, Access, Outlook, OneNote, and Publisher. No one can master all the products in Microsoft Office. Rarely can one master everything even in one Office product. You start by learning one or more Office products that are most relevant to you, such as Word and Excel. Even if you focus on Word, most likely you just need to know 5% of its features that you use regularly and rely on a search engine such as Google or Bing to find the features you use rarely. For example, most people don't need to know how to create a table of contents or do a mail merge in Word.

The same is true for SAS. SAS was invented to solve virtually every major problem imaginable in the full data analysis cycle. No one has the time or need to master everything in SAS.

Which 5% of SAS to learn? A good starting point for anyone to learn is how to get the data. The full data analysis cycle can be roughly divided into two phases: getting the data (data retrieval, cleaning, and transformation) and making sense of the data. In a typical real-world project, getting the data is 80% of the work; analyzing the data is 20%.

Getting the data consumes most of the time on an analytics project. Knowing how to retrieve and manipulate data is beneficial even if your job is mostly making sense of the data. Even if someone else fetches the data for you, you almost always need to clean and reformat the data before you can analyze it.

However, even getting the data is too much for most people to master. One has to narrow his focus even further.

PROC SQL, the DATA step, and the macro facility are the cornerstones of SAS programming. They are the 5% of SAS programming every aspiring SAS programmer needs to learn. If you can master the fundamentals of these three areas, the immense power of SAS as a data extraction and manipulation tool will be at your disposal. With Google on your side to help you find details of the SAS language that are too numerous to remember, you'll be able to write complex SAS programs and solve challenging real-world problems.

This book focuses on PROC SQL, the DATA step, and the macro facility.

Origin of the Book

This is the book I needed when I was first learning SAS. None of the books on the market seemed to be right for a true novice like me. Most SAS books focused on statistical analysis with a sprinkling of SAS programming. Their coverage of SAS as a data extraction and data manipulation tool was insufficient for a newcomer to be up and running with SAS programming.

Though there were a handful of books on SAS programming, some of them didn't tell me much more than what I could find by reading the SAS documentation, while others were reference books for someone who already knew SAS programming fundamentals. In addition, almost all the SAS programming books threw too much stuff at me too fast.

In the end, I muddled through SAS by trial and error without using any books. I learned it the hard way.

While there were a plethora of books on Java, Python, or almost any other programming language that would bring a complete beginner to the advanced level, there was none for SAS. I wrote this book to fill the void. I hope it will teach you SAS programming from the ground up and transform you from a complete novice to an advanced programmer.

What You Get from This Book

This book is a fast paced tutorial on SAS data programming. It assumes that you know nothing about SAS and builds your knowledge from the ground up to the advanced level. It starts from a simple Hello World program and gradually moves onto advanced topics such as the macro facility.

The focus of this book is the SAS programming language, not the statistical analysis features of SAS.

One distinguishing feature of this book is that it teaches SAS by solving real-world problems.

Prerequisites

An absolute requirement for you to learn SAS from this book is that you have access to the SAS software. If you run PC SAS on your computer or SAS on a Unix server, you are good to go. If you installed the free SAS University Edition in virtualbox, you are good to go. Finally, if you have a computer and an internet connection, you are good to go; you can access SAS in the cloud for free via a web browser.

It will be easier for you to learn SAS if you have some programming experience in whatever software tools or computer languages you happen to use. For example, if you know how to use the `sumif` or `vlookup` function in Excel or if you can create a simple table in Access, you are ready to follow this book. If you have programmed in Visual Basic, Python, Javascript, or any other programming languages, you are ready to go.

If SAS is your first programming language, you can still follow this book, but you'll have to work much harder.

Target Readers

The intended audience for this book is beginning and intermediate SAS programmers. If you belong to one of those two groups, you'll find that this book answers many of the questions you have about SAS as a data extraction and manipulation tool.

A true beginner can quickly learn how to program in SAS by reading Chapters 1 – 6. An intermediate SAS programmer can enhance his knowledge by reading the remainder of the book, which covers advanced topics including the macro facility, combining tables, and regular expressions.

If you are an expert SAS programmer, you might learn something here and there, but most of the content should be familiar to you.

Advice for a True Beginner

First and foremost, be patient with yourself. Unless you are a PL/I veteran, SAS syntax will be new to you. It will take some time for you to get comfortable with the syntax, the vocabulary, and how things are done in SAS.

Second, when you learn SAS, focus on learning enough to solve one problem a time. Don't try to learn many things all at once. If you try to understand everything before solving anything, you'll get so sidetracked and overwhelmed that you want to quit.

Third, though you may fall in love with the DATA step, learn relational database basics and SQL. SAS is often used to extract and analyze large tables from a third party relational database. Virtually every complex real-world project requires a programmer to write SQL.

Finally, write a lot of SAS code. Some say that you are not an experienced driver unless you have driven more than $100,000$ miles. Though no one can foretell how many SAS programs you need to write to reach the advanced level, you clearly have a long way to go if you have written only a handful of programs. Besides, even advanced programmers need to keep up with the major changes introduced by each new release of SAS.

Why Chapter 1 is PROC SQL, not the DATA Step

Most SAS books start from the DATA step. However, this book teaches PROC SQL first.

The SAS DATA step is nothing short of a miracle of ingenious software design. It avoids the need to write complex SQL and allows a user to create and manipulate a table with simple procedural syntax. Want to drop Column `X` from a table? Just write `drop=X`. Want to rename Column `X` to `Y`? Just write `rename=(X=Y)`. Want to add a `tax` column which is 3% of the `sales` column? Just write `tax=0.03*sales`. Want to create ten character columns `var1` to `var10` each with a column width of 8? Just write `length var1-var10 $8`. Tired of writing CASE statements in SQL? Just write IF-THEN-ELSE in a DATA step.

As the popular saying goes, with great power comes great responsibility. To use the power of the DATA step, one must take on the responsibility of learning relational database fundamentals. SAS is a database engine for manipulating tables, not a procedural language for manipulating variables.

For example, a novice SAS programmer tends to create more variables in a DATA step than is truly necessary. Creating a variable incurs little overhead in a procedural language; a variable is just a memory cell to store a value. An extraneous variable in a procedural language is not the best thing to do but the harm is minimal.

However, an extraneous variable in a SAS DATA step is far more costly because a variable introduced in a DATA step automatically becomes a column of the output table unless it is dropped in the end. If your input table has ten million rows, then the extraneous variable `x=1` will cause the output table to have a column `x` filled with ten million rows of the constant value 1. This increases your table size and slows down your program execution, not to mention how confused your users will be when they see a column filled with row after row of the same value.

Here's another example where the ease of the DATA step can lead an inexperienced programmer astray. Newcomers tend to make multiple trips to a large table when only one pass to the table is truly needed to perform all the operations. When this happens, operations on a table that should take only thirty minutes can take three hours.

The SAS DATA step is a shortcut to the traditional SQL (structured query language) approach. As multiplication is a shortcut to addition and one must learn addition before learning multiplication, one needs to learn how to create and manipulate a table the hard way before learning the easy way.

How to Best Use This Book

This book alone cannot make you an advanced SAS programmer. You have to do your part. The best thing you can do is to write a lot of code. You won't become an advanced coder if you speed read this book without trying my programs in SAS.

Though I posted all the program code on my website for you to download, you'll learn most by resisting the temptation to copy and paste the code to your SAS code editor. Instead, manually type each line of code. Create your own prime number generator in SAS. If you are an expert in another language such as Python, rewrite your favorite Python programs in SAS.

In addition, recreate at least some, if not all, of the programs in the *Further Reading* section at the end of each chapter.

Adjust the File Path for SAS in the Cloud

Most programs in this book work in both PC SAS and SAS in the cloud out of the box. However, if a program reads from or writes to an external file, you'll need to adjust the SAS code that references the external file.

Some SAS programs in this book read in an external text file such as `abc.txt`. Other programs export an existing SAS table to a CSV file such as `xyz.csv`. In my PC SAS code, I assume that both `abc.txt` and `xyz.csv` are stored in the `C:\LearnSAS` folder. My SAS code will reference these two files as:

- `C:\LearnSAS\abc.txt`
- `C:\LearnSAS\xyz.csv`

In my code for SAS in the cloud, however, I assume that both `abc.txt` and `xyz.csv` are stored in the `/home/userid/myProgram` folder, where `userid` should be replaced by your user ID for SAS in the cloud. My SAS code will reference these two files as:

- `/home/userid/myProgram/abc.txt`
- `/home/userid/myProgram/xyz.csv`

To follow this book, create the `C:\LearnSAS` folder if you use PC SAS.

If you use SAS in the cloud, create the `myProgram` subfolder under `My Folders`. The full path of `My Folders` is `/home/userid`, where `userid` is your ID for SAS in the cloud. To find the full path of

`My Folders`, right click `My Folders` and select "Properties." Then the "Folder Properties" window will open up. The "Location" property is `/home/userid`. `My Folders` is the home directory of SAS in the cloud. The full path of the `myProgram` subfolder is `/home/userid/myProgram`.

To avoid repetition, I may show you only the code for PC SAS. If a program doesn't reference any files or folders in the operating system, the PC SAS program will also work for SAS in the cloud. However, if a program references a file path in the operating system, you'll need to adjust the file path so the operating system can recognize the file path.

For example, if you need to read in `C:\LearnSAS\sales.txt` into SAS, first create `sales.txt` in the `C:\LearnSAS` folder. Then you can reference the text file in a PC SAS program via an `infile` statement:

```
infile 'C:\LearnSAS\sales.txt';
```

If you use SAS in the cloud, upload `sales.txt` from your computer to the `myProgram` folder in the remote SAS Unix server in the cloud. Next, adjust the file path so the `infile` statement becomes:

```
infile '/home/userid/myProgram/sales.txt';
```

Here's another example. This is how to export the `work.employee` table to the `employee.csv` file in the `C:\LearnSAS` folder using PC SAS:

```
proc export
   data=work.employee
   outfile='C:\LearnSAS\employee.csv'
   dbms=csv
   replace;
run;
```

This is how to export the `work.employee` table to the `employee.csv` file in the `myProgram` folder in SAS in the cloud:

```
proc export
   data=work.employee
   outfile='/home/userid/myProgram/employee.csv'
   dbms=csv
   replace;
run;
```

Unlike in Windows, in SAS Unix servers in the cloud, folder names and file names are case sensitive. In addition, while both forward slashes and backslashes can be used in a file path in Windows, only forward slashes can be used in a file path in SAS in the cloud (see Section 1.6.3).

About Me

I grew up in China and immigrated to the United States in 1996. I received a master's degree in accounting and business information systems from Indiana University. After graduation, I worked in the IT department of an insurance company for two years and then became an actuary. I have been programming in SAS since 2009.

Where to Download the SAS Program Code in This Book

All the SAS programs in this book can be downloaded from my website:

`saszero2hero.com`

Submit or View Errata

If you find typos or programming errors, please email me at `yufeng.guo.actuary@gmail.com`.

The errata will be posted at my website:

`saszero2hero.com`

Acknowledgements

I couldn't have written this book without the support of my family. My wife Tonya helped me cut out dead wood (including some cute Chinese proverbs) in the preface. My oldest son Abraham proofread each chapter and helped set up the support site `http://saszero2hero.com/`. My second oldest son Benjamin designed the book cover.

I would like to thank the SAS Institute for making the SAS software in the cloud and the SAS University Edition free for anyone to learn SAS.

Finally, I would like to thank many SAS users who published papers to share their programming techniques. I have benefited greatly by reading papers by other SAS programmers.

Chapter 1

Hello World and PROC SQL Basics

In this chapter, you'll learn how to create a table using PROC SQL. You'll begin with a trivial Hello World example and move on to more serious programs.

1.1 Where to Get the SAS Software

If you are reading this book, most likely you already have SAS installed on your computer perhaps through your employer or your school.

If you don't have SAS installed in your machine, you can access the SAS software in the cloud via a web browser for free. You don't need to install anything special in your machine. As long as you have an internet connection, you can access SAS in the cloud, write and submit the code, and see the result.

