SAS® Certification Review: Base Programming for SAS®9 Course Notes was developed by Michele Ensor. Additional contributions were made by Kent Reeve, Lori Rothenberg, Lorilyn Russell, Larry Stewart, and Su Chee Tay. Editing and production support was provided by the Curriculum Development and Support Department.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.

SAS® Certification Review: Base Programming for SAS®9 Course Notes

Copyright © 2013 SAS Institute Inc. Cary, NC, USA. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

Book code E2408, course code LWCRB/CRB, prepared date 27Mar2013. LWCRB_004

ISBN 978-1-61290-698-0
# Table of Contents

Course Description ...................................................................................................................... vi

Prerequisites ............................................................................................................................... vii

## Chapter 1  Introduction  .......................................................................................... 1-1

1.1  Specifics about the SAS Base Programming Exam ......................................................... 1-3

1.2  Specifics about This Review Course ............................................................................. 1-10

1.3  SAS Fundamental Concepts .......................................................................................... 1-14

1.4  Answers to Questions .................................................................................................... 1-31

## Chapter 2  Working with SAS Data Sets ............................................................... 2-1

2.1  Reading and Creating Data Sets ...................................................................................... 2-3

2.2  Selecting Observations ................................................................................................... 2-29

2.3  Sorting Observations with the SORT Procedure ............................................................ 2-42

2.4  Combining Data Sets ..................................................................................................... 2-45

2.5  Answers to Questions .................................................................................................... 2-57

## Chapter 3  Working with Raw Data and Microsoft Excel Files ............................ 3-1

3.1  Reading Raw Data Files: Part 1 ....................................................................................... 3-3

3.2  Reading Raw Data Files: Part 2 ..................................................................................... 3-16

3.3  Controlling When a Record Loads .................................................................................. 3-32

3.4  Reading Microsoft Excel Files ....................................................................................... 3-38

3.5  Answers to Questions .................................................................................................... 3-42

## Chapter 4  Creating Variables ................................................................................ 4-1

4.1  Creating Variables with the Assignment Statement ......................................................... 4-3
4.2 Creating Variables Conditionally ................................................................. 4-8
4.3 Creating Accumulator Variables ................................................................. 4-19
4.4 Answers to Questions ..................................................................................... 4-25

Chapter 5  Manipulating Data ........................................................................... 5-1

5.1 Using Functions to Manipulate Data .............................................................. 5-3
5.2 Converting Character and Numeric Data ....................................................... 5-34
5.3 Processing Data with DO Loops ................................................................. 5-41
5.4 Processing Data with Arrays ....................................................................... 5-51
5.5 Answers to Questions ..................................................................................... 5-59

Chapter 6  Generating Reports ......................................................................... 6-1

6.1 Creating Detail Reports with the PRINT Procedure ...................................... 6-3
6.2 Creating Formats with the FORMAT Procedure ............................................ 6-22
6.3 Creating Frequency Tables with the FREQ Procedure ............................... 6-30
6.4 Creating Summary Reports with the MEANS Procedure ............................ 6-35
6.5 Directing Reports to External Files with ODS ............................................. 6-40
6.6 Answers to Questions ..................................................................................... 6-46

Chapter 7  Additional Information ................................................................... 7-1

7.1 More Specifics about the SAS Base Programming Exam ............................. 7-3
7.2 Additional Preparation Resources ................................................................. 7-6
7.3 Test-Taking Strategies ................................................................................... 7-9

Appendix A  Exercises and Solutions ................................................................. A-1

A.1 Exercises ...................................................................................................... A-3
   Chapter 1 ...................................................................................................... A-3
Course Description

This course provides a review of the majority of topics in the SAS® Base Programming Exam for SAS®9. It addresses the five exam content areas: Accessing Data, Creating Data Structures, Managing Data, Generating Reports, and Handling Errors.

To learn more…

For information about other courses in the curriculum, contact the SAS Education Division at 1-800-333-7660, or send e-mail to training@sas.com. You can also find this information on the Web at support.sas.com/training/ as well as in the Training Course Catalog.

For a list of other SAS books that relate to the topics covered in this Course Notes, USA customers can contact our SAS Publishing Department at 1-800-727-3228 or send e-mail to sasbook@sas.com. Customers outside the USA, please contact your local SAS office.

Also, see the Publications Catalog on the Web at support.sas.com/pubs for a complete list of books and a convenient order form.
Prerequisites

Before attending this course, you should

• be an experienced programmer with knowledge of the five exam content areas
• have taken SAS® Programming 1: Essentials and SAS® Programming 2: Data Manipulation Techniques or have equivalent experience.
Chapter 1  Introduction

1.1  Specifics about the SAS Base Programming Exam....................................................1-3

1.2  Specifics about This Review Course ........................................................................1-10

1.3  SAS Fundamental Concepts.....................................................................................1-14

1.4  Answers to Questions................................................................................................1-31
1.1 Specifics about the SAS Base Programming Exam

SAS Certification

SAS offers professional certifications that validate a candidate’s knowledge within the following areas:

- SAS Foundation
- SAS Advanced Analytics
- SAS Information Management
- SAS Data Management
- SAS Business Intelligence

SAS Foundation Credentials and Exams

SAS offers the following credentials and exams in the SAS Foundation area:

<table>
<thead>
<tr>
<th>Credentials</th>
<th>Certification Exams</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS Certified Base Programmer for SAS®9</td>
<td>SAS Base Programming for SAS®9</td>
</tr>
<tr>
<td>SAS Certified Advanced Programmer for SAS®9</td>
<td>SAS Base Programming for SAS®9</td>
</tr>
<tr>
<td></td>
<td>SAS Advanced Programming for SAS®9</td>
</tr>
</tbody>
</table>
SAS Base Programming Exam for SAS®9

The intended candidate for the SAS Base Programming Exam is someone with current SAS programming experience in the following five content areas:

1. Accessing Data
2. Creating Data Structures
3. Managing Data
4. Generating Reports
5. Handling Errors

In addition, the candidate must be familiar with the enhancements and new functionality available in SAS®9.

1. Accessing Data

- Use FORMATTED and LIST input to read raw data files.
- Use INFILE statement options to control processing when you read raw data files.
- Use various components of an INPUT statement to process raw data files, including column and line pointer controls and trailing @ controls.
- Combine SAS data sets.
- Access Microsoft Excel workbooks.
2. Creating Data Structures
- Create temporary and permanent SAS data sets.
- Create and manipulate SAS date values.
- Export data to standard and comma-delimited raw data files.
- Control which observations and variables in a SAS data set are processed and output.

3. Managing Data
- Investigate SAS libraries using utility procedures.
- Sort observations in a SAS data set.
- Use assignment statements in the DATA step.
- Modify variable attributes using options and statements in the DATA step.
- Accumulate subtotals and totals using DATA step statements.
- Use SAS functions to manipulate character data, numeric data, and SAS date values.
- Use SAS functions to convert character data to numeric and vice versa.
- Process data using DO loops and SAS arrays.
- Validate and clean data.
4. Generating Reports

- Use the PRINT procedure to generate list reports.
- Use Base SAS procedures to generate summary reports and frequency tables.
- Enhance reports through the use of labels, SAS formats, user-defined formats, titles, footnotes, and SAS reporting options.
- Use ODS statements to direct reports to external files.

5. Handling Errors

- Identify and resolve programming logic errors.
- Recognize and correct syntax errors.
- Examine and resolve data errors.
Exam Details

For up-to-date information about the SAS Base Programming exam, go to http://support.sas.com/certify.

Go to the following link to complete the exam details: http://support.sas.com/certify/creds/bp.html#t2

- Exam is administered by SAS and ____________
- ____ multiple-choice and short-answer questions
- Must achieve score of _____ correct to pass
- _______ to complete exam
- Exam taken on a computer
- Closed book
- Score received after completing the exam

Exam details are subject to change at any time.
Multiple-Choice Questions

All multiple-choice questions contain a question and four possible answers.

For example:

Which city is the headquarters for SAS?

A. Boston
B. Cary
C. Detroit
D. San Francisco

There is only one correct answer.

For example:

Which city is the headquarters for SAS?

A. Boston
B. Cary
C. Detroit
D. San Francisco
Multiple-Choice Questions

Given the following data set:

<table>
<thead>
<tr>
<th>State</th>
<th>City</th>
</tr>
</thead>
<tbody>
<tr>
<td>MA</td>
<td>Boston</td>
</tr>
<tr>
<td>NC</td>
<td>Cary</td>
</tr>
<tr>
<td>MI</td>
<td>Detroit</td>
</tr>
<tr>
<td>CA</td>
<td>San Francisco</td>
</tr>
</tbody>
</table>

What is the value of City for the second observation?

A. Boston
B. Cary
C. Detroit
D. San Francisco

Short-Answer Questions

All short-answer questions contain a question with explicit instructions for entering the answer. Support information can appear before the question.

For example:

What is the maximum length for a SAS variable name?

Enter your numeric answer.

32
1.2 Specifics about This Review Course

SAS Certification Review: Base Programming for SAS®9

This course is a review of the majority of the topics in the SAS Base Programming Exam for SAS®9. The five content areas of the exam are presented in six chapters.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SAS Fundamental Concepts</td>
</tr>
<tr>
<td>2</td>
<td>Working with SAS Data Sets</td>
</tr>
<tr>
<td>3</td>
<td>Working with Raw Data and Excel Files</td>
</tr>
<tr>
<td>4</td>
<td>Creating Variables</td>
</tr>
<tr>
<td>5</td>
<td>Manipulating Data</td>
</tr>
<tr>
<td>6</td>
<td>Generating Reports</td>
</tr>
</tbody>
</table>

This course assumes prior knowledge of the topics.

re·view
1: to view or see again
2: to examine or study again
SAS Certification Review: Base Programming for SAS®9

This course is a review of the majority of the topics in the SAS Base Programming Exam for SAS®9.

majority
1 : a number or percentage equaling more than half of a total

The certification exam can include topics beyond those topics discussed in this course.

SAS Certification Review: Base Programming for SAS®9

This review course will
- refresh your mind on some details that you might have forgotten
- include topics with which you might have limited experience
- familiarize you with SAS terminology
- determine in what areas you need to do additional studying
- help you determine whether you are ready to take the exam
- provide practice for answering SAS questions.
Questions in This Course

Different question format types are used throughout the lecture portion of this course.
- multiple choice with one correct answer (A-D)
- multiple choice with more than one correct answer (1-6)
- yes or no
- discussion
- polls

Remember: The certification exam does not use all of these question format types.

Questions in This Course

All questions in this course are identified with a [ ] in the top right corner of the slide.

Selecting Observations
The following program is submitted:

```
data work.portion;
  set sashelp.retail(firstobs=5 obs=10);
run;
```

How many observations are in the output data set?

A. 5
B. 6
C. 10
D. 14
Questions in This Course

All answers to the questions are identified with an \( \text{A} \) in the top right corner of the slide.

Selecting Observations
The following program is submitted:

```
data work.portion;
    set aashelp.retail(firstobs=5 obs=10);
run;
```

How many observations are in the output data set?

A. 5  B. 6  C. 10  D. 14

Answer slides are not in your course notes. A summary of the correct answers is located in the last section of each chapter.

Exercises in This Course (Appendix A)

Different exercise format types are used throughout this course.

- True or False
- Answer the Questions
- Match the Values
- Fill in the Blanks
- Fill in the Table
- Complete the Program
- Add a Statement
- Find the Mistakes

Some exercises are completed \textit{before} a topic is reviewed, whereas other exercises are completed \textit{after} a topic is reviewed.

Remember: The certification exam does not use all of these exercise format types.
Practice Exam in This Course (Appendix B)

The practice exam contains 50 questions in six sections.

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Questions</th>
<th>Time Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SAS Fundamental Concepts</td>
<td>5</td>
<td>9 minutes</td>
</tr>
<tr>
<td>2</td>
<td>Working with SAS Data Sets</td>
<td>9</td>
<td>15 minutes</td>
</tr>
<tr>
<td>3</td>
<td>Working with Raw Data and Excel Files</td>
<td>9</td>
<td>15 minutes</td>
</tr>
<tr>
<td>4</td>
<td>Creating Variables</td>
<td>9</td>
<td>15 minutes</td>
</tr>
<tr>
<td>5</td>
<td>Manipulating Data</td>
<td>9</td>
<td>15 minutes</td>
</tr>
<tr>
<td>6</td>
<td>Generating Reports</td>
<td>9</td>
<td>15 minutes</td>
</tr>
</tbody>
</table>

Remember: Refer to the certification website for the actual number of questions and time limit.

1.3 SAS Fundamental Concepts

Refer to Exercise 1 for Chapter 1 in Appendix A.