The SAS Institute offers SAS in the cloud for anyone to learn SAS. Go to the SAS OnDemand for Academics Control Center at `https://odamid.oda.sas.com/SASODAControlCenter/`. Create an account and you can access the SAS server in the cloud right away.

This book uses SAS OnDemand and SAS in the cloud interchangeably.

Here's one major advantage of SAS in the cloud over SAS on your computer. SAS in the cloud not only uses the latest version of Base SAS, it also includes many other useful packages such as SAS/ACCESS Interface to PC Files, SAS/STAT, and SAS/IML, to name a few. In contrast, the SAS software installed in your machine may be several versions behind the most current version; it may not have SAS/ACCESS Interface to PC Files or other packages.

Since this book focuses on SAS programming, the most relevant package included in SAS in the cloud is SAS/ACCESS Interface to PC Files. The most common use of this interface is reading Excel files into SAS or exporting SAS tables into Excel files.

SAS in the cloud is slightly slower than SAS installed in your machine. Besides, SAS in the cloud may be down temporarily for maintenance.

1.2 SAS Windowing Environment

SAS runs on various operating systems including Windows, Unix, Mac dual boot, and mainframe. On each operating system, SAS lets a user, through a windowing environment, write and submit code, check the program log and output, and open the data tables generated by the SAS program. By the way, the windowing environment should be not confused with the Windows operating system.

This book uses PC SAS (e.g. SAS in Windows) and SAS in the cloud (e.g. SAS in Unix servers) to teach you how to program in SAS.

The SAS Institute published SAS companion guides for various operating systems. Search the web and you should find the following online documentation or pdf files:

- *SAS Companion for Windows*
- *SAS Companion for Unix*
- *SAS Companion for z/OS*

These companion guides are mostly reference books for advanced users. However, some content in Chapter 1 of these guides is friendly to new SAS users.

If you use SAS in the cloud, please Google the *SAS Studio User's Guide*. SAS Studio is the windowing environment for SAS in the cloud.

1.3 PC SAS Main Windows

This section is for PC SAS. Skip this section if you use SAS in the cloud.

Launch your SAS software by selecting Start » All Programs » SAS » SAS x, where x is your SAS version number. SAS will open 5 windows:

- `Editor - Untitled1`, the enhanced editor window. The enhanced editor window typically has a running man icon with a plus (+) sign. The plus sign means "enhanced" (enhanced editor). Your cursor should be blinking in this window after you launch SAS. This is where you type your SAS code.
- `Log - (Untitled)`, the log window. The log window generally sits on the top of the enhanced editor. After you launch SAS, the log window will just display copyright information. If you write any code in the enhanced editor and submit the code, SAS will provide feedback in the log window, telling you whether your code has executed successfully or has failed.
- `Output- (Untitled)` window. When you initially launch SAS, your `output` window will be blank. However, if you submit any SAS code and your SAS code generates any output, the `output` window will have some content.
- `Explorer` window, sitting in the left of the SAS window pane. In the explorer window, you should see a file cabinet icon called `Libraries`. Double click `Libraries` and you'll see several items (libraries) such as `Sashelp`, `Maps`, `Sasuser`, and `Work`. We are mostly interested in the `Work` library. By default, the `Explorer` window is selected.

- `Results` window, sitting to the far left of the SAS window pane. When you initially launch SAS, this window won't have any results yet. After you execute your SAS program, if your SAS program generates any outputs, the `Results` window will show clickable links to the outputs. By default, the `Results` window is not selected.

If by any chance you don't see any of these windows, you can always go to the View menu and select any of these five windows.

1.4 Configure PC SAS Windows

This section is for PC SAS. Skip this section if you use SAS in the cloud.

1.4.1 Show Line Numbers in the Enhanced Editor

While your cursor is still blinking in the enhanced editor window, do the following:

- select Tools > Options > Enhanced Editor
- check "Show line numbers"

1.4.2 Configure PC SAS 9.3+

SAS 9.3 or above, among other things, defaults the output to HTML instead of plain text. While this change might be welcomed by experienced SAS users, most new learners will find that the plain text output is less overwhelming. If you use SAS 9.3 or higher, you can switch the default output to plain text:

- select Tools > Options > Preferences > Results
- check Create listing
- uncheck Create HTML
- uncheck Use ODS Graphics

For more details, see SAS documentation at `http://support.sas.com/documentation/cdl/en/odsug/65308/HTML/default/viewer.htm#p0xidv7ssklgg2n1lzf7u6wuei7z.htm`.

1.5 First Program - Hello World with SQL

SAS is for analyzing tables. A table is like an Excel spreadsheet, where columns represent data names and where rows represent data values. Here's an example of a sales table with 5 columns and 2 rows:

```
CustomerID   LastName  FirstName   OrderDate  OrderAmt
   1         Smith     John        3/12/2014    287.45
   2         Johnson   Mary        9/22/2014    135.74
```

SAS has another set of vocabulary to describe a table. A table is a data set. A column of a table is a variable. And a row of a table is an observation.

Our first program will create a table.

1.5.1 Create helloWorld.sas in SAS In the Cloud

If you use SAS in the cloud, please watch the official video tutorial at `http://support.sas.com/training/tutorial/studio/index.html` on how to write a basic SAS program in SAS Studio.

Let's create a Hello World program in SAS in the cloud. After you login to the SAS server in the cloud, in the left pane of SAS Studio, right click "My Folders." Choose New > Folder to create a new folder and name it `myProgram`. You can use the `myProgram` folder to store all your SAS programs.

Press F4 to open the SAS code editor window. Type the following program and save it as `helloWorld.sas` in the `myProgram` folder:

Program 1.5.1 *helloWorld.sas*

```
proc sql;
create table work.MyNum
(
   N num
);
quit;
```

Make sure you spell `sql` correctly. The last letter is `l` as in `light`, not the number one. Next, submit the program by pressing F3 or clicking the running man icon. The log window will open up displaying something like this:

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
57
58         proc sql;
59         create table work.MyNum
60         (
61          N num
62         );
NOTE: Table WORK.MYNUM created, with 0 rows and 1 columns.
63         quit;
NOTE: PROCEDURE SQL used (Total process time):
   real time           0.00 seconds
   user cpu time       0.00 seconds
   system cpu time     0.00 seconds
```

The main message in the log is `NOTE: Table WORK.MYNUM created, with 0 rows and 1 columns`. This warns you that the `work.MyNum` table is blank with no data in it (because we haven't put any data in it). Nothing to worry about.

To see the newly created `work.MyNum` table, in the left pane of SAS Studio, click the `Libraries`. This opens up a new window. Click the triangle icon to the left of `My Libraries` to open up all the sub-libraries under `My Libraries`. You should see `WORK` as one sub-library. Under `WORK`, you should see the `MYNUM` table. Double click to open the `MYNUM` table. Since the `MYNUM` table is blank, you just see the column header `N` without any data.

Congratulations! You've successfully created a blank `work.MyNum` table.

1.5.2 Create helloWorld.sas in PC SAS

In your SAS enhanced editor window, type the same `helloWorld.sas` program.

Program 1.5.2 *helloWorld.sas*

```
proc sql;
create table work.MyNum
(
   N num
);
quit;
```

How to submit a program To submit the code you just typed in the enhanced editor, issue the `submit` command in one of the following ways:

- Press the F8 key while your enhanced editor window is active
- Select Run > Submit
- Click the running man icon
- Enter submit in the command bar. The command bar is at the top left corner of the SAS window, below the `File` menu. The command bar has a check mark to its left.

Submit the `helloWorld.sas` by one of the four ways above.

What you should see Your `Log` window should display copyright information followed by feedbacks to your submitted program:

```
NOTE: Copyright (c) yyyy-yyyy by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Version x
      Licensed to xyz, Site xxxxxxxx.
NOTE: This session is executing on the WIN_PRO  platform.



NOTE: SAS initialization used:
      real time           0.01 seconds
      cpu time            0.01 seconds

1    proc sql;
2    create table work.MyNum
3    (
4       N num
5    );
```

```
NOTE: Table WORK.MYNUM created, with 0 rows and 1 columns.
6    quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

The SAS log records everything you did in your SAS session or with your SAS program. Your original program statements are identified by line numbers. Mixed with SAS statements are messages generated by SAS. These messages typically begin with the words NOTE, INFO, WARNING, ERROR, or an error number, and they may refer to a SAS statement by its line number in the log.

When you submit Program 1.5.2 the first time, the line number in the log starts from 1.

However, if you submit Program 1.5.2 the second time, a new log will be added to the end of the previous log. In the new log, the starting line number will be the ending line number of the previous log plus 1.

This is the main message in the log:

```
NOTE: Table WORK.MYNUM created, with 0 rows and 1 columns.
```

The `work.MyNum` table is blank with no data in it.

Next, let's see the `work.MyNum` table. Select View > Explorer to open the explorer window. You should see the `Work` folder displayed in either side of the explorer window. Click to open the `Work` folder on either side of the window. You should see the `MyNum` table. Click to open `MyNum` table. A message box pops up saying:

`NOTE: Data set has 0 observations.`

This message warns you again that the `work.MyNum` table is blank.

Click OK to exit the pop up window. Then you should see a blank table with a column header `N`.

Congratulations! You've successfully built your first SAS program and produced a blank table.

1.6 Understand helloWorld.sas

Program 1.5.2 can be shortened to three lines:

```
proc sql;
    create table work.MyNum (N num);
quit;
```

This is a complete, fully working SAS program. It generates a blank table `work.MyNum` with a numeric column `N`. This table sits in the `work` folder.

SAS uses a two-level name such as `work.MyNum` to unambiguously identify a table. The first level `work` identifies the folder that contains the table. The second level `MyNum` is the name of the table.

If you omit the first level name, SAS will use `WORK` as the default first-level name. The statement `create table MyNum (N num)` is the same as `create table work.MyNum (N num)`.

In Section 1.8 you'll learn how to create a different first level name than `work`.

Statements This program has three statements. Each statement ends with a semicolon.

The first statement invokes the SQL procedure. Many pre-built procedures are shipped with Base SAS to enable users to perform specific tasks. For example, you can use the SORT procedure to sort a table by one or more columns, in ascending or descending order (see Program 9.1.1).

`SQL` stands for structured query language, a special-purpose programming language designed for managing data held in a relational database management system (RDBMS). `Proc sql` is a pre-built procedure shipped with Base SAS. Its main functionality is `CRUD`, that is, to create, read, update, or delete a table.

The second statement `create table work.MyNum (N num)` creates a new table that has one numeric column `N`. The word `num` is short for `numeric`.

Finally, the `quit` statement terminates the SQL procedure.

1.6.1 How SAS Is Different from Other RDBMS

It's debatable whether SAS should be called a RDBMS. Though SAS has PROC SQL, the SAS database engine is different from most relational databases. Most RDBMS engines are designed for transaction processing and allow multiple users to have concurrent access to data. As such, most RDBMS engines have record locking to ensure data integrity when multiple users are attempting to read or update the same row simultaneously. In addition, most RDBMS engines have transaction rollback to return a database to its previous state.

However, the SAS database engine is designed for data analysis. In many companies, copies of transactional data were made solely for data analysis so data analysis won't interfere with real time transaction processing. And a copy of transactional table can be exclusively used by a single user while he is reading the data. After the user finishes reading the data, the data sits idle until the next user reads it and has the exclusive access to it. Then the second user releases the data after he finishes reading it. Hence the SAS database engine doesn't have to worry about maintaining data integrity due to concurrent access and can sequentially read large amounts of data quickly.

Here's another difference between SAS and other databases. While a table in a typical relational database is a collection of unordered rows, a SAS table is a collection of ordered rows in a SAS DATA step program (the DATA step is explained in Chapter 2). If you use a DATA step to manipulate a table, the order of the rows in the output table is the same as the order of the rows in the input table. To change the order of the rows of a table, you must purposely reorder the rows using PROC SORT.

However, similar to other database engines, SAS does not guarantee the order of the rows in the output of PROC SQL. If you use PROC SQL to query a SAS table, the output of the query is not guaranteed unless you specify a sort order using the ORDER BY clause.

The third major difference between SAS and other databases is how null is handled. This becomes an issue when you use SAS to retrieve a table from a third party database system such as DB2, Oracle, and Microsoft SQL server. For more information, see `http://support.sas.com/kb/23/225.html`. If you are new to SAS, skip this topic for now and come back to it later.

1.6.2 SAS Case Sensitivity

In PC SAS and SAS in the cloud, the log generated by `helloWorld.sas` calls the `WORK.MYNUM` table, but the table is spelled `work.MyNum` in the code. SAS is case insensitive. `WORK.MYNUM` is the same as `work.MyNum`. The `N` column is the same as the `n` column. Program 1.5.2 can be rewritten in uppercase or in the mixture of uppercase and lowercase:

```
PROC SQL;
CREATE TABLE WORK.MYNUM
(
   N NUM
);
QUIT;
```

```
PROC SQL;
CREATE TABLE work.MyNum
(
   N num
);
QUIT;
```

However, quoted strings or literal strings are always case sensitive in SAS. 'US', 'Us', 'uS', and 'us' are four different literal strings. If your table has a country column and the United States is entered as any of the above four quoted strings, then to do a case insensitive comparison, you can write:

```
if upcase(country)='US' then ... ;
```

The `upcase` function converts all letters in an argument to uppercase.

1.6.3 Case Sensitivity: SAS in the Cloud Versus PC SAS

Though SAS is case insensitive, your operation system can be case sensitive or case insensitive.

Windows is case insensitive in folder names and file names. For example, `C:\LearnSAS` is the same as `C:\learnsas`. If you already have the `C:\LearnSAS` folder and you want to create another folder named `C:\learnsas`, your Windows computer will warn you that a folder with the same name already exists and asks you whether you want to replace the existing folder.

Similarly, in Windows, you can't create `abc.txt` and `Abc.txt` in the same folder because they refer to the same file.

The following three statements are equivalent in PC SAS. Each statement creates the same file nickname `myfile` pointing to the same external file `abc.txt` in the `C:\LearnSAS` folder:

```
filename myFile 'C:\LearnSAS\abc.txt';
filename myfile 'C:\learnSAS\Abc.txt';
filename Myfile 'C:/learnsas/ABC.txt';
```

Because SAS is case insensitive, `myFile`, `myfile`, and `Myfile` are equivalent. Because Windows is case insensitive, the three full file paths are equivalent.

By the way, in Windows, a backslash and a forward slash are interchangeable as a file path separator.

Unlike Windows, the Unix operating system is case sensitive. Since SAS in the cloud is hosted in Unix servers, a file path and a file name are case sensitive in SAS in the cloud. For example, in the `/home/userid` folder in SAS in the cloud (where `userid` should be replaced by your user ID for SAS in the cloud), you can create two subfolders, one named `myProgram` and the other `myprogram`. Similarly, you can create two text files, `abc.txt` and `Abc.txt` in the same folder in SAS in the cloud.

In addition, Unix uses a forward slash as the path separator.

If you use SAS in the cloud, when you need to reference a folder name or file path, the folder name and the file path you specify are case sensitive. For example, in SAS in the cloud, the following three statements are equivalent:

```
filename myFile '/home/userid/sasuser.v94/abc.txt';
filename MyFile '/home/userid/sasuser.v94/abc.txt';
filename myfile '/home/userid/sasuser.v94/abc.txt';
```

The file nicknames, `myFile`, `MyFile`, and `myfile` are equivalent because SAS is case insensitive.

However, in SAS in the cloud, `/home/userid/sasuser.v94/abc.txt` cannot be specified, for example, as `/home/userid/sasuser.V94/abc.txt`.

Since a forward slash as a path separator works in PC SAS and SAS in the cloud, if you use a forward slash in your file path, your SAS program will work in PC SAS and SAS in the cloud.

1.7 Second Program - Populate a Table with SQL

The next program works in both PC SAS and SAS in the cloud.

Program 1.7.1 *one2tenWork.sas*

```
1  /* clear previous log and output */
2  dm log 'clear';
3  dm output 'clear';
4
5  /******************************************************/
6  /* create and populate MyNum (only one numeric column N) */
7  /******************************************************/
8  proc sql;
9
10 * Define table structure ;
11 create table work.MyNum
12 (
13    N num
14 )
15 ;
16
17 * Populate the blank table with 1 to 10 ;
18 insert into work.MyNum
19 values ( 1 )
20 values ( 2 )
21 values ( 3 )
22 values ( 4 )
23 values ( 5 )
24 values ( 6 )
25 values ( 7 )
26 values ( 8 )
27 values ( 9 )
28 values ( 10 )
29 ;
30
31 *Display work.MyNum in Output window unless noprint is on;
32 select * from work.MyNum;
33
34 ************************************************************ ;
35 * When done, always use quit to exit sql procedure            ;
36 * Otherwise proc sql keeps running, wasting computer resources ;
37 *************************************************************;
38 quit;
```

Output:

```
       N
--------
       1
       2
       3
       4
       5
       6
       7
       8
       9
      10
```

Submit the program. A new `work.MyNum` table will be created with integers from 1 to 10. You'll get the following log:

```
NOTE: 10 rows were inserted into WORK.MYNUM.
```

Open the `work.MyNum` table. It has ten rows populated with 1 to 10 respectively.

What just happened

Comment Line 1 is a comment. SAS has 2 comment styles:

```
/* one line or multi-line comment */
*  one line or multi-line comment  ;
```

The first style is widely used in many programming languages. The second style, however, appears to be specific to SAS.

Under the second style, a comment starts from the star sign `*`, can run many lines, and ends with a semicolon.

This program has two pretty boxes (Lines 5–7 and Lines 34–37). A pretty box is just disguised comments. The first pretty box is several one-line comments using the first comment style; the second is several one-line comments using the second comment style.

DM statements The `dm` statements (Lines 2 and 3) are for PC SAS. If you use SAS in the cloud, you don't need them, though adding them won't cause any harm.

These two lines instruct the SAS display manager to clear the `log` window and the `output` window before executing the rest of the code.

Every time when you submit a SAS program via any of the four ways described in Section 1.5.2, a log is written to the log window indicating whether the SAS program has executed successfully. And the output, if any, is written to the output window.

Without the DM statements, the logs and the outputs generated by all the previously submitted PC SAS programs will stay in the `log` window and in the `output` window respectively. And the new log and the new output generated by your current program will be appended to the previous logs and to the previous outputs, making it hard for you to see your current log and current output.

The two DM statements erase any previously created logs and outputs. As a result, only the log and the output generated by the newly submitted SAS program will appear in the `log` window and the `output` window.

If you are using SAS 9.3+, you'll want to configure it according to Section 1.4.2. Otherwise, the `dm output 'clear'` statement won't work.

INSERT This is the main code. The `insert` statement (Lines 18–29) adds new rows to a table. The `values` clause lists the values to be inserted into each new row. Each data type in the `values` clause must match the data type of the corresponding column.

SELECT The `select * from work.MyNum` statement (Line 32) retrieves all the columns from the source table `work.MyNum`. The `from` clause indicates the source table. An asterisk `*` represents all the columns of the source table.

`work.MyNum` has only one column `N`. Hence `select * from work.MyNum` can be replaced by `select N from work.MyNum`.

By default, all the retrieved columns in the `select` statement will be displayed in the `output` window. However, you can prevent Line 32 from writing the query result to the `output` window by specifying the `noprint` option. Change Line 8 into

```
proc sql noprint;
```

Resubmit the program. You won't see the 10 rows of integers from the `work.MyNum` displayed in the `output` window.

Finally, the ending `quit` statement terminates the SQL procedure.

1.8 Third Program - Store a Table in a Permanent Folder

1.8.1 PC SAS

Now close the SAS software by clicking the top right X. Re-launch SAS from the `Start` menu. Select View › Explorer to open the SAS explorer window. Open the `Work` folder on either side of the window. What's in the `Work` folder? Nothing. The `MyNum` table generated by Program 1.7.1 is gone.

The `Work` folder is a temporary storage of your tables. If you close SAS, SAS will automatically delete all the files stored in the `Work` folder.

The next program stores `MyNum` table in a permanent folder `C:\LearnSAS`. Create the folder `C:\LearnSAS` before running Program 1.8.1.

Program 1.8.1 *one2tenPermFolder.sas*

```
/************************************************/
/* Store MyNum table permanntly in C:\LearnSAS  */
/* First, create C:\LearnSAS                    */
/* This program works in SAS 9.0 and above      */
/************************************************/
dm log 'clear'; dm output 'clear';

proc sql;
create table 'C:\LearnSAS\MyNum'
(
   N num
)
;

insert into 'C:\LearnSAS\MyNum'
values ( 1 )
values ( 2 )
values ( 3 )
values ( 4 )
values ( 5 )
values ( 6 )
values ( 7 )
values ( 8 )
values ( 9 )
values ( 10 )
;

select * from 'C:\LearnSAS\MyNum';
quit;
```

Output:

```
       N
--------
       1
       2
       3
       4
       5
       6
       7
       8
       9
      10
```

Submit the program. If you are using SAS version 9, this program will run without errors. The `output` window will display integers 1 to 10.

Check your `C:\LearnSAS` folder. You should see the `mynum.sas7bdat` table. The `.sas7bdat` is the file extension of a SAS table. Click to open the `mynum` table and you should see ten rows filled with integers 1 to 10 respectively.

In this program, we identify the `MyNum` table by its full path and name, `C:\LearnSAS\MyNum`. As a result, SAS will create the `MyNum` table in the `C:\LearnSAS` folder. Now if you exit SAS and re-launch it from the START menu, the `MyNum` table remains in the `C:\LearnSAS` folder.

What if later you want to change the folder from `C:\LearnSAS` to another folder? You'll need to replace each occurrence of `C:\LearnSAS` in Program 1.8.1 with the new folder name. Sure you can do a find-and-replace in the enhanced editor by selecting Edit › Replace, but there is a better way.

Program 1.8.2 *one2tenLibname.sas*

```
dm log 'clear'; dm output 'clear';

/* create C:\LearnSAS first before you nickname it as mylib */
/* works in all SAS versions */
libname mylib 'C:\LearnSAS';

proc sql;
create table mylib.MyNum
(
   N num
)
;

insert into mylib.MyNum
values ( 1 )
values ( 2 )
values ( 3 )
values ( 4 )
values ( 5 )
values ( 6 )
values ( 7 )
values ( 8 )
values ( 9 )
values ( 10 )
;

select * from mylib.MyNum;
quit;
```

Output:

```
       N
--------
       1
       2
       3
       4
       5
       6
       7
       8
       9
      10
```

Program 1.8.2 works for all SAS versions. Next, highlight the `libname` statement:

```
libname mylib 'C:\LearnSAS';
```

Submit Program 1.8.2. You should see the following log (Vx is your SAS version number):

```
NOTE: Libref MYLIB was successfully assigned as follows:
      Engine:        Vx
      Physical Name: C:\LearnSAS
```

SAS creates a folder nickname `mylib` that points to the actual folder `C:\LearnSAS`. Once `mylib` is created, `C:\LearnSAS\mytable` can be referenced by a short name `mylib.mytable`.

Now unhighlight the `libname` statement and submit Program 1.8.2. Go to `C:\LearnSAS`. You should see the `MyNum` table.

If later you decide to use a different folder, just update the `libname` statement and point `mylib` to the new folder.

For simplicity, I nicknamed `C:\LearnSAS` as `mylib`. You can use a different nickname such as `folder` or `dir`. Make sure that the nickname is no more than 8 character long. In addition, the folder nickname can not be a SAS reserved word like `work`, `user`, or `sashelp`.

1.8.2 SAS in the Cloud

Similar to in PC SAS, in SAS in the cloud, the `work` folder is a temporary storage. If you exit SAS in the cloud by clicking “Sign Out” at top right of SAS Studio and re-login to SAS in the cloud, the `work` folder will be empty. All the tables previously stored in the `work` folder were deleted.