SAS Steps

The DATA step and the PROC (procedure) step are the two types of steps in a SAS program.

```sas
data work.enroll;
  infile 'enrollment.dat';
  input @1 first_name $8. @9 last $9. @20 state2 $2. @23 _age_ 2.
    @26 enrolldate date9.;
run;
libname project 'C:\workshop\winsas\lwcrb';
proc sort data=work.enroll
  out=project.enroll;
  by last;
run;
proc print data=project.enroll;
  var last state2 _age_ enrolldate;
  format enrolldate mmdyy10.;
run;
```

Copyright © 2013, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.
DATA Step

In general, the DATA step manipulates data.
- The input for a DATA step can be of several types, such as raw data or a SAS data set.
- The output from a DATA step can be of several types, such as a SAS data set or a report.

```
data work.enroll;
  infile 'enrollment.dat';
  input @1 first_name $8. @9 last $9.
    @20 state2 $2. @23 _age_ 2.
    @26 enrolldate date9.;
run;
```

Yes or No: Do all SAS programs contain a DATA step?
**PROC Step**
In general, the PROC step analyzes data, produces output, or manages SAS files.
- The input for a PROC step is usually a SAS data set.
- The output from a PROC step can be of several types, such as a report or an updated SAS data set.

```sas
proc sort data=work.enroll out=project.enroll;
   by last;
run;
proc print data=project.enroll;
   var last state2 _age_ enrolldate;
   format enrolldate mmddyy10.;
run;
```

**SAS Statements**
A SAS statement is a series of items that might include keywords, SAS names, special characters, and operators.

The two types of SAS statements are as follows:
- those that are used in DATA and PROC steps
- those that are global in scope and can be used anywhere in a SAS program

All SAS statements end with a semicolon.
SAS Statements

How many statements are in the PROC SORT step?

```sas
proc sort data=work.enroll
   out=project.enroll;
   by last;
run;

proc print data=project.enroll;
   var last state2 age enrolldate;
   format enrolldate mmddyy10.;
run;
```

Global Statements

Global statements
- are used anywhere in a SAS program
- stay in effect until changed or canceled, or until you end your SAS session.

```sas
libname project 'C:\workshop\winsas\lwcrb';

proc sort data=work.enroll
   out=project.enroll;
   by last;
run;
```
Global Statements

What are some additional examples of global statements?

1. DATA  
2. TITLE  
3. LABEL  
4. FORMAT  
5. OPTIONS  
6. FOOTNOTE

SAS Data Sets

A SAS data set has these characteristics:
- is a SAS file stored in a SAS library that SAS creates and processes
- contains data values that are organized as a table of observations (rows) and variables (columns)
- contains descriptor information such as the data types and lengths of the variables

<table>
<thead>
<tr>
<th>first_name</th>
<th>last</th>
<th>state</th>
<th>age</th>
<th>enrollate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Danny</td>
<td>Brown</td>
<td>CD</td>
<td>.</td>
<td>1568</td>
</tr>
<tr>
<td>William</td>
<td>Johnson</td>
<td></td>
<td>22</td>
<td>17318</td>
</tr>
<tr>
<td>Samantha</td>
<td>McCormick</td>
<td>CA</td>
<td>47</td>
<td>16674</td>
</tr>
<tr>
<td>Tina</td>
<td>Stewart</td>
<td>TX</td>
<td>53</td>
<td>14283</td>
</tr>
</tbody>
</table>
SAS Libraries

A SAS library is a collection of one or more SAS files, including SAS data sets, that are referenced and stored as a unit.

- In a directory-based operating environment, a SAS library is a group of SAS files that are stored in the same directory.
- In z/OS (OS/390), a SAS library is a group of SAS files that are stored in an operating environment file.

SAS Libraries

A logical name (libref) can be assigned to a SAS library using the LIBNAME statement.

```
libname project 'C:\workshop\winsas\lwcrb';
```

The `libref`
- can be up to 8 characters long
- must begin with a letter (A-Z) or an underscore (_)
- can contain only letters, digits (0-9), or underscores.
SAS Libraries

Which of the following sentences is true concerning the LIBNAME statement?

A. The LIBNAME statement must go in a DATA step.
B. The LIBNAME statement must end in a semicolon.
C. The LIBNAME statement must be the first statement in a program.
D. The LIBNAME statement must be followed by the RUN statement.

Two-Level SAS Data Set Names

A SAS data set can be referenced using a two-level SAS data set name.

```
proc sort data=work.enroll out=project.enroll;
```

- `libref` is the logical name that is associated with the physical location of the SAS library.
- `data-set` is the data set name, which can be up to 32 characters long, must begin with a letter or an underscore, and can contain letters, digits, and underscores.
One-Level SAS Data Set Names

A data set referenced with a one-level name is automatically assigned to the **Work** library by default. For example, the following two statements are equivalent:

```sas
proc sort data=work.enroll out=project.enroll;
proc sort data=enroll out=project.enroll;
```

Temporary and Permanent SAS Data Sets

How many of the following data sets are permanent data sets?

- `work.enroll`
- `temp.enroll`
- `project.enroll`
- `enroll`
Temporary and Permanent SAS Data Sets

A temporary SAS data set is one that exists only for the current SAS session or job.
- The Work library is a temporary data library.
- Data sets held in the Work library are deleted at the end of the SAS session.

A permanent SAS data set is one that resides on the external storage medium of your computer and is not deleted when the SAS session terminates.
- Any data library referenced with a LIBNAME statement is considered a permanent data library by default.

Variables

Data values are organized into columns called variables.

Variables have attributes, such as name and type, that enable you to identify them and that define how they can be used.
Variable Names

By default, a SAS variable name

- can be up to 32 characters long
- must begin with a letter (A-Z) or an underscore (_)
- can contain only letters, digits (0-9), or underscores.

```
first name last state2 age enroldate
```

---

Variable Names

Which of the following variable names is valid?

A. street#
B. zip_code
C. 2address
D. last name
Variable Types

The two types of SAS variables are listed below:
- character
- numeric

### Variable Types: Character

Character variables are stored with a length of 1 to 32,767 bytes with 1 character equaling 1 byte.

Character variables can contain letters (A-Z), numeric digits (0-9), and other special characters (\_, \#, \%, \\&,...).
Variable Types: Numeric

Numeric variables are stored as floating-point numbers with a default byte size of 8.

To be stored as a floating point number, the numeric value can contain numeric digits (0-9), plus or minus sign, decimal point, and E for scientific notation.

Variable Types

How should a date be stored in SAS?

A. character
B. numeric
SAS Dates

A SAS date value is a value that represents the number of days between January 1, 1960, and a specified date.
- Dates before January 1, 1960, are negative numbers.
- Dates after January 1, 1960, are positive numbers.

To reference a SAS date value in a SAS program, use a SAS date constant.
- A SAS date constant is a date (DDMMYYYY) in quotation marks followed by the letter D.
- Example: 
  
  `'12NOV1986'd`

SAS Dates

What is the numeric SAS date value for December 25, 1959?

A. -6
B. -7
C. 6
D. 8
Missing Data

Missing data is a value that indicates that no data value is stored for the variable in the current observation.

- A missing numeric value is displayed as a single period (.)
- A missing character value is displayed as a blank space.

CONTENTS Procedure

The CONTENTS procedure shows the descriptor portion of a SAS data set.

```
proc contents data=project.enroll;
run;
```

Alphabetic List of Variables and Attributes

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td><em>age</em></td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>enrolldate</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>first_name</td>
<td>Char</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>last</td>
<td>Char</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>state2</td>
<td>Char</td>
<td>2</td>
</tr>
</tbody>
</table>

The VARNUM option can be used to print the variable list in the order of the variables' positions in the data set.
CONTENTS Procedure

Which step displays the directory of the Project library and suppresses printing the contents of individual data sets?

A. `proc contents data=project; run;`
B. `proc contents data=project.all; run;`
C. `proc contents data=project nocontents; run;`
D. `proc contents data=project._all_.nods; run;`

PRINT Procedure

The PRINT procedure can show the data portion of a SAS data set.

```
proc print data=project.enroll;
run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>first_</th>
<th>name</th>
<th>last</th>
<th>state2</th>
<th><em>age</em></th>
<th>enrolldate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Danny</td>
<td>Brown</td>
<td>CO</td>
<td>.</td>
<td>15684</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>William</td>
<td>Johnson</td>
<td></td>
<td>22</td>
<td>17318</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Samantha</td>
<td>McCormick</td>
<td>CA</td>
<td>47</td>
<td>16674</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Tina</td>
<td>Stewart</td>
<td>TX</td>
<td>53</td>
<td>14287</td>
<td></td>
</tr>
</tbody>
</table>
CONTENTS and PRINT Procedures

Which program has the correct syntax for the CONTENTS and PRINT procedures?

A.  
   proc contents project.enroll;
   run;
   proc print project.enroll;
   run;

B.  
   proc contents data=project.enroll
   run;
   proc print data=project.enroll
   run;

C.  
   proc contents data=project.enroll;
   run;
   proc print data=project.enroll;
   run;

Comments

There are two ways to add comments in a SAS program.

*The SORT procedure is creating a data set;
   proc sort data=work.enroll
      out=project.enroll;
      by last;
   run;

/* The PRINT procedure is creating a report. */
   proc print data=project.enroll;
      var last state2 age enrolldate;
      format enrolldate mmddyy10.;
   run;

During processing, SAS ignores text in comments.
SAS Log

The SAS log is a record of your submitted SAS program.

125 libname project 'C:\workshop\winsas\lwcrb\data';
NOTE: Libref PROJECT was successfully assigned as follows:
        Engine:   V9
        Physical Name: C:\workshop\winsas\lwcrb\data
126 proc sort data=work.enroll;
127    out=project.enroll;
128    by last;
129 run;
NOTE: There were 4 observations read from the data set WORK.ENROLL.
NOTE: The data set PROJECT.ENROLL has 4 observations and 5 variables.

- Original program statements are identified by line numbers.
- SAS messages can include the words NOTE, INFO, WARNING, or ERROR.

SAS Log

What are the issues with the following program based on the SAS log?

154 proc content data=project.enroll;
ERROR: Procedure CONTENT not found.
156 run;
NOTE: The SAS System stopped processing this step because of errors.
156 proc print project.enroll;
---------------
22            200
ERROR 22-322: Syntax error, expecting one of the following: ;, DATA,
            DOUBLE, HEADING, LABEL, N, NOOBS, OBS, ROUND, ROWS,
            SPLIT, STYLE, UNIFORM, WIDTH.
ERROR 200-322: The symbol is not recognized and will be ignored.
157 run;
NOTE: The SAS System stopped processing this step because of errors.
# 1.4 Answers to Questions

<table>
<thead>
<tr>
<th>Question Slide Number</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>No</td>
</tr>
<tr>
<td>43</td>
<td>three statements</td>
</tr>
<tr>
<td>46</td>
<td>2., 5., and 6.</td>
</tr>
<tr>
<td>51</td>
<td>B.</td>
</tr>
<tr>
<td>55</td>
<td>two permanent data sets (<em>temp.enroll</em> and <em>project.enroll</em>)</td>
</tr>
<tr>
<td>60</td>
<td>B.</td>
</tr>
<tr>
<td>65</td>
<td>B.</td>
</tr>
<tr>
<td>68</td>
<td>B.</td>
</tr>
<tr>
<td>72</td>
<td>D.</td>
</tr>
<tr>
<td>75</td>
<td>C.</td>
</tr>
</tbody>
</table>
| 79                    | • CONTENTS misspelled  
                        • DATA= missing |
Chapter 2  Working with SAS Data Sets

2.1  Reading and Creating Data Sets ................................................................. 2-3
2.2  Selecting Observations .............................................................................. 2-29
2.3  Sorting Observations with the SORT Procedure ...................................... 2-42
2.4  Combining Data Sets .................................................................................. 2-45
2.5  Answers to Questions .................................................................................. 2-57
2.1 Reading and Creating Data Sets

**Reading and Creating Data Sets**

```sas
data work.newprice;
  set golf.supplies;
  <additional programming statements>
run;
```

1. What is the name of the data set being read?

2. What is the name of the data set being created?

**DATA Statement**

The DATA statement begins a DATA step and provides names for any output SAS data sets that are created.

```sas
data work.newprice;  // output data set
  set golf.supplies;
  <additional programming statements>
run;
```

- The DATA statement can create temporary or permanent data sets.
SET Statement

The SET statement reads an observation from one or more SAS data sets for further processing in the DATA step.