In SAS in the cloud, as in PC SAS, we can use the `libname` statement to create a permanent folder to store our SAS tables. We’ll use the `myProgram` folder created earlier as the permanent location to store our tables. Create the following program in SAS in the cloud:

Program 1.8.3 *one2tenLibnameCloud.sas*

```
libname mylib '/home/userid/myProgram';
proc sql;
create table mylib.MyNum
(
   N num
)
;

insert into mylib.MyNum
values ( 1 )
values ( 2 )
values ( 3 )
values ( 4 )
values ( 5 )
values ( 6 )
values ( 7 )
values ( 8 )
values ( 9 )
values ( 10 )
;

select * from mylib.MyNum;
quit;
```

Output:

```
       N
--------
       1
       2
       3
       4
       5
       6
       7
       8
       9
      10
```

In `'/home/userid/myProgram'`, replace `userid` with your user ID for SAS in the cloud.

You can save Program 1.8.3 in the `myProgram` folder. Press F3 to submit the program. Now under the `myProgram` folder, you should see the newly created `MyNum` table.

Here’s another way to view the newly created `MyNum` table. Click to open the `Libraries`. You should see the `MYLIB` folder. Under the `MYLIB` folder, you should see the `MYNUM` table.

1.8.3 WORK Pointing to Which Physical Folder?

`Work` is also a nickname pointing to some temporary folder in your hard drive. To find out which folder `work` points to, after you submit Program 1.7.1 and the table `work.MyNum` is created, submit the next program:

Program 1.8.4 *one2tenWork2.sas*

```
/*show content of the table; prints library and physical path*/
proc contents data=work.MyNum;
run;
```

Your output should be similar to the following no matter you submit the program in PC SAS or SAS in the cloud:

```
                The CONTENTS Procedure

Data Set Name: WORK.MYNUM         Observations:         10
Member Type:   DATA               Variables:            1
Engine:        Vx                 Indexes:              0
Created:       date time          Observation Length:   8
Last Modified: date time          Deleted Observations: 0
Protection:                       Compressed:           NO
Data Set Type:                    Sorted:               NO
Label:


-----Engine/Host Dependent Information-----

Data Set Page Size:         4096
Number of Data Set Pages:   1
First Data Page:            1
Max Obs per Page:           501
Obs in First Data Page:     10
Number of Data Set Repairs: 0
File Name:                  path\_TDxxxx\mynum.sas7bdat


-----Alphabetic List of Variables and Attributes-----

#    Variable    Type    Len    Pos
1    N           Num       8      0
```

Under the `Engine/Host Dependent Information` section, the `File Name` tells you where the `MyNum` table is. If you use PC SAS, the `File Name` may contain `_TDxxxx`. `TD` stands for temporary directory and `xxxx` is a combination of random integers and characters generated by SAS. `_TDxxxx` is the temporary directory that contains the `MyNum` table. The `work` library points to the `path\_TDxxxx` folder. Open the `path\_TDxxxx` folder and you should see the `MyNum` table.

If you use SAS in the cloud, your `File Name` may look like this:

`/saswork/.../.../mynum.sas7bdat`.

Unfortunately, you won't be able to access the folder that contains the `MyNum` table:

`/saswork/.../.../`.

However, you can still access the `MyNum` table by opening the `work` library in SAS Studio.

Let's experiment with PC SAS. Go to the `path` folder, which is the parent folder of the `_TDxxxx` folder. Next, close the SAS software and you'll witness that the `_TDxxxx` temporary folder disappearing right in front of your eyes. Now re-launch SAS from the `Start` menu and you'll see that another temporary folder `_TDyyyy` appears in the `path` folder. Every time you launch SAS, SAS creates a temporary folder whose name begins with `_TD`. This temporary folder is deleted when you close SAS.

Open the _TDyyyy folder. The folder contains some utility files but it doesn't have the MyNum table yet. Now submit Program 1.8.4 and you'll see that more files are added to the _TDyyyy folder. One of the files added is the MyNum table.

The MyNum table you see in the work folder is not the physical table, but a pointer to the physical MyNum table stored in the _TDyyyy folder. That's why the MyNum table in the work folder doesn't have the file extension .sas7bdat.

By the way, if you delete the MyNum table from either the _TDyyyy folder or the work folder, the MyNum table will disappear from both the _TDyyyy folder and the work folder.

Go to the path_TDxxxx folder and delete the MyNum table. Then go to the work folder. Now the work folder is empty because we just deleted the physical MyNum table.

Now re-submit Program 1.8.4. A new temporary directory _TDxxx with different integers and characters will be generated.

1.8.4 Manually Delete PC SAS Temporary Folders

After your PC SAS programming job is done and you exit the SAS software, you'll want to manually delete all the temporary _TDxxxx folders that SAS failed to delete. If everything goes well, there shouldn't be any _TDxxxx folders in your hard drive after you close SAS. However, if your SAS program has errors, SAS may not be able to delete a temporary folder. Over time you'll have many temporary folders taking up space and slowing down your SAS program. You'll want to periodically delete all the temporary folders that SAS failed to delete.

1.9 Fourth Program - Calculate the Total with SQL

Often we need to find the total of some values such as integers from 1 to 10:

$$1 + 2 + ... + 10 = 55$$

Your first instinct probably is to implement the following loop in SAS:

```
int i, sum=0;
for(i=1; i<=10; i++)
{
 sum+=i;
}
```

You can certainly do that (see Program 9.1.5). However, for a large table, it's typically inefficient to scan each row. A better approach is to ask SAS to get the sum for you. After the work.MyNum table is created, submit the next program:

Program 1.9.1 *one2tenSum.sas*

```
proc sql;
create table MyTotal as
select sum(N) as total
from MyNum;

select * from MyTotal;
quit;
```

Output:

```
   total
--------
      55
```

For a big table, Program 1.9.1 is can be more efficient because internally SAS may use a faster approach to finding the sum than performing a full table scan.

1.10 Fifth Program - Create the Worker Table with SQL

Program 1.10.1 *worker.sas*

```
dm log 'clear'; dm output 'clear';
proc sql;
create table worker
(
   id num,
   sex char(1),
   firstName char(12),
   lastName char,
   salary num
);

insert into work.worker
values(1,'M','John','Smith',60000)
values(2,'F','Jane','Johnson',70000)
values(5,'F','Mary','Williams',80000)
values(20,'M','Robert','Walker',90000)
;

select * from worker;
quit;
```

Output:

```
      id  sex  firstName     lastName      salary
-------------------------------------------------
       1  M    John          Smith          60000
       2  F    Jane          Johnson        70000
       5  F    Mary          Williams       80000
      20  M    Robert        Walker         90000
```

This program specifies `sex` to be a 1-character column and `firstName` a 12-character column.

In the `create table` statement, if you don't specify the length of a numeric or character column, that column gets the default 8 bytes. As a result, `ID` and `salary` are both 8-byte numeric columns; `lastName` is an 8-character column.

You can verify that `sex` is a 1-byte character column. If you use SAS in the cloud, expand the `worker` table in the `work` library and you'll see all the columns in the `worker` table. Double click the `sex` column and the column properties window will show up. The column properties window should display the 6 properties of the `sex` column, `name`, `label`, `length`, `type`, `format`, and `informat`. Verify that the length of the `sex` column is 1.

If you have PC SAS, open the `work.worker` table. Highlight the `sex` column. Select Data > Column Attributes. The following window pops up:

```
----------------------------------------------
Column Attributes
```

```
        General | Colors | Fonts

Name:     sex
Label:                   ----------------------
Length:   1             | Type (radio button)  |
Format:   $1.           |  Character (checked) |
Informat: $1.           |  Numeric             |
-----------------------------------------------
```

In the Type radio button, the `Character` is checked, indicating that `sex` is a character column. In addition, the attribute window indicates that the column `name` is `sex`; the `label` is blank; the `length` is 1 byte; and the `format` and the `informat` are both `$1.`

Each column in a SAS table has these six attributes (or properties): `name`, `label`, `length`, `type`, `format`, and `informat`. `Name` and `type` are mandatory in the `create table` statement; you must specify them in your `create table` statement. In contrast, if you don't specify the remaining four attributes, SAS will assign each one a default value.

`Label` is a string of text that helps explain what a column is. Its default value is blank. If your column has a cryptic name, consider creating a meaningful `label`.

An `informat` tells SAS how to read raw strings from an external file into SAS. For example, the `yymmdd10.` informat will interpret the raw string `01/01/1960` as January 1, 1960. Without any `informat` to guide SAS, SAS will read in the raw string `01/01/1960` as text and you won't be able to do math on dates such as finding the number of days between two dates. The `informat` is solely for a DATA step to read in raw data from an external file; it is not used in PROC SQL.

A `format` controls how column values are displayed in the SAS table, in the output window, and in a printed report. For example, the `mmddyy10.` format will display the integer 0 as `01/01/1960`, while the `yymmdd10.` format will display the same date as `1960-01-01`.

An `informat` and a `format` must end with a period to distinguish from variable names. In addition, the `format` and the `informat` of a character column must begin with a dollar sign.

Under the `$1.` format, `sex` will be displayed as a 1-column string. Since the CREATE TABLE statement defines `sex` as 1-character column, SAS automatically assigns the `$1.` format to `sex`.

Similarly, this is the attribute window for the `salary` column:

```
-----------------------------------------------
Column Attributes
        General | Colors | Fonts

Name:     salary
Label:                   ----------------------
Length:   8             | Type (radio button)  |
Format:   BEST12.       |  Character           |
Informat: 12.           |  Numeric   (checked) |
-----------------------------------------------
```

The `BEST12.` format tells SAS to display `salary` with maximum precision subject to the limitation that salary has the maximum width of 12 characters. This is the default format of a numeric column.

You can open the attribute window for all the other columns in the `worker` table.

Another way to check the column attributes is to use the pre-built CONTENTS procedure. After the `work.worker` is created, run the following program:

Program 1.10.2 *workerProcContents.sas*

```
proc contents data=worker;
run;
```

No matter you use SAS in the cloud or PC SAS, this is part of your output:

```
 -----Alphabetic List of Variables and Attributes-----

       #    Variable     Type    Len    Pos
       ------------------------------------
       3    firstName    Char     12     17
       1    id           Num       8      0
       4    lastName     Char      8     29
       5    salary       Num       8      8
       2    sex          Char      1     16
```

The next program illustrates how to define optional attributes.

Program 1.10.3 *worker2.sas*

```
dm log 'clear'; dm output 'clear';
proc sql;
create table worker
(
   id num format=8. informat=8. ,
   sex char(1),
   firstName char(12) format $12.,
   lastName char,
   salary num label="annual$" format=best12.
);

insert into work.worker
values(1,'M','John','Smith',60000)
values(2,'F','Jane','Johnson',70000)
values(5,'F','Mary','Williams',80000)
values(20,'M','Robert','Walker',90000)
;

select * from worker;
quit;

proc contents data=worker;
run;
```

Submit the program. This is part of the output:

```
               id  sex  firstName     lastName       annual$
       ---------------------------------------------------
                1  M    John          Smith            60000
                2  F    Jane          Johnson          70000
                5  F    Mary          Williams         80000
               20  M    Robert        Walker           90000


                       The CONTENTS Procedure

         -----Alphabetic List of Variables and Attributes-----

#    Variable     Type    Len    Pos    Format     Informat    Label
----------------------------------------------------------------------
3    firstName    Char     12     17    $12.
1    id           Num       8      0    8.         8.
4    lastName     Char      8     29
5    salary       Num       8      8    BEST12.                annual$
2    sex          Char      1     16
```

Though `id` is assigned an informat `8.`, the informat is not used in PROC SQL.

Even though the `salary` column is now shown as `annual$` in the `worker` table and in the output window, `salary` is still the name of the column. If you need to reference the `salary` column, don't use `annual$`; the label `annual$` is just a decorative string.

In the next program, the lengths of two numeric columns, `id` and `salary`, are specified to be different than the default 8 bytes, but SAS resets their lengths to 8 bytes.

Program 1.10.4 *numLength.sas*

```
dm log 'clear'; dm output 'clear';
proc sql;
create table numLength
(
   id num(3), /*will still default to 8 bytes*/
   sex char(1),
   firstName char(12),
   lastName char,
   salary num(12) /*will still default to 8 bytes*/
);

quit;

proc contents data=numLength;
run;
```

Submit the program. You'll see the following as a part of the output:

```
-----Alphabetic List of Variables and Attributes-----

       #    Variable     Type    Len    Pos
       ------------------------------------
       3    firstName    Char     12     17
```

```
1    id          Num     8     0
4    lastName    Char    8    29
5    salary      Num     8     8
2    sex         Char    1    16
```

All the numeric columns in the `create table` statement get the default 8 bytes, even if you purposely assign to them a length that is less or greater than the default 8 bytes. So `id` and `salary` get the default length of 8 bytes.

However, you can assign to a character column a length different from the default 8 bytes, subject to the limitation that the maximum length of a character column is $32,767$ bytes.

Here's one thing that often trips SAS newcomers. Though an 8-byte character column stores up to 8 characters, the greatest integer that can be stored in an 8-byte numeric column under Windows is $2^{53} = 9,007,199,254,740,992$, not 99999999 as one might think.

Though character data is stored one character per byte in SAS, numeric data is stored as floating point numbers in real binary representation, which allows for $16-$ or $17-$digit precision within 8 bytes. 8 bytes are equal to 64 bites. The maximum base 10 value of a 64 binary is 2^{64}. However, some bits are reserved for representing the negative sign, the decimal point, or the exponent. Consequently, the base 10 value of a floating point number of a 64 binary is less than 2^{64} in Windows and Unix. The decimal precision of a full 8-byte number is effectively 15 decimal digits.

Be careful when you want to set the length of a numeric column to less than 8 bytes. If the value of a variable becomes large or has many significant digits, you can lose precision in the results of arithmetic calculations if the length of a numeric variable is less than 8 bytes. Even for an integer column, you may want to set its length wide enough to allow room for future growth.

1.11 SAS Session - Three Experiments

If you use PC SAS, you don't need to do any configuration. If you use SAS in the cloud, for the first two experiments, however, you need to change the mode from the default batch mode to the interactive mode. You do so using one of the following two methods. While the CODE window is active, click the "Go Interactive" icon. Alternatively, click the "More application options" icon, which is located to the right of the "Sign Out" button. Select "Preferences." Check "Start new programs in interactive mode."

1.11.1 First Experiment

PC SAS and SAS in the cloud Instead of submitting the three statements in Program 1.5.2 all together, let's submit one statement a time and see what happens. Once again, this is our `helloWorld.sas` program:

```
proc sql;
create table work.MyNum
(
   N num
);
quit;
```

Highlight the first statement:

```
proc sql;
```

Submit the program. SAS will execute only the highlighted code while ignoring the unhighlighted code. After the program is submitted, no table is generated. However, if you use PC SAS, the top left corner of the enhanced editor window should display the following title:

`PROC SQL running`

This message indicates that the SQL procedure is invoked and that SAS is ready to do a `CRUD` job.

If you use SAS in the cloud, you will not get the `PROC SQL running` title. You'll get this log:

```
NOTE: This session is in interactive mode.

            proc sql;
```

This indicates that PROC SQL was invoked.

Next, highlight the second statement:

```
create table work.MyNum
(
   N num
)
;
```

Submit the program. Once again, only the highlighted code is executed. You will get this log from PC SAS and SAS in the cloud:

`NOTE: Table WORK.MYNUM created, with 0 rows and 1 columns.`

A new `work.myNum` table is generated, overwriting the existing table with the same name.

Finally, highlight and submit the third statement:

```
quit;
```

This terminates the SQL procedure. In PC SAS, this causes the title `PROC SQL running` to disappear from the top left corner of your enhanced editor window. In addition, you'll get a log similar to this:

```
56   quit;
NOTE: PROCEDURE SQL used:
      real time           3.86 seconds
      cpu time            0.00 seconds
```

In SAS in the cloud, submitting the `quit` statement will generate a log similar to this:

```
16         quit;
NOTE: PROCEDURE SQL used (Total process time):
   real time           6.10 seconds
   user cpu time       0.01 seconds
   system cpu time     0.01 seconds
```

The effect of you sequentially submitting each statement in Program 1.5.2 is the same as submitting Program 1.5.2 all at once.

1.11.2 Second Experiment

PC SAS and SAS in the cloud After PROC SQL is terminated, highlight and submit the second statement of `helloWorld.sas`:

```
create table work.MyNum
(
   N num
)
;
```

In both PC SAS and SAS in the cloud, you get this error log:

```
ERROR 180-322: Statement is not valid or it is used out of proper order.
```

You got an error in the log because you were trying to create a table without first invoking the SQL procedure. The `create table` functionality is contained in the SQL procedure. To create a table, you must first invoke the SQL procedure.

1.11.3 Third Experiment

PC SAS First, create a new program:

Program 1.11.1 *worker3.sas*

```
dm log 'clear'; dm output 'clear';
libname mylib 'C:\LearnSAS';
proc sql;
create table mylib.worker
(
   id num,
   sex char(1),
   firstName char(12),
   lastName char,
   salary num
);

insert into mylib.worker
values(1,'M','John','Smith',60000)
values(2,'F','Jane','Johnson',70000)
values(5,'F','Mary','Williams',80000)
values(20,'M','Robert','Walker',90000)
;

select * from mylib.worker;
quit;
```

Output:

```
      id  sex  firstName     lastName    salary
-----------------------------------------------
       1  M    John          Smith        60000
       2  F    Jane          Johnson      70000
       5  F    Mary          Williams     80000
      20  M    Robert        Walker       90000
```

We'll partition `worker.sas` into 5 subprograms:

Program 1.11.2 *worker3Sub1.sas*

```
dm log 'clear'; dm output 'clear';
libname mylib 'C:\LearnSAS';
```

Program 1.11.3 *worker3Sub2.sas*

```
proc sql;
create table mylib.worker
(
   id num,
   sex char(1),
   firstName char(12),
   lastName char,
   salary num
);
```

Program 1.11.4 *worker3Sub3.sas*

```
insert into mylib.worker
values(1,'M','John','Smith',60000)
values(2,'F','Jane','Johnson',70000)
values(5,'F','Mary','Williams',80000)
values(20,'M','Robert','Walker',90000)
;
```

Program 1.11.5 *worker3Sub4.sas*

```
select * from mylib.worker;
```

Program 1.11.6 *worker3Sub5.sas*

```
quit;
```

It doesn't matter where you store `worker3.sas` and the 5 subprograms. However, for easy tracking, we'll store these 6 programs in the `C:\LearnSAS` folder.

Next, make sure all the 5 subprograms are opened under the same running instance of SAS. For example, while Program 1.11.2 is open, from the same running instance of SAS, use File > Open to open Program 1.11.3. Now these two subprograms are opened under the same running instance of SAS.

Next, submit Program 1.11.2, 1.11.3, 1.11.4, 1.11.5, and 1.11.6 in that order. This will create the same output table `mylib.worker` as does Program 1.11.1.

The effect of executing these 5 seemingly independent SAS programs under the same running instance of SAS is the same as submitting their combined program 1.11.1.

Multiple SAS programs under the same running instance of the SAS software is said to share the same SAS session.

The effect of executing several SAS programs under the same running instance of SAS is the same as executing a bigger SAS program that combines each executed program, with the earlier executed program appearing earlier in the combined program.

SAS in the cloud First reset the mode to the batch mode. Next, create the following program:

Program 1.11.7 *worker3cloud.sas*

```
libname mylib '/home/userid/myProgram';
proc sql;
create table mylib.worker
(
   id num,
   sex char(1),
   firstName char(12),
   lastName char,
   salary num
);

insert into mylib.worker
values(1,'M','John','Smith',60000)
values(2,'F','Jane','Johnson',70000)
values(5,'F','Mary','Williams',80000)
values(20,'M','Robert','Walker',90000)
;

select * from mylib.worker;
quit;
```

`mylib.worker` table:

id	sex	firstName	lastName	salary
1	M	John	Smith	60000
2	F	Jane	Johnson	70000
5	F	Mary	Williams	80000
20	M	Robert	Walker	90000

We'll partition `worker3cloud.sas` into 4 subprograms:

Program 1.11.8 *worker3cloudSub1.sas*

```
libname mylib '/home/userid/myProgram';
```

Program 1.11.9 *worker3cloudSub2.sas*

```
proc sql;
create table mylib.worker
(
   id num,
   sex char(1),
   firstName char(12),
   lastName char,
   salary num
);
quit;
```

Program 1.11.10 *worker3cloudSub3.sas*

```
proc sql;
insert into mylib.worker
values(1,'M','John','Smith',60000)
values(2,'F','Jane','Johnson',70000)
values(5,'F','Mary','Williams',80000)
values(20,'M','Robert','Walker',90000)
;
quit;
```

Program 1.11.11 *worker3cloudSub4.sas*

```
proc sql;
select * from mylib.worker;
quit;
```

Submit Program 1.11.8, 1.11.9, 1.11.10, and 1.11.11 in that order in the batch mode. This creates the same output `worker` table as does Program 1.11.7.

If you invoke PROC SQL in the batch mode in SAS in the cloud, to avoid server overload, SAS in the cloud automatically terminates PROC SQL after the program is executed regardless of whether you have an ending `quit` statement. As a result, you'll need to invoke PROC SQL in each program to perform CRUD.

If you use SAS in the cloud, you'll want to use the default batch mode most of the time. If you use the interactive mode, different SAS programs are independent from one another as if they were created under different running instances of the SAS software.

1.12 Sixth Program - Create the Employee Table

The next program creates the `work.employee` table and exports it to a CSV file in `C:\LearnSAS`.

Program 1.12.1 *employee.sas*

```
dm log 'clear'; dm output 'clear';
proc sql;
create table work.employee
(
   SSN num,
   Gender char(1),
   Dept char(11),
   JobClass char(3),
   HireDate num,
   Salary num
)
;

insert into work.employee
values(123456789,'M','Engineering','EA1','31Dec1959'd,60000)
values(234567890,'F','Engineering','EA2','1Jan1960'd,70000)
values(345678901,'M','Engineering','EA3','2Jan1960'd,80000)
;
quit;

proc export
data=work.employee
outfile='C:\LearnSAS\employee.csv'
dbms=csv
replace
;
run;
```

This is part of the log:

```
NOTE: 3 rows were inserted into WORK.EMPLOYEE.

3 records created in C:\LearnSAS\employee.csv from WORK.EMPLOYEE                    .
NOTE: C:\LearnSAS\employee.csv was successfully created.
```

This is the newly created `C:\LearnSAS\employee.csv` file:

SSN	Gender	Dept	JobClass	HireDate	Salary
123456789	M	Engineering	EA1	-1	60000
234567890	F	Engineering	EA2	0	70000
345678901	M	Engineering	EA3	1	80000

If you use SAS in the cloud, change the `outfile` line to:

`outfile='/home/userid/myProgram/employee.csv'`

Submit the program. It will create the same `employee.csv` file in the `/home/userid/myProgram` folder. This is the log:

```
3 records created in /home/userid/myProgram/employee.csv from WORK.EMPLOYEE.

NOTE: "/home/userid/myProgram/employee.csv" file was successfully created.
```

This is the syntax of `proc export`:

- `data` specifies the source table.
- `outfile` specifies the full path of the CSV file to be created.
- `dbms=csv` specifies that the output file type is csv.
- `replace` forces the newly generated csv file to overwrite any existing csv file with the same name.

Unlike PROC SQL, PROC EXPORT ends with the `run` statement.

You can open the CSV file with Excel and save it as an Excel file. If you use SAS in the cloud, download the CSV file to your hard drive before saving it as an Excel file.

1.12.1 SAS Dates

In Program 1.12.1, the `HireDate` column was declared to be `numeric`. In SAS, a date is an integer that represents the number of days between January 1, 1960 and a specified date. For example,

- December 30, 1959 is -2 (Day negative 2)
- December 31, 1959 is -1 (Day negative 1)
- January 1, 1960 is 0 (Day 0)

- January 2, 1960 is 1 (Day 1)
- January 3, 1960 is 2 (Day 2)

January 1, 1960 is expressed as the integer 0 or more conveniently `'1Jan1960'd` in SAS. If you don't know the integer corresponding to the date `mm/dd/yyyy`, you can express it as `'ddmmmyyyy'd`.