```
data work.newprice;
  set golf.supplies;  input data set
  <additional programming statements>
run;
```

- By default, the SET statement reads all variables and all observations from the input data sets.
- The SET statement can read temporary or permanent data sets.

Additional Programming Statements

Additional programming statements can be added to perform further processing in the DATA step.

For example, an assignment statement can be added to create a new variable based on an expression.

```
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;  creates the variable
run;                    saleprice based on the price
```

variable from the golf.supplies data set
2.1 Reading and Creating Data Sets

Additional Programming Statements

data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;

How many variables are in the output data set work.newprice?

A. 3  
B. 4  
C. 5  
D. 6  

DATA Step Execution

When a DATA step is submitted, it is first compiled and then executed.

```
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;
```

<table>
<thead>
<tr>
<th>Model</th>
<th>Height</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>8.1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>8.35</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>9.1</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>13.71</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>14.4</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>10.6</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>12.7</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>14.5</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>15.3</td>
</tr>
</tbody>
</table>
Compilation Phase

During the compilation phase, SAS does the following:
- checks the syntax of the SAS statements
- translates the statements into machine code
- identifies the name, type, and length of each variable

The following three items are potentially created:
- input buffer
- program data vector (PDV)
- descriptor information

Input Buffer

The *input buffer* is a logical area in memory into which SAS reads each record of a raw data file when SAS executes an INPUT statement.
- This buffer is created only when the DATA step reads raw data.
- When the DATA step reads a SAS data set, SAS reads the data directly into the program data vector.
Program Data Vector (PDV)

The program data vector is a logical area in memory where SAS builds a data set, one observation at a time.

Along with data set variables and computed variables, the PDV contains the following two automatic variables:

- the _N_ variable, which counts the number of times the DATA step begins to iterate
- the _ERROR_ variable, which signals the occurrence of an error caused by the data during execution

The value of _ERROR_ is either 0 (indicating no errors exist) or 1 (indicating that one or more errors occurred).

Program Data Vector (PDV)

Which of the following statements is **false** concerning the _N_ and _ERROR_ variables?

A. SAS does not write the _N_ and _ERROR_ variables to the output data set.
B. SAS increments the _N_ variable by 1 for each iteration of the DATA step.
C. SAS automatically generates the _N_ and _ERROR_ variables for every DATA step.
D. SAS sets the _ERROR_ variable equal to the total number of errors caused by the data during execution.
Program Data Vector (PDV)

data work.newprice;
set golf.supplies;
saleprice=price*0.75;
run;

Program Data Vector (PDV):

<table>
<thead>
<tr>
<th>mfg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>$ 6</td>
<td>$ 8</td>
<td>N 8</td>
<td>N 8</td>
<td>N 8</td>
<td>N 8</td>
</tr>
</tbody>
</table>

variables from
golf.supplies
data set

variables from
assignment
statement

automatic
variables

Which of the following is not one of the items in the PDV at compile time?

A. byte size of the variable
B. initial value of the variable
C. name of the variable
D. type (character or numeric) of the variable
Descriptor Information

The descriptor portion is information that SAS creates and maintains about each SAS data set, including data set attributes and variable attributes.

Examples of descriptor information include the following:
- the name of the data set
- the date and time that the data set was created
- the names, data types (character or numeric), and lengths of the variables

```
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;
```

Partial Descriptor Information (Variable Attributes)

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mfg</td>
<td>Char</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>price</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>saleprice</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>type</td>
<td>Char</td>
<td>8</td>
</tr>
</tbody>
</table>
Execution Phase

During the execution phase, SAS does the following:
- initializes the PDV to missing and sets the initial values of _N_ and _ERROR_
- reads data values into the PDV
- executes any subsequent programming statements
- outputs the observation to a SAS data set
- returns to the top of the DATA step
- resets the PDV to missing for any variables not read directly from a data set and increments _N_ by 1
- repeats the process until the end-of-file is detected

This is the default flow of the DATA step execution phase. Options and the placement of statements can alter this flow.
Execution Phase

- initializes the PDV to missing and sets the initial values of \_N\_ and \_ERROR\_.

```plaintext
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;
```

Program Data Vector (PDV)

<table>
<thead>
<tr>
<th>mfg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>$6</td>
<td>$8</td>
<td>N 8</td>
<td>N 8</td>
<td>N 8</td>
<td>N 8</td>
</tr>
</tbody>
</table>

...
Execution Phase

- executes any subsequent programming statements

```sas
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;
```

Program Data Vector (PDV)

<table>
<thead>
<tr>
<th>mfg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>$6</td>
<td>$8</td>
<td>N 8</td>
<td>N 8</td>
<td>N 8</td>
<td>N 8</td>
</tr>
<tr>
<td>Crew</td>
<td>Distance</td>
<td>8.1</td>
<td>6.075</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Execution Phase

- outputs the observation to a SAS data set

```sas
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;
```

Program Data Vector (PDV)

<table>
<thead>
<tr>
<th>mfg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>$6</td>
<td>$8</td>
<td>N 8</td>
<td>N 8</td>
<td>N 8</td>
<td>N 8</td>
</tr>
<tr>
<td>Crew</td>
<td>Distance</td>
<td>8.1</td>
<td>6.075</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
2.1 Reading and Creating Data Sets

**Execution Phase**
- returns to the top of the DATA step

```sas
data work.newprice;
set golf.supplies;
saleprice=price*0.75;
run;
```

**Program Data Vector (PDV)**

<table>
<thead>
<tr>
<th>mfg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Crew</td>
<td>Distance</td>
<td>8.1</td>
<td>6.075</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

---

**Execution Phase**
- resets the PDV to missing for any variables not read directly from a data set and increments _N_ by 1

```sas
data work.newprice;
set golf.supplies;
saleprice=price*0.75;
run;
```

**Program Data Vector (PDV)**

<table>
<thead>
<tr>
<th>mfg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Crew</td>
<td>Distance</td>
<td>8.1</td>
<td>.</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
Execution Phase
- reads data values into the PDV

```sas
data work.newprice;
set golf.supplies;
saleprice=price*0.75;
run;
```

Program Data Vector (PDV)

```
<table>
<thead>
<tr>
<th>mfg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Crew</td>
<td>Spin</td>
<td>8.25</td>
<td>.</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
```

Execution Phase
- executes any subsequent programming statements

```sas
data work.newprice;
set golf.supplies;
saleprice=price*0.75;
run;
```

Program Data Vector (PDV)

```
<table>
<thead>
<tr>
<th>mfg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Crew</td>
<td>Spin</td>
<td>8.25</td>
<td>6.1875</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
```
Execution Phase

- outputs the observation to a SAS data set

```
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;
```

Program Data Vector (PDV)

<table>
<thead>
<tr>
<th>mfg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Crew</td>
<td>Spin</td>
<td>8.25</td>
<td>6.1875</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Execution Phase

- returns to the top of the DATA step

```
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;
```

Program Data Vector (PDV)

<table>
<thead>
<tr>
<th>mfg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Crew</td>
<td>Spin</td>
<td>8.25</td>
<td>6.1875</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>
Execution Phase

- repeats the process until the end of file is detected

```sas
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;
```

Program Data Vector (PDV)

<table>
<thead>
<tr>
<th>mfg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Crew</td>
<td>Spin</td>
<td>8.25</td>
<td>6.1875</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

How many times does SAS iterate through the DATA step?

A. 0
B. 1
C. 9
D. 10
Execution Phase

Next steps:
- Eliminate variables.
- Add dollar signs.
- Add column headings.
- Use the debugger.

```sas
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;
```

### DROP and KEEP Statements

- The DROP statement specifies the names of the variables to omit from the output data set.
- The KEEP statement specifies the names of the variables to write to the output data set.

```sas
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
  drop mfg price;
run;
```

Placement of statement is irrelevant; statement is applied at output time.
2-18 Chapter 2  Working with SAS Data Sets

**DROP and KEEP Statements**

```sas
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
  drop mfg price;
run;
```

Which KEEP statement is equivalent to the DROP statement in the above program?

A. `keep type;`
B. `keep type saleprice;`

**FORMAT and LABEL Statements**

- The FORMAT statement associates formats to variable values.
- The LABEL statement assigns descriptive labels to variable names.

```sas
data work.newprice;
  set golf.supplies;
  saleprice = price * 0.75;
  drop mfg price;
  format saleprice dollar18.2;
  label type='Type of Ball'
    saleprice='Sale Price';
run;
```
FORMAT and LABEL Statements

\begin{verbatim}
proc contents data=work.newprice;
run;
\end{verbatim}

Alphabetic List of Variables and Attributes

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>saleprice</td>
<td>Num</td>
<td>8</td>
<td>DOLLAR18.2</td>
<td>Sale Price</td>
</tr>
<tr>
<td>1</td>
<td>type</td>
<td>Char</td>
<td>8</td>
<td></td>
<td>Type of Ball</td>
</tr>
</tbody>
</table>

- FORMAT and LABEL statements assigned in a DATA step are considered \textit{permanent} attributes (stored in the descriptor portion).

\begin{verbatim}
proc print data=work.newprice label;
run;
\end{verbatim}

<table>
<thead>
<tr>
<th>Obs</th>
<th>Type of Ball</th>
<th>Sale Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Distance</td>
<td>$6.08</td>
</tr>
<tr>
<td>2</td>
<td>Spin</td>
<td>$6.19</td>
</tr>
<tr>
<td>3</td>
<td>Titanium</td>
<td>$7.13</td>
</tr>
<tr>
<td>4</td>
<td>X12000</td>
<td>$10.31</td>
</tr>
<tr>
<td>5</td>
<td>X22000</td>
<td>$10.95</td>
</tr>
<tr>
<td>6</td>
<td>Strata</td>
<td>$7.95</td>
</tr>
<tr>
<td>7</td>
<td>Aero</td>
<td>$9.23</td>
</tr>
<tr>
<td>8</td>
<td>XL</td>
<td>$10.88</td>
</tr>
<tr>
<td>9</td>
<td>Flite</td>
<td>$12.15</td>
</tr>
</tbody>
</table>
DATA Step Debugger

How is the DATA step debugger invoked?

A. adding a / DEBUG option to the DATA statement
B. adding a DEBUG statement after the DATA statement
C. adding the DEBUG option to the OPTIONS statement
D. adding the DEBUG=YES option to the OPTIONS statement

DATA Step Debugger

The DATA step debugger consists of windows and a group of commands that provide an interactive way to identify logic and data errors in DATA steps.

Debugger Commands: STEP, EXAMINE, WATCH, and QUIT
PUTLOG Statement

The PUTLOG statement can be used to write messages to the SAS log to help identify logic errors.

```sas
data work.newprice;
  putlog 'Top of Step';
  set golf.supplies;
  putlog type $quote12. price=;
  saleprice = price * 0.75;
  putlog _all_; 
  drop mfg price;
run;
```

**Top of Step**

```
"Distance" price=8.1
mfg=Crew type=Distance price=8.1 saleprice=6.075 _ERROR_=0 _N_=1
Top of Step
"Spin"  price=8.25
mfg=Crew type=Spin price=8.25 saleprice=6.1875 _ERROR_=0 _N_=2
...```

Refer to Exercise 1 for Chapter 2 in Appendix A.

Implicit Output

By default, at the end of each iteration, every DATA step contains an implicit OUTPUT statement that tells SAS to write observations to the data set or data sets that are being created.

```sas
data work.total;
  set work.scores;
  total=test1+test2;
run;
```

**Input Data Set**

```
name   test1   test2
Kent   73      73
Mary   89      89
Sally  75      75
Thomas 92     92
```

**Output Data Set**

```
name   test1   test2   total
Kent   73      73      146
Mary   89      89      178
Sally  75      85      160
Thomas 92     92      184
```
Implicit Output

Yes or No: Will the following two programs produce the same results?

```
data work.total;
  set work.scores;
  total=test1+test2;
run;
```

```
data work.total;
  set work.scores;
  total=test1+test2;
  output;
run;
```

OUTPUT Statement

The OUTPUT statement without arguments causes the current observation to be written to all data sets that are named in the DATA statement.

```
data work.total;
  set work.scores;
  total=test1+test2;
 output;
run;
```
OUTPUT Statement

Multiple OUTPUT statements can be used in a DATA step.

```sas
data work.rotate; 
  set work.scores; 
  test=test1; 
  output; 
  test=test2; 
  output; 
  drop test1 test2; 
run;
```

How many observations are in the final data set, `work.rotate`?