Make no mistake. `'1Jan1960'd` is not a string, but the number 0. The expression `'1Jan1960'd=0` evaluates to `true`. This makes date math simple. For example, `'1Jan1960'd+1=2`, meaning that one day after Jan 1, 1960 is January 2, 1960.

The followings are some of the equivalent ways of representing January 1, 1960:

- 0
- `'01jan1960'D`
- `'01JAN1960'd`
- `'01jAn1960'd`

You can use the `mdy()` function to find the integer value of a random date. For example,

- `mdy(12,31,1959)`=-1
- `mdy(1,1,1960)`=0
- `mdy(1,2,1960)`=1

Conversely, to find the month, the day, and the year of a given SAS date, use the `month()`, `day()`, and `year()` functions. For example,

- `month(-1)=12`
- `day(-1)=31`
- `year(-1)=1959`

1.12.2 Explore Informats and Formats

Program 1.12.2 *dateInformatFormat.sas*

```
dm log 'clear'; dm output 'clear';
proc sql;
create table work.HireDateTable
(
   HireDate num
)
;

```

```
/*  method 1 - use date integer*/
insert into work.HireDateTable
values('31Dec1959'd)
values('1Jan1960'd)
values('2Jan1960'd)
;

/*  method 2 - use mdy function*/
insert into work.HireDateTable
set HireDate=mdy(12,31,1959)
set HireDate=mdy( 1, 1,1960)
set HireDate=mdy( 1, 2,1960)
;

/*  method 3 --use informat date9.*/
insert into work.HireDateTable
set HireDate=input('31dec1959',date9.)
set HireDate=input('1jan1960', date9.)
set HireDate=input('2jan1960', date9.)
;

/*  method 4 --use informat date11.*/
insert into work.HireDateTable
set HireDate=input('31 dec 1959',date11.)
set HireDate=input('1 jan 1960', date11.)
set HireDate=input('2 jan 1960', date11.)
;

/*  method 5 --use informat mmddyy10.*/
insert into work.HireDateTable
set HireDate=input('12/31/1959', mmddyy10.)
set HireDate=input('1/1/1960',   mmddyy10.)
set HireDate=input('1/2/1960',   mmddyy10.)
;
quit;

options linewidth=min nonumber nodate;
/*  method 1 - display date as integer*/
proc print data= work.HireDateTable;
    title 'method 1;default best12. format; all the rows;';
run;

/*  method 2 - display date as integer*/
proc print data= work.HireDateTable(obs=3);
    title 'method 2;best12. format; first 3 rows';
    format HireDate best12.;  /* default format*/
run;

/*  method 3 - display date as mm/dd/yyyy */
proc print data= work.HireDateTable(obs=3);
    title 'method 3; 4.2 format; first 3 rows';
        format HireDate 4.2;
run;

/*  method 4 - display date as mm/dd/yyyy */
proc print data= work.HireDateTable(obs=3);
    title 'method 4;mmddyy10. format; first 3 rows';
    format HireDate mmddyy10.;
```

```
run;

/*  method 5 - display date as dd/mm/yyyy */
proc print data= work.HireDateTable(obs=3);
    title 'method 5;ddmmyy10. format; first 3 rows';
    format HireDate ddmmyy10.;
run;

/*  method 6 - display date as dd MMM yyyy */
proc print data= work.HireDateTable(obs=3);
    title 'method 6;yymmdd10. format; first 3 rows';
    format HireDate yymmdd10.;
run;

/*  method 7 - display date as ddMMMyyyy */
proc print data= work.HireDateTable(obs=3);
    title 'method 7;date9. format; first 3 rows';
    format HireDate date9.;
run;

/*method 8 - display date as weekday, full month-name dd, yyyy*/
proc print data= work.HireDateTable(obs=3);
    title 'method 8;WEEKDATE. format; first 3 rows';
    format HireDate WEEKDATE.;
run;

/*  method 9 - display date as 3-letter month-name dd, yyyy */
proc print data= work.HireDateTable(obs=3);
    title 'method 9;worddate12. format; first 3 rows';
    format HireDate worddate12.;
run;

/*  method 10 - display date as yyyyQquarter-name */
proc print data= work.HireDateTable(obs=3);
    title 'method 10;yyq6. format; first 3 rows';
    format HireDate yyq6.;
run;
```

This is the output:

```
method 1;default best12. format; all the rows;

                          Hire
                 Obs      Date

                   1        -1
                   2         0
                   3         1
                   4        -1
                   5         0
                   6         1
                   7        -1
                   8         0
                   9         1
                  10        -1
                  11         0
                  12         1
```

```
              13     -1
              14      0
              15      1
 method 2;best12. format; first 3 rows

          Obs        HireDate

            1              -1
            2               0
            3               1
  method 3; 4.2 format; first 3 rows

                     Hire
              Obs    Date

                1    -1.0
                2    0.00
                3    1.00
method 4;mmddyy10. format; first 3 rows

           Obs      HireDate

             1    12/31/1959
             2    01/01/1960
             3    01/02/1960
method 5;ddmmyy10. format; first 3 rows

           Obs      HireDate

             1    31/12/1959
             2    01/01/1960
             3    02/01/1960
method 6;yymmdd10. format; first 3 rows

           Obs      HireDate

             1    1959-12-31
             2    1960-01-01
             3    1960-01-02
 method 7;date9. format; first 3 rows

           Obs     HireDate

             1    31DEC1959
             2    01JAN1960
             3    02JAN1960
method 8;WEEKDATE. format; first 3 rows

  Obs             HireDate

    1    Thursday, December 31, 1959
    2        Friday, January 1, 1960
    3      Saturday, January 2, 1960
method 9;worddate12. format; first 3 rows

          Obs        HireDate

            1    Dec 31, 1959
```

```
              2     Jan 1, 1960
              3     Jan 2, 1960
    method 10;yyq6. format; first 3 rows

                     Hire
              Obs     Date

                1    1959Q4
                2    1960Q1
                3    1960Q1
```

The 5 `insert` statements (Lines 10, 17, 24, 31, and 38)) each write the same three dates, `12/31/1959`, `1/1/1960`, and `1/2/1960`, to the output table `HireDateTable`.

The INPUT function converts a character expression using a specified informat. The informat determines whether the resulting value is character or numeric.

The `date9.` informat converts a raw string `'ddmmmyyyy'` to a SAS date. For example, `date9.` reads the raw string `'31dec1959'` as the date of December 31, 1959 or the integer negative 1.

The `input('31dec1959',date9.)` function returns the integer date for December 31, 1959. Notice that the width of `'ddmmmyyyy'` is 9, hence the number 9 in the `date9.` informat.

The `date11.` informat interprets `'dd mmm yyyy'` as a date. Notice that the total width of `'dd mmm yyyy'` is 11. Similarly, the informat `mmddyy10.` interprets string `'mm/dd/yyyy'` as a date.

Finally, the ten PROC PRINT steps in Program 1.12.2 display the integer `HireDate` as various string representations.

In PROC PRINT, the `linesize=min` option tells SAS to use for each variable the minimum column width that accommodates all the variable values. The `nonumber` option tells SAS not to print the page number. Finally, the `nodate` option tells SAS not to display date in the output.

1.12.3 Set Column Format Attributes

You specify the `format` option in the `create table` statement to set a column's format attribute.

Program 1.12.3 *ssnTable.sas*

```
dm log 'clear'; dm output 'clear';
proc sql;
create table ssnTable1
(
   ssn num format ssn11.
)
;
insert into ssnTable1
values(123456789)
values(234567890)
values(345678901)
;
quit;

proc print data=ssnTable1;
 title 'ssnTable1 using ssn11. format';
run;

proc sql;
create table ssnTable2
(
   ssn num
)
;
insert into ssnTable2
values(123456789)
values(234567890)
values(345678901)
;

quit;

proc print data=ssnTable2;
  title 'ssnTable2 using ssn11. format';
  format ssn ssn11.;
run;

proc print data=ssnTable1;
  title 'ssnTable1 using best12. format';
  format ssn best12.;
run;
```

Output:

```
ssnTable1 using ssn11. format

     Obs            ssn

      1     123-45-6789
      2     234-56-7890
      3     345-67-8901
ssnTable2 using ssn11. format

     Obs            ssn

      1     123-45-6789
      2     234-56-7890
      3     345-67-8901
ssnTable1 using best12. format

     Obs             ssn

      1        123456789
      2        234567890
      3        345678901
```

The `ssn11.` format displays a social security number such as `123456789` in the form of `123-45-6789`.

1.12.4 Difference between Informats and Formats

To summarize. You use an `informat` to tell SAS how your raw data looks like so SAS can extract the correct value from your raw input data and store the correct value in a table. You use a `format` to tell SAS how to display the data stored in a SAS table to you.

`Formats` are for displaying purpose only. They don't alter or truncate the data stored in a SAS table. `Informats`, on the other hand, determine how to extract the correct value from the raw input data. With a wrong `informat`, SAS will retrieve the wrong value from the input data.

An `informat` and a `format` may have the same expression. For example, `mmddyy10.` can be used either as an `informat` or a `format`. As an `informat`, it instructs SAS to read the string `mm/dd/yyyy` as a date. As a `format`, it tells SAS to display a date in the string form `mm/dd/yyyy`.

1.13 Further Reading

The free official SAS tutorial pdf, *Step-by-Step Programming with Base SAS Software*, by SAS Institute Inc, 2001. Cary, NC: SA Institute Inc. Google it to download the pdf. Chapter 1, *What Is the SAS System?*

The free official SQL user guide pdf, *SAS 9.3 SQL Procedure User's Guide*, Chapter 4, *Creating and Updating Tables and Views.*

Step-by-Step Programming with Base SAS Software, Chapter 14, *Working with Dates in the SAS System.*

Introduction to SAS Informats and Formats

`http://support.sas.com/publishing/pubcat/chaps/59498.pdf`.

Choose your own adventure with SAS OnDemand for Professionals

`http://blogs.sas.com/content/sasdummy/2012/01/18/sasondemand-forpros/`

Free SAS!

`http://statisticalhorizons.com/free-sas`

On the difference between the batch mode and the interactive mode, refer to *Step-by-Step Programming with Base SAS Software*, Chapter 38, *Introducing the SAS Environment*, Page 645, *Selecting a SAS Processing Mode.*

Chapter 2

DATA Step Basics

There's almost always more than one way to do everything in SAS. Instead of using PROC SQL, you can use a DATA step to create or modify a table.

2.1 Hello World DATA Step Program

The next program creates a `work.MyNum` table with one blank row. A period represents a missing numeric value.

Program 2.1.1 *helloWorldDataStep.sas*

```
1 data MyNum;
2 length N 3;
3 output MyNum;
4 return;
5 run;
```

`MyNum` table:

	N
1	.

A DATA step begins with the `data` statement and ends with the `run` statement.

Line 1 declares that the `MyNum` table is to be created. `Data` in SAS means a table or a data set.

Line 2 declares the column `N` and specifies its length to be 3 bytes. Since the data type of `N` is not specified, SAS automatically sets `N` to be a numeric column.

Unlike the CREATE TABLE statement in PROC SQL, a DATA step can set the length of a numeric column to be less than the default 8 bytes, subject to the limitation that the minimum length of a numeric variable is 2 bytes in mainframe environments and 3 bytes in non-mainframe environments (Windows and Unix).

Line 3 writes the value of `N` as one row to the `MyNum` table. The `output` statement is similar to the `insert into` statement in PROC SQL. Since no value is explicitly assigned to `N` before the `output` statement, the `output` statement will write a missing numeric value to the `MyNum` table.

Line 4. The `return` statement does two things: (1) It causes the execution to stop at the current point in the DATA step, and (2) it returns control to the beginning of the DATA step so the DATA step can start its next iteration.

Every DATA step program is an implicit loop so the `output` statement can be executed multiple times to create multiple rows in the destination table specified in the `data` statement (more on this in Section 4.12). However, the DATA step loop in Program 2.1.1 has only one iteration and the `return` statement isn't necessary.

Next, open the `work.MyNum` table and verify that `N` is indeed a numeric column (see Section 1.10).

In Program 2.1.1, you can omit the `output` statement and the `return` statement altogether because SAS by default will automatically execute an implicit `output` statement and an implicit `return` statement at the end of the DATA step. Program 2.1.1 can be shortened to:

Program 2.1.2 *helloWorldDataStepShort.sas*

```
data MyNum;
length N 3;
run;
```

Though Program 2.1.1 and Program 2.1.2 are equivalent, newcomers to SAS are encouraged to use the coding style in Program 2.1.1 to spell out the DATA step. After they have a good grasp of SAS DATA step fundamentals, they can switch to a terse coding style as in Program 2.1.2.

The next program creates a blank table `MyNum` with no rows.

Program 2.1.3 *helloWorldDataStepBlankTable.sas*

```
1 data MyNum;
2 length N 3;
3 stop;
4 output MyNum;
5 return;
6 run;
```

`MyNum` table:

N

The newly created `work.MyNum` table has the column header `N` but doesn't have any rows:

The `stop` statement instructs SAS to immediately stop processing the current DATA step and to resume processing statements after the end of the current DATA step.

In Program 2.1.3, after the `stop` statement is executed, neither the `output` statement nor the `return` statement will be executed and the entire DATA step will come to an end. If there are any statements after Line 6, these statements will be executed next after the `stop` statement is executed.

In a SAS DATA step, a variable is numeric by default unless specified otherwise. The next program creates `MyChar` table that has a character column `N` with length 3.

Program 2.1.4 *helloWorldDataStepChar.sas*

```
data MyChar;
length N $ 3;
output MyChar;
return;
run;
```

`MyChar` table:

	N
1	

The newly created `work.MyChar` table has one row populated with a missing character value.

In the `length` statement, the dollar sign indicates that `N` is a character column.

In SAS, while a missing numeric value is represented by a period, a missing character value is represented by a blank.

Next, open the `work.MyNum` table and verify that `N` is a character column (see Section 1.10).

Similar to Program 2.1.3, the next program creates a blank `work.MyChar` table without any rows.

Program 2.1.5
helloWorldDataStepCharBlankTable.sas

```
data MyChar;
length N $ 3;
stop;
output MyChar;
return;
run;
```

`MyChar` table:

N

The newly created `work.MyChar` table has only the column header `N` but doesn't have any rows.

While PROC SQL ends with the `quit` statement, a DATA step ends with the `run` statement. This is largely an inconsistency in the SAS language implementation. More on this in Section 2.7.

By the way, this is the shortest DATA program you can ever write:

Program 2.1.6 *shortestDataStepProgram.sas*

```
data;
run;
```

Submit Program 2.1.6 the first time and you'll get this log:

```
NOTE: The data set WORK.DATA1 has 1 observations and 0 variables.
```

You have created the `work.data1` table but it doesn't have any columns. Though the `work.data1` table isn't useful, Program 2.1.6 is a valid DATA step program.

If you don't specify the destination table in the `data` statement, SAS automatically uses `dataN` as the destination table. The integer N starts from 1 and increments by 1.

Submit Program 2.1.6 the second time and you'll get this log:

```
NOTE: The data set WORK.DATA2 has 1 observations and 0 variables.
```

2.2 Populate a Table with Integers from 1 to 10

Here's the DATA step counterpart to Program 1.7.1:

Program 2.2.1 *one2tenDataStep.sas*

```
dm log 'clear'; dm output 'clear';
data MyNum;
N=1;  output MyNum;
N=2;  output MyNum;
N=3;  output MyNum;
N=4;  output MyNum;
N=5;  output MyNum;
N=6;  output MyNum;
N=7;  output MyNum;
N=8;  output MyNum;
N=9;  output MyNum;
N=10; output MyNum;

return;
run;

proc print data=MyNum;
run;
```

Output:

```
Obs     N

  1     1
  2     2
  3     3
  4     4
  5     5
  6     6
  7     7
  8     8
  9     9
 10    10
```

In the output, `Obs` stands for observation, which means a row in a table.

Every variable in a DATA step will automatically become a column of the output table specified in the `data` statement. Since numeric values are assigned to `N`, SAS automatically sets `N` to be a numeric column. Since no length is specified for `N`, SAS sets its length to the default 8 bytes.

Finally, the pre-built PRINT procedure writes the content of the `MyNum` table to the output window.

PROC PRINT is similar to the following SQL statements:

```
proc sql;
select * from MyNum;
quit;
```

Compare the output generated by PROC SQL and that generated by PROC PRINT:

```
select * from MyNum     PROC PRINT data=MyNum

       N                 Obs     N
--------
       1                   1     1
       2                   2     2
       3                   3     3
       4                   4     4
       5                   5     5
       6                   6     6
       7                   7     7
       8                   8     8
       9                   9     9
      10                  10    10
```

For debugging your code, the two methods are roughly the same. Both display the data values of a table in the output window. However, PROC PRINT offers finer controls over the final output and can produce headers, total lines, and grand total lines. You can use PROC PRINT to produce pretty documents to distribute to end users. In addition, using ODS (output delivery system), PROC PRINT can export a table to a CSV file, a PDF file, an HTML file, or a Microsoft Word document. To learn more about PROC PRINT and ODS, refer to Section 2.8.

2.3 More DATA Step Programs

The next DATA step creates the `work.MNO` table with three columns, `M`, `N`, and `O`.

Program 2.3.1 *MNO.sas*

```
dm log 'clear';  dm output 'clear';
data MNO;
M=1; N=10; O='abc'; output MNO;
M=2; N=20; O='def'; output MNO;
M=3; N=30; O='ghi'; output MNO;

return;
run;

proc print data=MNO;
run;
```

Output:

```
Obs    M     N     O

 1     1    10    abc
 2     2    20    def
 3     3    30    ghi
```

Program 2.3.1 is equivalent to the following PROC SQL program:

Program 2.3.2 *MNOequivalent.sas*

```
dm log 'clear';  dm output 'clear';
proc sql;
create table MNO
(
   M num,
   N num,
   O char(3)
);

insert into MNO
values(1,10,'abc')
values(2,20,'def')
values(3,30,'ghi')
;

select * from MNO;
quit;
```

Output:

```
       M         N  O
-----------------------
       1        10  abc
       2        20  def
       3        30  ghi
```

2.4 Do-Loop and If-Then-Else in a DATA Step

So far a DATA step appears to be just another way to populate a table; nothing spectacular. But wait! Unlike PROC SQL, a DATA step allows a DO-LOOP. Program 2.2.1 can be rewritten as:

Program 2.4.1
one2tenDataStepLoop.sas

```
dm log 'clear'; dm output 'clear';
data MyNum;
do N=1 to 10;
  output MyNum;
end;

return;
run;

proc print data=MyNum;
run;
```

Output:

```
Obs    N

  1    1
  2    2
  3    3
  4    4
  5    5
  6    6
  7    7
  8    8
  9    9
 10   10
```

The DATA step in Program 2.4.1 iterates only once. However, inside the DATA step, the DO-LOOP iterates ten times.

While a CREATE TABLE statement can create only one table, one DATA step can create multiple tables. The next program creates three identical tables `MyNum1`, `MyNum2`, and `MyNum3`. In each iteration of the DO-LOOP, the same value of `N` is inserted to three tables, `MyNum1`, `MyNum2`, and `MyNum3`.

Program 2.4.2
one2tenDataStepLoop3Tables.sas

```
dm log 'clear'; dm output 'clear';
data MyNum1 MyNum2 MyNum3;
do N=1 to 10;
  output MyNum1 MyNum2 MyNum3;
end;

return;
run;

proc print data=MyNum1;
 title 'MyNum1';
run;

proc print data=MyNum2;
 title 'MyNum2';
run;

proc print data=MyNum3;
 title 'MyNum3';
run;
```

Output:

```
  MyNum1

Obs     N

  1     1
  2     2
  3     3
  4     4
  5     5
  6     6
  7     7
  8     8
  9     9
 10    10
  MyNum2

Obs     N

  1     1
  2     2
  3     3
  4     4
  5     5
  6     6
  7     7
  8     8
  9     9
 10    10
  MyNum3

Obs     N

  1     1
  2     2
  3     3
  4     4
  5     5
  6     6
  7     7
  8     8
  9     9
 10    10
```

In PROC SQL, you cannot use the IF-THEN-ELSE statement; you need to use the CASE expression instead. In contrast, a DATA step allows the IF-THEN-ELSE statement.

Program 2.4.3
one2tenDataStepLoopEvenOdd.sas

```
dm log 'clear'; dm output 'clear';
data Both Odd Even;
do N=1 to 10;
  output Both;
  if mod(N,2)=1 then output Odd;
  else output Even;
end;

return;
run;

proc print data=Both;
 title 'Both';
 title2 'Contains integer 1 to 10';
run;

proc print data=Odd;
 title 'Odd';
 title2 'Contains integer 1, 3, 5, 7, and 9';
run;

proc print data=Even;
 title 'Even';
 title2 'Contains integer 2, 4, 6, 8, and 10';
run;
```

Output:

```
                Both
      Contains integer 1 to 10

             Obs     N

               1     1
               2     2
               3     3
               4     4
               5     5
               6     6
               7     7
               8     8
               9     9
              10    10
                Odd
 Contains integer 1, 3, 5, 7, and 9

             Obs    N

              1     1
              2     3
              3     5
              4     7
              5     9
                Even
 Contains integer 2, 4, 6, 8, and 10

             Obs     N

              1      2
              2      4
              3      6
              4      8
              5     10
```

The unconditional `output Both` statement inserts every `N` into the `Both` table. The `mod(N,2)` function returns the remainder of N divided by 2. If the remainder is 1, then `N` is inserted into the `Odd` table; if the remainder is not 1 (e.g. 0), then `N` is inserted into the `Even` table.

The `title` statement adds a line of text to the top of the report. You can use up to ten `title` statements, `title1` (same as `title`), `title2`, ..., and `title10`, to include 1 to 10 lines of text at the top of the report. You must enclose the text of each `title` in single or double quotes.

Once you specify a `title` for a line, it is used for all subsequent output under the same running instance of the SAS software until you cancel the title or define a new title.

Let's experiment. After submitting Program 2.4.3, open a new enhanced editor window and create Program 2.4.4:

Program 2.4.4 *ByHundred.sas*

```
dm log 'clear'; dm output 'clear';
data ByHundred;
do i=100 to 1000 by 100;
 output ByHundred;
end;

return;
run;

proc print data=ByHundred;
run;
```

Output:

```
                 Even
Contains integer 2, 4, 6, 8, and 10

             Obs      i

               1     100
               2     200
               3     300
               4     400
               5     500
               6     600
               7     700
               8     800
               9     900
              10    1000
```

To cancel a title, you use an empty `title` statement without an argument. The next PROC PRINT uses an empty `title` statement to reset the report title to blank.

```
proc print data=one2tenByHundred;
 title;
run;
```

The next program creates three big tables. The `Both` table has one million integers: 1, 2, ..., $1,000,000$. The `Odd` table contains $500,000$ odd integers: 1, 3, ..., $999,999$. And the `Even` table contains $500,000$ even integers: 2, 4, ..., $1,000,000$.

Program 2.4.5 *one2oneMillionDataStepLoop.sas*

```
dm log 'clear'; dm output 'clear';
data Both Even Odd;
do N=1 to 1e6;
  output Both;
  if mod(N,2)=0 then output Even; else output Odd;
end;

return;
run;

proc print data=Both(firstobs=999996 obs=1000000);
 title 'last 5 rows from Both table';
run;

proc print data=Odd(firstobs=499996 obs=500000);
 title 'last 5 rows from Odd table';
run;

proc print data=Even(firstobs=499996 obs=500000);
 title 'last 5 rows from Even table';
run;
```

Output:

```
last 5 rows from Both table

        Obs        N

     999996     999996
     999997     999997
     999998     999998
     999999     999999
    1000000    1000000
last 5 rows from Odd table

        Obs        N

     499996    999991
     499997    999993
     499998    999995
     499999    999997
     500000    999999
last 5 rows from Even table

        Obs        N

     499996     999992
     499997     999994
     499998     999996
     499999     999998
     500000    1000000
```

The `data xyz(firstobs=m obs=n)` expression where $m \leq n$ retrieves $n - m + 1$ rows from Table `xyz`, starting from Row m and ending with Row n. If `firstobs=`m is omitted, m defaults to 1.