A. 0  
B. 2  
C. 4  
D. 8
OUTPUT Statement

Placing an explicit OUTPUT statement in a DATA step overrides the implicit output, and SAS adds an observation to a data set only when an explicit OUTPUT statement is executed.

data work.rotate;
set work.scores;
  test=test1;
output;
  test=test2;
drop test1 test2;
run;

Creating Multiple Data Sets

- The DATA statement can specify multiple output data sets.
- The OUTPUT statement can specify the data set names.

data work.first
  work.second;
set work.scores;
  test=test1;
output work.first;
  test=test2;
output work.second;
drop test1 test2;
rush;
2.1 Reading and Creating Data Sets

Creating Multiple Data Sets

```sas
data work.first
  work.second;
set work.scores;
  test=test1;
output work.first;
  test=test2;
output work.second;
  drop test1 test2;
run;
```

Input Data Set

<table>
<thead>
<tr>
<th>name</th>
<th>test1</th>
<th>test2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kent</td>
<td>73</td>
<td>75</td>
</tr>
<tr>
<td>Mary</td>
<td>69</td>
<td>94</td>
</tr>
<tr>
<td>Sally</td>
<td>75</td>
<td>86</td>
</tr>
<tr>
<td>Thomas</td>
<td>92</td>
<td>96</td>
</tr>
</tbody>
</table>

Output Data Set

<table>
<thead>
<tr>
<th>name</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kent</td>
<td>73</td>
</tr>
<tr>
<td>Mary</td>
<td>69</td>
</tr>
<tr>
<td>Sally</td>
<td>75</td>
</tr>
<tr>
<td>Thomas</td>
<td>92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kent</td>
<td>75</td>
</tr>
<tr>
<td>Mary</td>
<td>86</td>
</tr>
<tr>
<td>Sally</td>
<td>94</td>
</tr>
<tr>
<td>Thomas</td>
<td>96</td>
</tr>
</tbody>
</table>

Creating Multiple Data Sets

```sas
data work.total work.first work.second;
set work.scores;
  total=test1+test2;
output;
run;
```

Which data set(s) does the OUTPUT statement populate?

A. work.total
B. work.first
C. work.second
D. work.total, work.first, and work.second
Creating Multiple Data Sets

Using the OUTPUT statement without arguments causes the current observation to be written to all data sets that are named in the DATA statement.

```sas
data work.total
   work.first
   work.second;
set work.scores;
total=test1+test2;
output;
drop test1 test2;
run;
```

The DROP and KEEP statements apply to all output data sets.

```
data work.total
   work.first
   work.second;
set work.scores;
total=test1+test2;
output work.total;
test=test1;
output work.first;
test=test2;
output work.second;
drop test1 test2;
run;
```
**DROP= and KEEP= Options**

- The DROP= data set option excludes the variables for writing to a specific output data set.
- The KEEP= data set option specifies the variables for writing to a specific output data set.

```sas
data work.total(keep=name total test1 test2)
  work.first(drop=test1 test2)
  work.second(keep=name total test);
set work.scores;
total=test1+test2;
output work.total;
test=test1;
output work.first;
test=test2;
output work.second;
run;
```

```
data work.total(keep=name total test1 test2)
  work.first(drop=test1 test2)
  work.second(keep=name total test);
set work.scores;
total=test1+test2;
output work.total;
test=test1;
output work.first;
test=test2;
output work.second;
run;
```

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Total</th>
<th>Output Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>total</td>
</tr>
<tr>
<td>Kent</td>
<td>152</td>
</tr>
<tr>
<td>Mary</td>
<td>103</td>
</tr>
<tr>
<td>Sally</td>
<td>161</td>
</tr>
<tr>
<td>Thomas</td>
<td>187</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.First</th>
<th>Output Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>total</td>
</tr>
<tr>
<td>Kent</td>
<td>152</td>
</tr>
<tr>
<td>Mary</td>
<td>103</td>
</tr>
<tr>
<td>Sally</td>
<td>161</td>
</tr>
<tr>
<td>Thomas</td>
<td>187</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Second</th>
<th>Output Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>total</td>
</tr>
<tr>
<td>Kent</td>
<td>152</td>
</tr>
<tr>
<td>Mary</td>
<td>103</td>
</tr>
<tr>
<td>Sally</td>
<td>161</td>
</tr>
<tr>
<td>Thomas</td>
<td>187</td>
</tr>
</tbody>
</table>
Other Statements Using the OUTPUT Statement

The OUTPUT statement can stand alone or be part of an IF-THEN or SELECT/WHEN statement or be in DO loop processing. 

Example with the IF-THEN statement:

```sas
data female
  male
  all(keep=name weight height);
set sashelp.class;
if sex='F' then output female all;
else if sex='M' then output male all;
run;
```

- Multiple data sets can be specified in the OUTPUT statement.

---

How many variables are in the output data set female?

A. 2
B. 3
C. 5
D. 6

Refer to Exercise 2 for Chapter 2 in Appendix A.
2.2 Selecting Observations

Selecting Observations
By default, all observations of the input data set are written to the output data set.

```
data work.all;
  set sashelp.retail;
run;
```

Input data set `sashelp.retail` has 58 observations.  Output data set `work.all` has 58 observations.

Selecting Observations
The FIRSTOBS= and OBS= data set options can be used to control which observations are read from the input data set.

```
data work.ten;
  set sashelp.retail(obs=10);
run;
```

Input data set `sashelp.retail` has 58 observations.  Output data set `work.ten` has 10 observations.

FIRSTOBS= and OBS= are valid for input processing only. That is, they are not valid for output processing.
Selecting Observations

The following program is submitted:

```sas
data work.portion;
  set sashelp.retail(firstobs=5 obs=10);
run;
```

How many observations are in the output data set?

A. 5  
B. 6  
C. 10  
D. 14

FIRSTOBS= and OBS= Options

- The FIRSTOBS= data set option specifies a starting point for processing an input data set.
- The OBS= data set option specifies an ending point for processing an input data set.

```sas
data work.portion;
  set sashelp.retail(firstobs=5 obs=10);
run;
```

Input data set `sashelp.retail` has 58 observations.

Output data set `work.portion` has 6 observations (obs # 5, 6, 7, 8, 9, and 10).

The OBS= option specifies the number of the last observation, and not how many observations there are to process.
2.2 Selecting Observations

**FIRSTOBS= and OBS= Options**

Which step has invalid syntax?

A. ```
data shoes(firstobs=101 obs=200);
set sashelp.shoes;
run;
```  

B. ```
data shoes;
set sashelp.shoes
(firstobs=101 obs=200);
run;
```  

C. ```
proc print data=sashelp.shoes
(firstobs=101 obs=200);
run;
```  

Selecting Observations Based on an Expression

The following statements can be used to select observations based on an expression:

- WHERE statement
- subsetting IF statement
- IF-THEN DELETE statement

All three of the statements reference an *expression*. 
Expression

An expression is a sequence of operands and operators that forms a set of instructions that define a condition for selecting observations.

- **Operands** are the following:
  - constants (character or numeric)
  - variables (character or numeric)
  - SAS functions
- **Operators** are symbols that request a comparison, logical operation, or arithmetic calculation.

Operands

- A *constant* is a fixed value such as a number, quoted character string, or date constant.
  - If the value is numeric, do not use quotation marks.
  - If the value is character, use quotation marks.
  - A SAS date constant is a date (DDMMYYYY) in quotation marks followed by the letter D.

- A *variable* is a variable coming from a data set, a variable created in an assignment statement, or an automatic variable created by the DATA step.
- A SAS *function* is a routine that performs a computation or system manipulation on arguments and returns a value.
**Comparison Operators**

Comparison operators compare a variable with a value or with another variable.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>EQ</td>
<td>= equal to</td>
</tr>
<tr>
<td>NE</td>
<td>^= ^= ¬= not equal to</td>
</tr>
<tr>
<td>GT</td>
<td>&gt; greater than</td>
</tr>
<tr>
<td>GE</td>
<td>&gt;= greater than or equal to</td>
</tr>
<tr>
<td>LT</td>
<td>&lt; less than</td>
</tr>
<tr>
<td>LE</td>
<td>&lt;= less than or equal to</td>
</tr>
<tr>
<td>IN</td>
<td>= equal to one of a list</td>
</tr>
</tbody>
</table>

**Operands and Comparison Operators**

Which of the following is **not** a valid expression?

A. qtr1 <= qtr2
B. address = ''
C. sales gt 6400
D. name ne Mary Ann
### Logical Operators

*Logical operators* combine or modify expressions.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>&amp; logical and</td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>^ logical not</td>
</tr>
</tbody>
</table>

### Arithmetic Operators

*Arithmetic operators* indicate that an arithmetic calculation is performed.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>exponentiation</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
</tbody>
</table>

If a missing value is an operand for an arithmetic operator, the result is a missing value.
Logical and Arithmetic Operators

Which of the following is *not* a valid expression?

A. \( X \times 5 / A - C \equiv Y \times 2 \)
B. `level = 'up' | type = 'low'`
C. `january + february le 90000`
D. `salary > 50000 title not = 'Manager'`

Special WHERE Operators

The WHERE statement can use special WHERE operators.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BETWEEN – AND</td>
<td>an inclusive range</td>
</tr>
<tr>
<td>CONTAINS</td>
<td>? a character string</td>
</tr>
<tr>
<td>LIKE</td>
<td>a character pattern</td>
</tr>
<tr>
<td>SOUNDS LIKE</td>
<td>=* spelling variation</td>
</tr>
<tr>
<td>IS NULL</td>
<td>missing value</td>
</tr>
<tr>
<td>IS MISSING</td>
<td>missing value</td>
</tr>
<tr>
<td>SAME AND ALSO</td>
<td>augments an expression</td>
</tr>
</tbody>
</table>
Special WHERE Operators

\[ \text{name like 'M__k\%'} \]

Which names will be selected based on the above expression?

1. Mark
2. Marcia
3. Mickey
4. Matthew
5. Michael

Expression Examples

- sales > 100000
- sales eq .
- name = 'Smith'
- name = '
- sales gt 100000 and name = 'Smith'
- sales gt 100000 or name = 'Smith'
- revenue >= 150 and revenue <= 999
- revenue between 150 and 999
- revenue not between 150 and 999
- month contains 'uary'
- birthdate > '11JUL1968'd
- upcase(state) = 'TX'
Selecting Observations Based on an Expression

```
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;
```

Which statement must be added to the above program to create an output data set with observations having `saleprice` greater than $10?

A. `if not (saleprice>10) then delete;`
B. `if saleprice>10;`
C. Either statement will work.

---

Selecting Observations Based on an Expression

```
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
run;
```

Which statement must be added to the above program to create an output data set with observations having `saleprice` greater than $10?

A. `where saleprice>10;`
B. `if saleprice>10;`
C. Either statement will work.
Selecting Observations Based on an Expression

There are three ways to select an observation based on an expression:

- `where expression;`
- `if expression;`
- `if not (expression) then delete;`

WHERE Statement

The WHERE statement causes the DATA step to process only those observations from a data set that meet the condition of the expression.

```sas
data work.newprice;
  set golf.supplies;
  where mfg='White';
  saleprice=price*0.75;
  if saleprice > 10;
  run;
```

The expression in the WHERE statement:
- can reference variables that are from the input data set
- cannot reference variables created from an assignment statement or automatic variables (`_N_` or `_ERROR_`).
WHERE Statement

```
data subset;
  set sales;
  difference=actual-predict;
  <insert statement here>
run;
```

Variables in Input Data Set

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DATE</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>STATE</td>
<td>Char</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>PRODUCT</td>
<td>Char</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>ACTUAL</td>
<td>Num</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>PREDICT</td>
<td>Num</td>
<td>8</td>
</tr>
</tbody>
</table>

Which WHERE statement will create an error when submitted, if inserted in the above program?

A. `where actual > predict;`
B. `where difference ge 1000;`
C. `where product in ('CHAIR','SOFA');`
D. `where state='Texas' and date<'01JAN1998';`

Subsetting IF Statement

The subsetting IF statement causes the DATA step to continue processing only those observations in the program data vector that meet the condition of the expression.

```
data work.newprice;
  set golf.supplies;
  saleprice=price*0.75;
  if saleprice > 10;
run;
```
Subsetting IF Statement

- If the expression is **true** for the observation, SAS continues to execute the remaining statements in the DATA step, including the implicit OUTPUT statement at the end of the DATA step. The resulting SAS data set (or data sets) contains a subset of the original SAS data set.

- If the expression is **false**, no further statements are processed for that observation, the current observation is not written to the data set, the remaining program statements in the DATA step are not executed, and SAS immediately returns to the beginning of the DATA step.

Which program will create an error when submitted?

A. ```
   data subset;
   set sales;
   if difference < 500;
   difference=actual-predict;
   if state='Texas';
   run;
```  

B. ```
   data subset;
   set sales;
   difference=actual-predict;
   if difference between 500 and 1000;
   run;
```  

C. ```
   data subset;
   set sales;
   difference=actual-predict;
   if product='CHAIR' and difference ge 100;
   run;
```
WHERE Statement versus Subsetting IF Statement

- The WHERE statement selects observations before they are brought into the program data vector.
- The subsetting IF statement selects observations that were read into the program data vector.

```sas
data work.newprice;
set golf.supplies;
where mfg='White';
saleprice=price*0.75;
if saleprice > 10;
run;
```

IF-THEN DELETE Statement

The IF-THEN DELETE statement causes the DATA step to stop processing those observations in the program data vector that meet the condition of the expression.

```sas
data work.newprice;
set golf.supplies;
   saleprice=price*0.75;
if saleprice <= 10 then delete;
run;
```

If the expression is **true** for the observation, the current observation is not written to a data set, and SAS returns immediately to the beginning of the DATA step for the next iteration.

Refer to Exercise 3 for Chapter 2 in Appendix A.
2.3 Sorting Observations with the SORT Procedure

**SORT Procedure**

The SORT procedure does the following:
- orders SAS data set observations by the values of one or more character or numeric variables
- either replaces the original data set or creates a new data set
- produces only an output data set, but no report
- arranges the data set by the values in ascending order by default

```sas
proc sort data=sashelp.shoes
    out=shoes;
    by descending region product;
run;
```

Refer to Exercise 4 for Chapter 2 in Appendix A.

**PROC SORT Statement**

Examples:

```sas
proc sort data=sashelp.shoes;
```

```sas
proc sort data=sashelp.shoes
    out=shoes;
```

```sas
proc sort data=sashelp.shoes
    out=sasuser.sort;
```

- The DATA= option identifies the input SAS data set.
- The OUT= option names the output data set.
- Without the OUT= option, the SORT procedure overwrites the original data set.
2.3 Sorting Observations with the SORT Procedure

BY Statement

The BY statement specifies the sorting variables.

Examples:

- by region;
- by region product;
- by region subsidiary product;

PROC SORT first arranges the data set by the values of the first BY variable.
PROC SORT then arranges any observations that have the same value of the first BY variable by the values of the second BY variable.
This sorting continues for every specified BY variable.

BY Statement

By default, the SORT procedure orders the values by ascending order.
The DESCENDING option reverses the sort order for the variable that immediately follows in the statement.

Examples:

- by region descending product;
- by descending region product;
- by descending region descending product;
BY Statement

Yes or No: Will the following program run successfully?

```sas
proc sort data=sashelp.shoes
  out=shoes;
  by descending region ascending product;
run;
```

BY Statement

In addition to the SORT procedure, a BY statement can be used in the DATA step and other PROC steps. The data sets used in the DATA step and other PROC steps must be sorted by the values of the variables that are listed in the BY statement or have an appropriate index.

```sas
proc sort data=personnel;
  by descending empid lastname;
run;
proc print data=personnel;
  by descending empid;
run;
proc sort data=one;
  by id;
run;
proc sort data=two;
  by id;
run;
data both;
  merge one two;
  by id;
run;
```
2.4 Combining Data Sets

**BY Statement**

What are the two problems associated with the following program?

```sas
proc sort data=sashelp.shoes
   out=shoes;
   by descending region product;
run;

data new;
   set sashelp.shoes;
   by region product;
run;
```

**Concatenating**

If more than one data set name appears in the SET statement, the resulting output data set is a concatenation of all the data sets that are listed.

```sas
data work.thirdqtr;
set work.oct
   work.nov
   work.dec;
run;
```

SAS reads all observations from the first data set, then all from the second data set, and so on, until all observations from all the data sets are read.
Concatenating

How many observations and variables are in the output data set `work.thirdqtr`?

```
data work.thirdqtr;
  set work.oct
  work.nov
  work.dec;
run;
```

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Oct</th>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>date</td>
</tr>
<tr>
<td>East</td>
<td>17075</td>
</tr>
<tr>
<td>West</td>
<td>17075</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Nov</th>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>date</td>
</tr>
<tr>
<td>East</td>
<td>17106</td>
</tr>
<tr>
<td>West</td>
<td>17106</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Dec</th>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>date</td>
</tr>
<tr>
<td>East</td>
<td>17136</td>
</tr>
<tr>
<td>West</td>
<td>17136</td>
</tr>
</tbody>
</table>

Concatenating

How many observations and variables are in the output data set `work.thirdqtr1`?

```
data work.thirdqtr1;
  set work.octl
  work.novl
  work.decl;
run;
```

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Octl</th>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>sales</td>
</tr>
<tr>
<td>East</td>
<td>160000</td>
</tr>
<tr>
<td>West</td>
<td>210000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Novl</th>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>sales</td>
</tr>
<tr>
<td>East</td>
<td>250000</td>
</tr>
<tr>
<td>West</td>
<td>260000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Decl</th>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>sales</td>
</tr>
<tr>
<td>East</td>
<td>310000</td>
</tr>
<tr>
<td>West</td>
<td>300000</td>
</tr>
</tbody>
</table>
Concatenating

At compile time, SAS puts the variable information from the first data set into the PDV, and then puts the variable information from the second into the PDV, and so on.

```
data company;
  set divisionA divisionB;
run;
```

**divisionA**

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name</td>
<td>Char</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>state</td>
<td>Char</td>
<td>2</td>
</tr>
</tbody>
</table>

**divisionB**

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name</td>
<td>Char</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>location</td>
<td>Char</td>
<td>2</td>
</tr>
</tbody>
</table>

Next steps:
- Control byte size of name.
- Combine state and location.

**Output Data Set**

```
VIEWTABLE: Work.Company

<table>
<thead>
<tr>
<th>name</th>
<th>state</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joy</td>
<td>SC</td>
<td></td>
</tr>
<tr>
<td>Tony</td>
<td>NC</td>
<td></td>
</tr>
<tr>
<td>Michael</td>
<td>GA</td>
<td></td>
</tr>
<tr>
<td>Margaret</td>
<td>FL</td>
<td></td>
</tr>
<tr>
<td>Kelly</td>
<td>CA</td>
<td></td>
</tr>
<tr>
<td>Roger</td>
<td>NV</td>
<td></td>
</tr>
<tr>
<td>Maynana</td>
<td>TX</td>
<td></td>
</tr>
</tbody>
</table>
```

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name</td>
<td>Char</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>state</td>
<td>Char</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>location</td>
<td>Char</td>
<td>2</td>
</tr>
</tbody>
</table>
LENGTH Statement

The LENGTH statement specifies the number of bytes for storing variables.

```sas
data company;
  length name $ 15;
  set divisionA divisionB;
run;
```

Output Data Set

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>name</td>
<td>Char</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>state</td>
<td>Char</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>location</td>
<td>Char</td>
<td>2</td>
</tr>
</tbody>
</table>

LENGTH Statement

Yes or No: Will you get the same results if the LENGTH statement is after the SET statement?

```sas
data company;
  set divisionA divisionB;
  length name $ 15;
run;
```
2.4 Combining Data Sets

RENAME= Option

The RENAME= data set option changes the names of variables.

```
data company;
  length name $ 15;
  set divisionA(rename=(state=location))
      divisionB;
run;
```

- The RENAME= option specifies the variable that you want to rename equal to the new name of the variable.
- The list of variables to rename must be enclosed in parentheses.

RENAME= Option

Which of the following statements has proper syntax for the RENAME= option?

A. `set divisionA(rename=name=first, state=location)
     divisionB(rename=name=first);`

B. `set divisionA(rename=(name=first
    state=location))
     divisionB(rename=(name=first));`

C. `set divisionA(rename=(name=first)
    (state=location))
     divisionB(rename=(name=first));`

D. `set divisionA(rename=(name=first),
    (state=location))
     divisionB(rename=(name=first));`
Interleaving

Use a single SET statement with multiple data sets and a BY statement to interleave the specified data sets.

```sas
data company;
  length name $ 15;
  set divisionA(rename=(state=location)) divisionB;
  by name;
run;
```

The observations in the new data set are arranged by the values of the BY variable or variables. Then, within each BY group, they are arranged by the order of the data sets in which they occur.

Interleaving

The data sets that are listed in the SET statement must be sorted by the values of the variables that are listed in the BY statement, or they must have an appropriate index.

```sas
data company;
  length name $ 15;
  set divisionA(rename=(state=location)) divisionB;
  by name;
run;
```

ERROR: BY variables are not properly sorted on data set WORK.DIVISIONB.
name=Roger location=NV FIRST.name=1 LAST.name=1 _ERROR_=1 _N_=3
NOTE: The SAS System stopped processing this step because of errors.
### Interleaving

```sas
proc sort data=divisionA;
  by name;
run;
proc sort data=divisionB;
  by name;
run;
data company;
  length name $ 15;
  set divisionA(rename=(state=location)) divisionB;
  by name;
run;
```

What is the result of this program?

A. There is an **error** because BY variables are not properly sorted.

B. There is an **error** because the variable **location** is not found in **work.divisionA**.

C. The data set **work.company** is created with seven observations and two variables.
Merging

- The MERGE statement joins observations from two or more SAS data sets into single observations.
- The BY statement specifies the common variables to match-merge observations. The variables in the BY statement must be common to all data sets.

```
data combine;
merge revenue expense;
by name;
  profit=revenue-expense;
run;
```

- The data sets listed in the MERGE statement must be sorted in the order of the values of the variables that are listed in the BY statement, or they must have an appropriate index.

One-to-One Match-Merging

```
data combine;
merge revenue expense;
by name;
  profit=revenue-expense;
run;
```

**VIEWTABLE: Work.Rev**  **Input Data Set**
```
  name  revenue
  1   Joy     15000
  2  Margaret 26000
  3   Michael 19000
  4     Tony  25000
```

**VIEWTABLE: Work.Exp**  **Input Data Set**
```
  name  expense
  1   Joy     18000
  2  Margaret 20000
  3   Michael 19000
  4     Tony  19000
```

**VIEWTABLE: Work.Combine**  **Output Data Set**
```
  name  revenue  expense  profit
  1   Joy     15000   18000    -3000
  2  Margaret 26000   20000     6000
  3   Michael 19000   19000    -1000
  4     Tony  25000   15000     6000
```
Many-to-One Match-Merging

```plaintext
data moncombine;
  merge monrevenue monexpense;
  by month;
  profit=revenue-expense;
run;
```

Output Data Set

**Input Data Set**

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Monrevenue</th>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>month</td>
</tr>
<tr>
<td>Billy</td>
<td>Apr</td>
</tr>
<tr>
<td>Janet</td>
<td>Apr</td>
</tr>
<tr>
<td>Billy</td>
<td>May</td>
</tr>
<tr>
<td>Janet</td>
<td>May</td>
</tr>
</tbody>
</table>

**Input Data Set**

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Monexpense</th>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>month</td>
<td>expense</td>
</tr>
<tr>
<td>Apr</td>
<td>5000</td>
</tr>
<tr>
<td>May</td>
<td>5500</td>
</tr>
</tbody>
</table>

**Output Data Set**

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Moncombine</th>
<th>Output Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>month</td>
</tr>
<tr>
<td>Billy</td>
<td>Apr</td>
</tr>
<tr>
<td>Janet</td>
<td>Apr</td>
</tr>
<tr>
<td>Billy</td>
<td>May</td>
</tr>
<tr>
<td>Janet</td>
<td>May</td>
</tr>
</tbody>
</table>

Nonmatches

```plaintext
data combine1;
  merge revenue1 expense1;
  by name;
  profit=revenue-expense;
run;
```

**Input Data Set**

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Revenue</th>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>revenue</td>
</tr>
<tr>
<td>Carolina</td>
<td>25000</td>
</tr>
<tr>
<td>Jack</td>
<td>18000</td>
</tr>
<tr>
<td>Jenna</td>
<td>26000</td>
</tr>
<tr>
<td>Thomas</td>
<td>15000</td>
</tr>
</tbody>
</table>

**Input Data Set**

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Expense</th>
<th>Input Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>expense</td>
</tr>
<tr>
<td>Caroline</td>
<td>19000</td>
</tr>
<tr>
<td>Jenna</td>
<td>20000</td>
</tr>
<tr>
<td>Susan</td>
<td>19000</td>
</tr>
<tr>
<td>Thomas</td>
<td>18000</td>
</tr>
</tbody>
</table>

How many observations are in the output data set work.combine1?
IN= Option

The IN= option creates a variable that indicates whether the data set contributed data to the current observation.

```sas
data combine1;
merge revenue1(in=rev) expense1(in=exp);
by name;
profit=revenue-expense;
run;
```

Within the DATA step, the value of the variable is 1 if the data set contributed to the current observation, and 0 if the data set did not contribute to the current observation.