Make sure $m \leq n$ or you'll get an error log:

```
ERROR: FIRSTOBS option > OBS option - no data to read from file xyz.
```

Similar to Program 1.9.1, the next program finds the sum of the integers from 1 to 10:

Program 2.4.6 *one2tenSumDataStep.sas*

```
dm log 'clear';  dm output 'clear';
data MyNumTotal;
total=0;
do N=1 to 10;
  total=total + N;
  output MyNumTotal;
end;

return;
run;

proc print data=MyNumTotal;
run;
```

Output:

Obs	total	N
1	1	1
2	3	2
3	6	3
4	10	4
5	15	5
6	21	6
7	28	7
8	36	8
9	45	9
10	55	10

Can you explain the output of Program 2.4.7?

Program 2.4.7 *MNOTable.sas*

```
dm log 'clear';  dm output 'clear';
data MNO;
M=1; N=10; O='abc';
M=2; N=20; O='def';
M=3; N=30; O='ghi';
output MNO;
return;
run;

proc print data=MNO;
 title 'MNO table';
run;
```

Output:

MNO table

Obs	M	N	O
1	3	30	ghi

The resulting table `MNO` has only one row, not three rows as one might expect. The DATA step has only one iteration and the `output` statement is executed only once. Only the final values of `M`, `N`, and `O` before the `output` statement are inserted into `MNO`.

Program 2.4.7 is equivalent to the following PROC SQL program:

Program 2.4.8
MNOTableSQLequivalent.sas

```
dm log 'clear';  dm output 'clear';
proc sql;
create table MNO
(
   M num,
   N num,
   O char(3)
);
insert into MNO
values(3,30,'ghi');
quit;

proc sql;
 title 'MNO table';
 select * from MNO;
quit;
```

Output:

```
       MNO table

       M         N  O
-----------------------
       3        30  ghi
```

Can you explain the output of Program 2.4.9?

Program 2.4.9 *MNOTable2.sas*

```
dm log 'clear';  dm output 'clear';
data MNO2;
N=10; P=M+N;         output MNO2;
M=1; O='def'; P=M+N; output MNO2;
M=3; P=M+N;          output MNO2;

return;
run;

proc print data=MNO2;
 title 'MNO2 table';
run;
```

Output:

```
        MNO2 table

Obs      N      P     M      O

 1      10      .     .
 2      10     11     1     def
 3      10     13     3     def
```

SAS scans all the variables in the DATA step to determine the column names of the output table. All the variable names automatically become the column names of the output table. The columns in the output table are in the same order in which the variables first appear in the DATA step.

The four variables in the DATA step first appear in this order: `N`, `P`, `M`, and `O`. SAS creates four columns, `N`, `P`, `M`, and `O`, in the order of each variable's first appearance in the DATA step.

In a DATA step, to determine the data type and the length of a column in the output table, SAS uses the rule of the first encounter. When SAS first encounters a variable in the DATA step, that variable's data type and length become the data type and the length for that variable in the output table.

Before the first `output` statement executes, only the variable `N` is initialized. The other three variables are not initialized and SAS sets them to missing. The `output` statement inserts `N=10, P=missing, M=missing, O=missing` into the first row of the `MNO2` table.

Before the second `output` statement executes, SAS gathers the final data values up to that `output` statement and finds that `N=10`, `M=1`, `O='def'`, and `P=M+N=11`. SAS inserts `N=10,P=11, M=1, O='def'` into the second row of the table.

Before the third `output` statement executes, SAS gathers the final data values up to that `output` statement and finds that `N=10`, `M=3`, `O='def'`, and `P=M+N=13`. SAS inserts `N=10,P=13, M=1, O='def'` into the third row of the table.

The DATA step in Program 2.4.9 iterates only once.

2.5 DATA Step Boundary - PC SAS

Newcomers to PC SAS often have this experience. They create a DATA step program and submit it, but nothing happens. No table is created; no output is generated; and no log is produced to tell what has gone wrong. Here's an example.

Program 2.5.1 *dataStepNothingHappened.sas*

```
dm log 'clear'; dm output 'clear';
data A;
do i=1 to 10;
  output;
end;
```

Submit the program. Open the `work` folder and you won't see the table `A`. Check the log and you won't see any WARNING or NOTE messages. The only clue you get is that the following message title will appear at the top left corner of the enhanced editor window:

```
DATA STEP running
```

What just happened? ANSWER: You forgot to end the DATA step.

To fix the error, first cancel the submitted SAS job:

- From the Application Toolbar, click the exclamation mark enclosed in a circle. Now the Tasking Manager window opens up.
- Select `Cancel Submitted Statements` from the Tasking Manager window.
- Next, a `Break -> Submit` window opens up.
- Select `Y to cancel submitted statements`. Click OK.

Now your submitted SAS job is cancelled. The `DATA STEP running` title will disappear from the top left corner of the enhanced editor window. Next, add the statement `proc print data=A;` after the end of Program 2.5.1:

Program 2.5.2 *dataStepExecutedButProcPrintNot.sas*

```
dm log 'clear'; dm output 'clear';
data A;
do i=1 to 10;
  output;
end;

proc print data=A;
```

Submit this new program. You should get this log:

```
NOTE: The data set WORK.A has 10 observations and 1 variables.
NOTE: DATA statement used:
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Open the `work.A` table and you should see that `work.A` is correctly populated with the integers from 1 to 10. However, there's nothing in the output window; PROC PRINT was not executed. The top left corner of the enhanced editor window has this message title:

```
PROC PRINT running
```

What just happened When you submit a program, be it a DATA step or a PROC step, your program code goes into an area of memory called the input stack. Once your code is in the input stack, SAS will do the following:

- read the code in the input stack from left to right and from top to bottom
- send the code to the compiler
- stop sending code to the compiler when a step boundary such as a RUN statement, a new DATA step, or a PROC step is encountered
- execute the compiled code if there are no compilation errors
- repeat this process for any subsequent steps

In order for a DATA step or PROC step to be executed, SAS must find a step boundary such as a RUN statement or another DATA step or PROC step. The step boundary triggers SAS to execute the preceding step.

Program 2.5.1 doesn't have a step boundary to mark the ending of the DATA step. After you submit the program, SAS sends all your code to the compiler so your code gets compiled. Then SAS waits for you to type more code for it to send to the compiler. Without a step boundary, the compile phase never ends and your program never gets executed.

In Program 2.5.2, the PROC PRINT step after the DATA step serves as the implicit step boundary of the DATA step. However, there's no explicit or implicit step boundary to signal the ending of the PROC PRINT step. After Program 2.5.2 is submitted, the DATA step is executed, but the

PROC PRINT step will not be executed. SAS will wait forever for you to type more code in the PROC PRINT step for it to send to the compiler.

To execute the PROC PRINT step, at the end of Program 2.5.2, type the `run` statement. Highlight and submit the `run` statement:

```
run;
```

The `run` statement is the explicit boundary for the PROC PRINT step. It tells SAS that the compilation stage of the PROC PRINT step is over and the execution phase should begin. As a result, the PROC PRINT step is executed. You'll get a log similar to this:

```
     proc print data=A;
     run;

NOTE: There were 10 observations read from the dataset WORK.A.
NOTE: PROCEDURE PRINT used:
      real time           xx
      cpu time            0.00 seconds
```

where `xx` is the time in seconds it took SAS to complete the PROC PRINT job.

In addition, you'll get this desired output:

```
Obs     i

  1     1
  2     2
  3     3
  4     4
  5     5
  6     6
  7     7
  8     8
  9     9
 10    10
```

Though you don't need to use a `run` statement to explicitly mark the step boundary of a DATA step or a PROC PRINT step, it's considered the best practice to do so.

The next program use the `run` statement to explicitly mark the boundary of the DATA step and the PROC PRINT step:

Program 2.5.3 *useRunToMarkStepBoundary.sas*

```
dm log 'clear'; dm output 'clear';
data A;
do i=1 to 10;
  output;
end;
run;

proc print data=A;
run;
```

2.6 DATA Step Boundary - SAS in the cloud

If you submit Program 2.5.1 in SAS in the cloud, you'll get the following error in the log:

```
ERROR 180-322: Statement is not valid or it is used out of proper order.

WARNING: The data set WORK.A may be incomplete.  When this step was
 stopped there were 0 observations and 1 variables.
```

If you re-submit Program 2.5.1 in SAS in the cloud, you'll get the following error in the log:

```
ERROR 180-322: Statement is not valid or it is used out of proper order.

WARNING: The data set WORK.A may be incomplete.  When this step was
 stopped there were 0 observations and 1 variables.
WARNING: Data set WORK.A was not replaced because this step was stopped.
```

Program 2.5.1 is missing a boundary for the DATA step. SAS in the cloud will throw an error in the log, whereas PC SAS will display a message title `DATA STEP running`.

If your PROC PRINT is missing a step boundary, PC SAS will display a message title `PROC PRINT running`. In contrast, SAS in the cloud will execute the PROC PRINT without throwing an error or issuing a warning in the log.

Program 2.6.1 *procNoBoundaryCloud.sas*

```
data A;
do i=1 to 10;
  output;
end;
run;

proc print data=A;
```

Submit the program in SAS in the cloud. The content of the `work.A` table will be displayed in the RESULTS window, even though the PROC PRINT doesn't have a step boundary.

Even though SAS in the cloud executed your PROC PRINT job regardless of whether it has a step boundary, it's considered the best practice to explicitly mark the step boundary of PROC PRINT.

2.7 Quit Versus Run

You probably noticed that PROC SQL uses an ending `quit` statement but DATA steps or other procedures end with a `run` statement. What's the difference between `quit` and `run`?

In Section 1.11, you experimented with the `quit` statement. You found that in PC SAS once you invoke the SQL procedure, the SQL procedure is always running and that you can submit another `CRUD` statement without reinvoking the SQL procedure. To exit the SQL procedure, you submit the `quit` statement.

In PROC SQL, the step boundary is not important. Once PROC SQL is invoked, every `CRUD` statement you submit is executed immediately.

The SAS Institute adopted two different code-execution philosophies. For DATA steps and most non-SQL procedures, the SAS default behavior is not to execute the code until it sees a step boundary such as a `run` statement. For PROC SQL, however, the SAS default behavior is to execute the code immediately.

Why two different philosophies? My guess is that SAS evolved over time. In early versions of SAS, there wasn't PROC SQL and the SAS Institute felt that the "don't run unless the boundary is found" defensive philosophy was good for end users. Later when the SQL procedure was added, the SAS Institute switched its position and adopted the "always run PROC SQL unless told to stop" philosophy.

The SAS Institute perhaps should have adopted coherent syntax to free everyone from worrying about the difference between `quit` and `run`, but we live in an imperfect world.

2.8 Further Reading

Step-by-Step Programming with Base SAS Software, Chapter 2, *Introduction to DATA Step Processing.*

For the `output` statement, refer to

`http://v8doc.sas.com/sashtml/lgref/z0194540.htm`

PROC PRINT - the Granddaddy of all Procedures, Enhanced and Still Going Strong!

`http://www.lexjansen.com/nesug/nesug11/ds/ds09.pdf`

ODS Step by Step

`http://www.lexjansen.com/nesug/nesug03/ps/ps020.pdf`

PROC PRINT and ODS: Teaching an Old PROC New Tricks

`https://support.sas.com/resources/papers/proceedings11/270-2011.pdf`

Using 22 Easy Tricks with ODS to Generate Colorful Reports

`http://www.mwsug.org/proceedings/2007/tutorials/MWSUG-2007-T10.pdf`

General Index

Index of 263 SAS Programs

Index of Major Text Files