IN= Option

Which of the following statements is **false** concerning the IN= option?

A. The IN= variables are included in the SAS data set that is being created.
B. The values of the IN= variables are available to program statements during the DATA step.
C. When a data set contributes an observation for the current BY group, the IN= value is a numeric 1.
D. The IN= data set option is specified in parentheses after a SAS data set name in the SET and MERGE statements.
### IN= Option

```sas
data combine1;
  merge revenue1(in=rev)
    expense1(in=exp);
  by name;
  profit=revenue-expense;
run;
```

#### Possible Scenarios

- **Matches only**
  - `if rev=1 and exp=1;`
  - `if rev and exp;`

- **Nonmatches from `revenue1`**
  - `if rev=1 and exp=0;`
  - `if rev and not exp;`

- **Nonmatches from `expenses1`**
  - `if rev=0 and exp=1;`
  - `if not rev and exp;`

- **All observations from `revenue1`**
  - `if rev=1;`
  - `if rev;`

- **All observations from `expenses1`**
  - `if exp=1;`
  - `if exp;`

- **Nonmatches from `revenue1` and `expenses1`**
  - `if rev=0 or exp=0;`
  - `if not rev or not exp;`
IN= Option

```sas
data combine1;
merge revenue1(in=rev) expense1(in=exp);
by name;
profit=revenue-expense;
run;
```

Which statement will give all observations from the `revenue1` data set regardless of matches or nonmatches?

A. `if rev=1;`
B. `if rev=1 and exp=1;`
C. `if rev=1 and (exp=1 and exp=0);`
D. `if (rev=1 and exp=1) and (rev=1 and exp=0);`

---

IN= Option

```sas
data revexp revonly exponly;
merge revenue1(in=rev) expense1(in=exp);
by name;
if rev=1 and exp=1 then output revexp;
else if rev=1 and exp=0 then output revonly;
else if rev=0 and exp=1 then output exponly;
run;
```

---

Refer to Exercise 5 for Chapter 2 in Appendix A.
### 2.5 Answers to Questions

<table>
<thead>
<tr>
<th>Question Slide Number</th>
<th>Answer</th>
</tr>
</thead>
</table>
| 3                     | 1. golf.supplies  
                        | 2. work.newprice |
| 8                     | B.     |
| 14                    | D.     |
| 17                    | B.     |
| 34                    | C.     |
| 38                    | B.     |
| 43                    | A.     |
| 50                    | Yes    |
| 54                    | C.     |
| 59                    | D.     |
| 66                    | C.     |
| 73                    | B.     |
| 76                    | A.     |
| 82                    | D.     |
| 86                    | D.     |
| 89                    | 1. and 3. |
| 92                    | C.     |
| 94                    | B.     |
| 98                    | B.     |
| 102                   | B.     |
| 115                   | No     |
| 118                   | - DATA step is not using the sorted data set.  
                        - The BY statement of the DATA step is not specifying the correct sort order. |

(Continued on the next page.)
<table>
<thead>
<tr>
<th>Question Slide Number</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>122</td>
<td>Six observations and three variables</td>
</tr>
<tr>
<td>124</td>
<td>Six observations and three variables</td>
</tr>
<tr>
<td>129</td>
<td>No</td>
</tr>
<tr>
<td>132</td>
<td>B.</td>
</tr>
<tr>
<td>137</td>
<td>A.</td>
</tr>
<tr>
<td>142</td>
<td>Five observations (three matches and two nonmatches)</td>
</tr>
<tr>
<td>145</td>
<td>A.</td>
</tr>
<tr>
<td>149</td>
<td>A.</td>
</tr>
</tbody>
</table>
Chapter 3  Working with Raw Data and Microsoft Excel Files

3.1  Reading Raw Data Files: Part 1.................................................................3-3
3.2  Reading Raw Data Files: Part 2...............................................................3-16
3.3  Controlling When a Record Loads..........................................................3-32
3.4  Reading Microsoft Excel Files.................................................................3-38
3.5  Answers to Questions..............................................................................3-42
3.1 Reading Raw Data Files: Part 1

Reading Raw Data Files

```sas
data work.kids;
  infile 'kids.dat';
  input name $ 1-8 siblings 10
    @12 bdate mmddyy10.
    @23 allowance comma2.
    hobby1 $ hobby2 $ hobby3 $;
run;
```

1. What is the name of the raw data file being read?

2. What is the name of the data set being created?

DATA Statement

The DATA statement begins a DATA step and provides names for any output SAS data sets that are created.

```sas
data work.kids; // Output Data Set
  infile 'kids.dat';
  input name $ 1-8 siblings 10
    @12 bdate mmddyy10.
    @23 allowance comma2.
    hobby1 $ hobby2 $ hobby3 $;
run;
```

- The DATA statement can create temporary or permanent data sets.
INFIL Statement

With an INPUT statement, the INFIL statement identifies the physical name of the external file to read.

```sas
data work.kids;
  infile 'kids.dat';
  input name $ 1-8 siblings 10
    @12 bdate mmddyy10.
    @23 allowance comma2.
  hobby1 $ hobby2 $ hobby3 $;
run;
```

- The physical name is the name that the operating environment uses to access the file.

INPUT Statement

The INPUT statement describes the arrangement of values in the input data record and assigns input values to the corresponding SAS variables.

```sas
data work.kids;
  infile 'kids.dat';
  input name $ 1-8 siblings 10
    @12 bdate mmddyy10.
    @23 allowance comma2.
  hobby1 $ hobby2 $ hobby3 $;
run;
```
INPUT Statement

Which of the following is not an input style for the INPUT statement?

A. list input
B. column input
C. delimited input
D. formatted input

INPUT Statement

The following three ways can describe a record's values in the INPUT statement:

- column input
- formatted input
- list input

```
input name $ 1-8 siblings 10
@12 bdate mmddyy10.
@23 allowance comma2.
hobby1 $ hobby2 $ hobby3 $;
```

Input Raw Data File

<table>
<thead>
<tr>
<th>Name</th>
<th>Siblings</th>
<th>Date</th>
<th>Allowance</th>
<th>Hobbies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chloe</td>
<td>2</td>
<td>11/10/95</td>
<td>$5</td>
<td>Running, Music, Gymnastics</td>
</tr>
<tr>
<td>Travis</td>
<td>2</td>
<td>1/30/98</td>
<td>$2</td>
<td>Baseball, Nintendo, Reading</td>
</tr>
<tr>
<td>Jennifer</td>
<td>0</td>
<td>8/21/99</td>
<td>$0</td>
<td>Soccer, Painting, Dancing</td>
</tr>
</tbody>
</table>
**Column Input**

With column input, the column numbers that contain the value follow a variable name in the INPUT statement.

```
input name $ 1-8 siblings 10
   @12 bdate mmddyy10.
   @23 allowance comma2.
   hobby1 $ hobby2 $ hobby3 $;
```

To read with column input, data values
- must be in the same columns in all the input data records
- must be in standard form.

**Formatted Input**

With formatted input, an informat follows a variable name and defines how SAS reads the values of this variable. An informat gives the data type and the field width of an input value.

```
input name $ 1-8 siblings 10
   @12 bdate mmddyy10.
   @23 allowance comma2.
   hobby1 $ hobby2 $ hobby3 $;
```

To read with formatted input, data values
- must be in the same columns in all the input data records
- can be in standard or nonstandard form.
**List Input**

With list input, variable names in the INPUT statement are specified in the same order that the fields appear in the input data records.

```plaintext
input name $ 1-8 siblings 10
   @12 bdate mmddyy10.
   @23 allowance comma2.
   hobby1 $ hobby2 $ hobby3 $;
```

To read with list input, data values
- must be separated with a delimiter
- can be in standard or nonstandard form.

---

**Column, Formatted, and List Input**

```plaintext
input @45 name $10;
```

Which input technique is used in the above statement?

A. column input
B. formatted input
C. list input
DATA Step Execution

When a DATA step is submitted, it is first compiled and then executed.

1. **Compile Program**
2. **Initialize Variables to Missing (PDV)**
3. **Execute “Read” Statement**
4. **Execute Other Statements**
5. **End of File?**
   - No: **Go to Next Step**
   - Yes: **Compile Program Again**

### Compilation Phase

```sas
data work.kids;
  infile 'kids.dat';
  input name $ 1-8 siblings 10
        @12 bdate mmddyy10.
        @23 allowance comma2.
        hobby1 $ hobby2 $ hobby3 $;
  run;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Siblings</th>
<th>Birth Date</th>
<th>Allowance</th>
<th>Hobby 1</th>
<th>Hobby 2</th>
<th>Hobby 3</th>
<th>Error</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chloe</td>
<td>2</td>
<td>11/10/95</td>
<td>$5</td>
<td>Running</td>
<td>Music</td>
<td>Gymnastics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Travis</td>
<td>2</td>
<td>1/30/98</td>
<td>$2</td>
<td>Baseball</td>
<td>Nintendo</td>
<td>Reading</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jennifer</td>
<td>0</td>
<td>8/21/99</td>
<td>$0</td>
<td>Soccer</td>
<td>Painting</td>
<td>Dancing</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input Buffer:

```
...```

PDV:

<table>
<thead>
<tr>
<th>Name</th>
<th>Siblings</th>
<th>Birth Date</th>
<th>Allowance</th>
<th>Hobby 1</th>
<th>Hobby 2</th>
<th>Hobby 3</th>
<th>Error</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2013, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.
**Compilation Phase**

What third item is created at compile time in addition to the input buffer and the program data vector (PDV)?

A. report  
B. data values  
C. raw data file  
D. descriptor information

**Execution Phase**

```plaintext
data work.kids;  
  infile 'kids.dat';  
  input name $ 1-8 siblings 10  
    @12 bdate mmddyy10.  
    @23 allowance comma2.  
    hobby1 $ hobby2 $ hobby3 $;  
run;
```

Input Buffer:

- Chloe 2 11/10/1995 $5 Running Music Gymnastics
- Travis 2 1/30/1998 $2 Baseball Nintendo Reading
- Jennifer 0 8/21/1999 $0 Soccer Painting Dancing

**PDV:**

<table>
<thead>
<tr>
<th>name</th>
<th>siblings</th>
<th>bdate</th>
<th>allowance</th>
<th>hobby1</th>
<th>hobby2</th>
<th>hobby3</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>$8</td>
<td>N 8</td>
<td>N 8</td>
<td>N 8</td>
<td>$8</td>
<td>$8</td>
<td>$8</td>
<td>N 8</td>
<td>N 8</td>
</tr>
</tbody>
</table>

- initializes the PDV to missing and sets the initial values of _N_ and _ERROR_.

Copyright © 2013, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.
data work.kids;
  infile 'kids.dat';
  input name $ 1-8 siblings 10
    @12 bdate mmddyy10.
    @23 allowance comma2.
    hobby1 $ hobby2 $ hobby3 $;
run;

Chloe    2 11/10/1995 $5 Running Music Gymnastics
Travis   2 1/30/1998  $2 Baseball Nintendo Reading
Jennifer 0 8/21/1999  $0 Soccer Painting Dancing

PDV:

<table>
<thead>
<tr>
<th>name</th>
<th>siblings</th>
<th>bdate</th>
<th>allowance</th>
<th>hobby1</th>
<th>hobby2</th>
<th>hobby3</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Chloe</td>
<td>2</td>
<td>13097</td>
<td>5</td>
<td>Running</td>
<td>Music</td>
<td>Gymnast</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- reads data values into the input buffer and then the PDV
- executes any subsequent programming statements

...
data work.kids;
  infile 'kids.dat';
  input name $ 1-8 siblings 10
    @12 bdate mmddyy10.
    @23 allowance comma2.
    hobby1 $ hobby2 $ hobby3 $;
run;

**Execution Phase**

Input Buffer:

<table>
<thead>
<tr>
<th>Chloe</th>
<th>2 11/10/1995 $5 Running Music Gymnastics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Travis</td>
<td>2 1/30/1998 $2 Baseball Nintendo Reading</td>
</tr>
<tr>
<td>Jennifer</td>
<td>0 8/21/1999 $0 Soccer Painting Dancing</td>
</tr>
</tbody>
</table>

**PDV:**

<table>
<thead>
<tr>
<th>name</th>
<th>siblings</th>
<th>bdate</th>
<th>allowance</th>
<th>hobby1</th>
<th>hobby2</th>
<th>hobby3</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Chloe</td>
<td>2</td>
<td>11/10/95</td>
<td>$5</td>
<td>Running</td>
<td>Music</td>
<td>Gymnastics</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- returns to the top of the DATA step

```
PDV:
```

```
resets the PDV to missing for any variables not being read directly from a data set and increments _N_ by 1

Input Buffer:

<table>
<thead>
<tr>
<th>Chloe</th>
<th>2 11/10/1995 $5 Running Music Gymnastics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Travis</td>
<td>2 1/30/1998 $2 Baseball Nintendo Reading</td>
</tr>
<tr>
<td>Jennifer</td>
<td>0 8/21/1999 $0 Soccer Painting Dancing</td>
</tr>
</tbody>
</table>

**PDV:**

<table>
<thead>
<tr>
<th>name</th>
<th>siblings</th>
<th>bdate</th>
<th>allowance</th>
<th>hobby1</th>
<th>hobby2</th>
<th>hobby3</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>11/10/95</td>
<td>$5</td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

- resets the PDV to missing for any variables not being read directly from a data set and increments _N_ by 1
### Execution Phase

```sas
data work.kids;
  infile 'kids.dat';
  input name $ 1-8 siblings 10
    @12 bdate mmddyy10.
    @23 allowance comma2.
    hobby1 $ hobby2 $ hobby3 $;
run;
```

**Input Buffer:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Siblings</th>
<th>Birthday</th>
<th>Allowance</th>
<th>Hobby1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chloe</td>
<td>2</td>
<td>11/10/1995</td>
<td>$5</td>
<td>Running</td>
</tr>
<tr>
<td>Travis</td>
<td>2</td>
<td>1/30/1998</td>
<td>$2</td>
<td>Baseball</td>
</tr>
<tr>
<td>Jennifer</td>
<td>0</td>
<td>8/21/1999</td>
<td>$0</td>
<td>Soccer</td>
</tr>
</tbody>
</table>

Repeats the process until the end of file is detected.

### Execution Phase

```
data work.kids;
  infile 'kids.dat';
  input name $ 1-8 siblings 10
    @12 bdate mmddyy10.
    @23 allowance comma2.
    hobby1 $ hobby2 $ hobby3 $;
run;
```

How many times does SAS iterate through the DATA step?

A. 0  B. 1  C. 3  D. 4
### Execution Phase

```sas
data work.kids;
  infile 'kids.dat';
  input name $ 1-8 siblings 10
      @12 bdate mmddyy10.
      @23 allowance comma2.
      hobby1 $ hobby2 $ hobby3 $;
run;
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Siblings</th>
<th>BDate</th>
<th>Allowance</th>
<th>Hobby1</th>
<th>Hobby2</th>
<th>Hobby3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chloe</td>
<td>2</td>
<td>11/10/95</td>
<td>$5</td>
<td>Running</td>
<td>Music</td>
<td>Gymnastics</td>
</tr>
<tr>
<td>Travis</td>
<td>2</td>
<td>1/30/98</td>
<td>$2</td>
<td>Baseball</td>
<td>Nintendo</td>
<td>Reading</td>
</tr>
<tr>
<td>Jennifer</td>
<td>0</td>
<td>8/21/99</td>
<td>$0</td>
<td>Soccer</td>
<td>Painting</td>
<td>Dancing</td>
</tr>
</tbody>
</table>

### Data Errors

A data error is when the INPUT statement encounters invalid data in a field.

When SAS encounters a data error, these events occur:
- A note that describes the error is printed in the SAS log.
- The input record (contents of the input buffer) being read is displayed in the SAS log.
- The values in the SAS observation (contents of the PDV) being created are displayed in the SAS log.
- A missing value is assigned to the appropriate SAS variable.
- Execution continues.
Data Errors

```sas
data work.kids1;
  infile 'kids1.dat';
  input name $ 1-8 siblings 10
     @12 bdate mmddyy10.
     @23 allowance comma2.
     hobby1 $ hobby2 $ hobby3 $
run;
```

Chloe    2 11/10/1995 $5Running Music Gymnastics
Travis   X 1/30/1998  $2Baseball Nintendo Reading
Jennifer 0 8/21/1999  $0Soccer Painting Dancing

Input Raw Data File

The siblings values are read in as numeric.

```
data work.kids1;
  infile 'kids1.dat';
  input name $ 1-8 siblings 10
     @12 bdate mmddyy10.
     @23 allowance comma2.
     hobby1 $ hobby2 $ hobby3 $
run;
```

NOTE: The infile 'kids1.dat' is:
File Name=c:\workshop\winsas\lwcrb\kids1.dat,
RECFM=V,LRECL=256

NOTE: Invalid data for siblings in line 2 10-10.
RULE: 2         Travis   X 1/30/1998  $2Baseball Nintendo Reading 49
name=Travis siblings=. bdate=13909 allowance=2 hobby1=Baseball
hobby2=Nintendo hobby3=Reading _ERROR_=1 _N_=2
NOTE: 3 records were read from the infile 'kids1.dat'.
The minimum record length was 47.
The maximum record length was 49.

NOTE: The data set WORK.KIDS1 has 3 observations and 7 variables.

```
```
```
Data Errors

What is the reason for this invalid data?

DATALINES Statement

The DATALINES statement can be used with an INPUT statement to read data directly from the program, rather than data stored in a raw data file.

data work.kids;
    input name $ 1-8 siblings 10
        @12 bdate mmddyy10.
        @23 allowance comma2.
    hobby1 $ hobby2 $ hobby3 $;
    datalines;
    Chloe  2 11/10/1995 $5Running Music Gymnastics
    Travis 2 1/30/1998 $2Baseball Nintendo Reading
    Jennifer 0 8/21/1999 $0Soccer Painting Dancing
    run;
**DATALINES Statement**

Which statement is **false** concerning the DATALINES statement?

A. Multiple DATALINES statements can be used in a DATA step.
B. A null statement (a single semicolon) is needed to indicate the end of the input data.
C. The DATALINES statement is the last statement in the DATA step and immediately precedes the first data line.

---

**Refer to Exercise 1 for Chapter 3 in Appendix A.**

### 3.2 Reading Raw Data Files: Part 2

**Column Input**

With column input, the column numbers that contain the value follow a variable name in the INPUT statement.

```plaintext
input name $ 1-8 siblings 10
   @12 bdate mmddyy10.
   @23 allowance comma2.
   hobby1 $ hobby2 $ hobby3 $;
```

To read with column input, data values
- must be in the same columns in all the input data records
- must be in standard form.
Column Input

- Standard data is any data that SAS can read without any special instructions.
- Examples of standard numeric data:

  58  -23  67.23  00.99  5.67E5  1.2E-2

A column INPUT statement can contain the following:

- **variable**: Names a variable that is assigned input values.
- **$**: Indicates to store a variable value as a character value rather than as a numeric value.
- **start-column**: Specifies the first column of the input record that contains the value to read.
- **- end-column**: Specifies the last column of the input record that contains the value to read.

```sas
data work.kids2;
  infile 'kids2.dat';
  input name $ 1-8 siblings 10 bdate $ 12-21 allowance $ 23-24 hobby1 $ 26-35 hobby2 $ 36-45 hobby3 $ 46-55;
run;
```

---

Input Raw Data File

- Chloe  2 11/10/1995 $5 Running Music Gymnastics
- Travis 2 1/30/1998  $2 Baseball Nintendo Reading
- Jennifer 0 8/21/1999 $0 Soccer Painting Dancing
Column Input

```
input name $ 1-8
siblings 10
bdate $ 12-21
allowance $ 23-24
hobby1 $ 26-35
hobby2 $ 36-45
hobby3 $ 46-55;
```

How many variables are numeric and, ideally, how many variables should be numeric?

A. 1 numeric variable and 2 variables should be numeric.
B. 1 numeric variable and 3 variables should be numeric.
C. 2 numeric variables and 2 variables should be numeric.
D. 3 numeric variables and 3 variables should be numeric.

Formatted Input

With formatted input, an informat follows a variable name and defines how SAS reads the values of this variable. An informat gives the data type and the field width of an input value.

```
input name $ 1-8 siblings 10
@12 bdate mmddyy10.
@23 allowance comma2.
hobby1 $ hobby2 $ hobby3 $;
```

To read with formatted input, data values
- must be in the same columns in all the input data records
- can be in standard or nonstandard form.
### Formatted Input

- **Nonstandard data** is any data that SAS cannot read without a special instruction.
- Examples of nonstandard numeric data:
  
  
  | 5,823 | (23) | $67.23 | 01/30/1999 | 12MAY06 |
  
  A formatted INPUT statement can contain the following:

<table>
<thead>
<tr>
<th>pointer-control</th>
<th>Moves the input pointer to a specified column in the input buffer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>@n</td>
<td>moves the pointer to column n.</td>
</tr>
<tr>
<td>+n</td>
<td>moves the pointer n columns.</td>
</tr>
</tbody>
</table>

| variable        | Names a variable that is assigned input values.                  |
| informat        | Specifies a SAS informat to use to read the variable values.    |

### Informats

An **informat** is an instruction that SAS uses to read data values into a variable.

<table>
<thead>
<tr>
<th>5823.23</th>
<th>$6,711.23</th>
<th>01/30/1999</th>
<th>BASKET</th>
</tr>
</thead>
</table>

| 5823.23 | 6711.23   | 14274      | BASKET |

SAS uses the informat to determine the following:

- whether the variable is numeric or character
- the length of character variables
Informats

Which statement is \textit{false} regarding informats?

A. When you use an informat, the informat contains a period (.) as a part of the name.
B. The $ indicates a character informat, and the absence of a $ indicates a numeric informat.
C. An informat has a default width or specifies a width, which is the number of columns to read in the input data.
D. When a problem occurs with a valid informat, SAS writes a note to the SAS log, assigns a missing value to the variable, and terminates the DATA step.
### Informats

<table>
<thead>
<tr>
<th>Raw Data Value</th>
<th>Informat</th>
<th>SAS Data Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$12,345</td>
<td>COMMA7.</td>
<td>12345</td>
</tr>
<tr>
<td>$12,345</td>
<td>DOLLAR7.</td>
<td>12345</td>
</tr>
<tr>
<td>$12.345</td>
<td>COMMAX7.</td>
<td>12345</td>
</tr>
<tr>
<td>$12.345</td>
<td>DOLLARX7.</td>
<td>12345</td>
</tr>
<tr>
<td>12.345</td>
<td>EUROX7.</td>
<td>12345</td>
</tr>
<tr>
<td>Australia</td>
<td>$11.</td>
<td>Australia</td>
</tr>
<tr>
<td>Australia</td>
<td>$CHAR11.</td>
<td>Australia</td>
</tr>
<tr>
<td>au</td>
<td>$UPCASE2.</td>
<td>AU</td>
</tr>
<tr>
<td>01/01/1960</td>
<td>MMDDYY10.</td>
<td>0</td>
</tr>
<tr>
<td>31/12/60</td>
<td>DDMMYY8.</td>
<td>365</td>
</tr>
<tr>
<td>31DEC1959</td>
<td>DATE9.</td>
<td>-1</td>
</tr>
</tbody>
</table>

### Formatted Input

```plaintext
data work.kids2;
  infile 'kids2.dat';
  input @1 name $8.
    @10 siblings 1.
    @12 bdate mmddyy10.
    @23 allowance comma2.
    @26 hobby1 $10.
    @36 hobby2 $10.
    @46 hobby3 $10.;
run;
```

Chloe    2    11/10/1995 $5    Running   Music     Gymnastics
Travis   2    1/30/1998  $2    Baseball  Nintendo  Reading
Jennifer 0    8/21/1999  $0    Soccer    Painting  Dancing
**Formatted Input**

```plaintext
input @1 name $8.
   @10 siblings 1.
   @12 bdate mmddyy10.
   @23 allowance comma2.
   @26 hobby1 $10.
   @36 hobby2 $10.
   @46 hobby3 $10.;
```

What is the byte size of `bdate` and `hobby1`?

A. `bdate` = 8 and `hobby1` = 8
B. `bdate` = 8 and `hobby1` = 10
C. `bdate` = 10 and `hobby1` = 8
D. `bdate` = 10 and `hobby1` = 10

---

**List Input**

With list input, variable names in the INPUT statement are specified in the same order that the fields appear in the input data records.

```plaintext
input name $ 1-8 siblings 10
   @12 bdate mmddyy10.
   @23 allowance comma2.
   hobby1 $ hobby2 $ hobby3 $;
```

To read with list input, data values
- must be separated with a delimiter
- can be in standard or nonstandard form.
List Input

- You must specify the variables in the order that they appear in the raw data file, left to right.
- The default length for variables is 8 bytes.
- A space (blank) is the default delimiter.

- **pointer control** Moves the input pointer to a specified column in the input buffer.
- **variable** Names a variable that is assigned input values.
- **$** Indicates to store a variable value as a character value rather than as a numeric value.
- **:** Reads data values that need the additional instructions that informats can provide but are not aligned in columns.
- **informat** Specifies an informat to use to read the variable values.

```
data work.kids3;
  length hobby1 hobby2 hobby3 $ 10;
  infile 'kids3.dat';
  input name $ siblings
  bdate : mmddyy10.
  allowance : comma2.
  hobby1 $ hobby2 $ hobby3 $;
run;
```

The LENGTH statement specifies the number of bytes for storing variables.
data work.kids3;
  length hobby1 hobby2 hobby3 $ 10;
  infile 'kids3.dat';
  input name $ siblings
    bdate : mmddyy10.
    allowance : comma2.
    hobby1 $ hobby2 $ hobby3 $;
run;

The : modifier with an informat is used to read numeric values that contain nonstandard values.
List Input

Chloe 2 11/10/1995 $5 Running Music Gymnastics
Travis 2 1/30/1998 $2 Baseball Nintendo Reading
Jennifer 0 8/21/1999 $0 Soccer Painting Dancing

The : modifier with an informat can also be used to control the byte size of character variables.

data work.kids3;
  infile 'kids3.dat';
  input name $ siblings
    bdate : mmddyy10.
    allowance : comma2.
    hobby1 : $10.
    hobby2 : $10.
    hobby3 : $10.;
run;

: Modifier with an Informat

A : modifier with an informat enables SAS to do the following:
- treat the current field as a delimited field
- apply an informat to the field, ignoring the width

input name $ salary:comma10. state $;

list input

input name $ salary comma10. state $;

formatted input
### Modifier with an Informat

```
input name $ salary:comma10. state $;
```

<table>
<thead>
<tr>
<th>VIEWTABLE: Work Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1 Ted</td>
</tr>
<tr>
<td>2 Sam</td>
</tr>
</tbody>
</table>

```
input name $ salary comma10. state $;
```

<table>
<thead>
<tr>
<th>VIEWTABLE: Work Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1 Ted</td>
</tr>
<tr>
<td>2 Sam</td>
</tr>
</tbody>
</table>

### Modifier with an Informat

```
input name $ bdate date9. state $;
```

What is the result of `state` based on the above INPUT statement?

A. Georgia  
B. orgia Florida  
C. rgia Florida
3.2 Reading Raw Data Files: Part 2

: Modifier with an Informat

---|----10---|----20
Ted 1DEC07 Georgia
Sam 21DEC2007 Florida

input name $ bdate:date9. state $;

What is the result of state based on the above INPUT statement?

A. Georgia Florida  B. orgia Florida  C. rgia Florida

DLM= Option

The DLM= option specifies a delimiter to be used for list input. Blank is the default delimiter.

data work.kids4;
  infile 'kids4.dat' dlm=',';
  input name $ siblings
    bdate : mmddyy10.
    allowance : comma2.
    hobby1 : $10.
    hobby2 : $10.
    hobby3 : $10.;
run;
Missing Data

By default, SAS treats two consecutive delimiters as one, not as a missing value between the delimiters.

data work.kids5;
  infile 'kids5.dat' dlm=',';
  input name $
    siblings
    bdate : mmddyy10.
    allowance : comma2.
    hobby1 : $10.
    hobby2 : $10.
    hobby3 : $10.;
run;

NOTE: Invalid data for siblings in line 1 8-17.
NOTE: Invalid data for bdate in line 1 20-26.
NOTE: Invalid data for allowance in line 1 28-32.
RULE:     ----+----1----+----2----+----3----+----4----+----5----
2        Travis,2,1/30/1998,$2,Baseball,Nintendo,Reading 47
NOTE: Invalid data errors for file 'kids5.dat' occurred outside the printed range.
NOTE: Increase available buffer lines with the INFILE n= option.
name=Chloe siblings=. bdate=. allowance=. hobby1=Gymnastics
hobby2=Travis hobby3=2 _ERROR_=1 _N_=1
NOTE: 3 records were read from the infile 'kids5.dat'.
The minimum record length was 43.
The maximum record length was 47.
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
DSD Option

The DSD option can do the following:
- treat two consecutive delimiters as a missing value
- remove quotation marks from strings and treat any delimiter inside the quotation marks as a valid character
- set the default delimiter to a comma

```sas
data work.kids5;
  infile 'kids5.dat' dsd;
  input name $
    siblings       
    bdate : mmddyy10.
    allowance : comma2.
    hobby1 : $10.
    hobby2 : $10.
    hobby3 : $10.;
run;
```

DSD Option

Chloe,,11/10/1995,,Running,Music,Gymnastics
Travis,2,1/30/1998,$2,Baseball,Nintendo,Reading
Jennifer,0,8/21/1999,$0,Soccer,Painting,Dancing

```
VIEWTABLE: Work.Kids5

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>siblings</th>
<th>bdate</th>
<th>allowance</th>
<th>hobby1</th>
<th>hobby2</th>
<th>hobby3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chloe</td>
<td></td>
<td>11/10/95</td>
<td></td>
<td>Running</td>
<td>Music</td>
<td>Gymnastics</td>
</tr>
<tr>
<td>2</td>
<td>Travis</td>
<td>2</td>
<td>01/30/98</td>
<td>2</td>
<td>Baseball</td>
<td>Nintendo</td>
<td>Reading</td>
</tr>
<tr>
<td>3</td>
<td>Jennifer</td>
<td>0</td>
<td>08/21/99</td>
<td>0</td>
<td>Soccer</td>
<td>Painting</td>
<td>Dancing</td>
</tr>
</tbody>
</table>
```

NOTE: 3 records were read from the infile 'kids5.dat'.
The minimum record length was 43.
The maximum record length was 47.
NOTE: The data set WORK.KIDS5 has 3 observations and 7 variables.
NOTE: DATA statement used (Total process time):
real time 0.10 seconds
cpu time 0.02 seconds
DSD Option

Chloe/2/"11/10/1995"/$5/Running/Music/Gymnastics
Travis/2/"1/30/1998"/$2/Baseball/Nintendo/Reading
Jennifer/0/"8/21/1999"//Soccer/Painting/Dancing

Which statement will correctly read the raw data file?

A. `infile 'kids5a.dat' dsd;`
B. `infile 'kids5a.dat' dlm='/';
C. `infile 'kids5a.dat' dsd dlm='/';
D. `infile 'kids5a.dat' dsd, dlm='/';

Varying Number of Fields per Record

A raw data file might have a varying number of fields per record.

Chloe 2 11/10/1995 $5 Running Music Gymnastics
Travis 2 1/30/1998 $2 Baseball Nintendo
Jennifer 0 8/21/1999 $0 Soccer

NOTE: 3 records were read from the infile 'kids6.dat'.
The minimum record length was 30.
The maximum record length was 46.
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.KIDS6 has 2 observations and 7 variables.
3.2 Reading Raw Data Files: Part 2

MISSOVER Option

The MISSOVER option prevents an INPUT statement from reading a new input data record if it does not find values in the current input line for all the variables in the statement.

```sas
data work.kids6;
  infile 'kids6.dat' missover;
  input name $ siblings
       bdate : mmddyy10.
       allowance : comma2.
       hobby1 : $10.
       hobby2 : $10.
       hobby3 : $10.;
run;
```

MISSOVER Option

When an INPUT statement reaches the end of the current input data record, variables without any values assigned are set to missing with the MISSOVER option.

Refer to Exercise 2 for Chapter 3 in Appendix A.
3.3 Controlling When a Record Loads

Multiple INPUT Statements

By default, SAS advances the pointer to column 1 of the next input record when SAS encounters an INPUT statement.

```sas
data work.kids8;
  infile 'kids8.dat';
  input name $ 1-8 siblings 10;
  input @1 bdate mmddyy10.
    @12 allowance comma2.;
  input hobby1:$10.
    hobby2:$10.
    hobby3:$10.;
run;
```

Which statement is false?

A. The DATA step iterates nine times.
B. The data set work.kids8 will have three observations.
C. Three records are read with each DATA step iteration.
D. One observation is created with each DATA step iteration.
### Multiple INPUT Statements

```plaintext
data work.kids8;
  infile 'kids8.dat';
  input name $ 1-8
      siblings 10;
  input @1 bdate mmddyy10.
      @12 allowance comma2.;
  input hobby1:$10.
  hobby2:$10.
  hobby3:$10.;
run;
```

#### Input Raw Data File

- **Chloe** 2
  - 11/10/1995 $5
  - Running, Music, Gymnastics
- **Travis** 2
  - 1/30/1998 $2
  - Baseball, Nintendo, Reading
- **Jennifer** 0
  - 8/21/1999 $0
  - Soccer, Painting, Dancing

### Line-Pointer Controls

The / line-pointer control advances the pointer to column 1 of the next input record.

```plaintext
data work.kids8;
  infile 'kids8.dat';
  input name $ 1-8
      siblings 10
      / @1 bdate mmddyy10.
      @12 allowance comma2.
      / hobby1:$10.
  hobby2:$10.
  hobby3:$10.;
run;
```
Line-Pointer Controls

The \#n line-pointer control advances the pointer to column 1 of record n.

```sas
data work.kids8;
    infile 'kids8.dat';
    input #1 name $ 1-8
        siblings 10
        #2 @1 bdate mmddyy10.
            @12 allowance comma2.
        #3 hobby1:$10.
        hobby2:$10.
        hobby3:$10.;
run;
```

Line-Pointer Controls

Yes or No: Do the following three sections produce the same results?

```sas
input name $ 1-8 siblings 10 //
    hobby1:$10. hobby2:$10. hobby3:$10.;
```

```sas
input #1 name $ 1-8 siblings 10
    #3 hobby1:$10. hobby2:$10. hobby3:$10.;
```

```sas
input name $ 1-8 siblings 10;
    input;
    input hobby1:$10. hobby2:$10. hobby3:$10.;
```
### Line-Pointer Controls

The second record is skipped in each iteration of the DATA step.

```plaintext
input name $ 1-8 siblings 10 //
  hobby1:$10. hobby2:$10. hobby3:$10.;

input #1 name $ 1-8 siblings 10
  #3 hobby1:$10. hobby2:$10. hobby3:$10.;

input name $ 1-8 siblings 10;
input;
input hobby1:$10. hobby2:$10. hobby3:$10.;
```

<table>
<thead>
<tr>
<th>VECTABLE: Work.Kids8</th>
<th>Output Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>siblings</td>
</tr>
<tr>
<td>1</td>
<td>Chloe</td>
</tr>
<tr>
<td>2</td>
<td>Travis</td>
</tr>
<tr>
<td>3</td>
<td>Jennifer</td>
</tr>
</tbody>
</table>

### Single Trailing @

The single trailing @ has the following characteristics:
- holds an input record for the execution of the next INPUT statement within the same iteration of the DATA step
- is useful when you must read from a record multiple times

This raw data file has different layouts depending on the value of state.
**Single Trailing @**

Chloe IN 11/10/1995 $5 Running Music Gymnastics
Travis IL Baseball Nintendo Reading
Jennifer IN 8/21/1999 $0 Soccer Painting Dancing

```sas
data work.kids9;
  infile 'kids9.dat';
  input name $ 1-8 state $ 10-11 @;
  if state='IN' then
    input @13 bdate mmddyy10. @24 allowance comma2.
    hobby1:$10. hobby2:$10. hobby3:$10.;
  else input @13 hobby1:$10. hobby2:$10. hobby3:$10.;
run;
```

**Input Raw Data File**

**Output Data Set**

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>State</th>
<th>Bdate</th>
<th>Allowance</th>
<th>Hobby1</th>
<th>Hobby2</th>
<th>Hobby3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chloe</td>
<td>IN</td>
<td>11/10/95</td>
<td>$5</td>
<td>Running</td>
<td>Music</td>
<td>Gymnastics</td>
</tr>
<tr>
<td>2</td>
<td>Travis</td>
<td>IL</td>
<td></td>
<td></td>
<td>Baseball</td>
<td>Nintendo</td>
<td>Reading</td>
</tr>
<tr>
<td>3</td>
<td>Jennifer</td>
<td>IN</td>
<td>8/21/99</td>
<td>$0</td>
<td>Soccer</td>
<td>Painting</td>
<td>Dancing</td>
</tr>
</tbody>
</table>

**Single Trailing @**

Which statement is **false** concerning the single trailing @?

A. In the INPUT statement, the single trailing @ must be the last item before the semicolon.
B. SAS releases a record held by a single trailing @ when an INPUT statement without a trailing @ executes.
C. The single trailing @ prevents the next INPUT statement from automatically releasing the current input record.
D. The single trailing @ holds the input record for the execution of the next INPUT statement across iterations of the DATA step.
Double Trailing @

The double trailing @ has the following characteristics:

- holds the input record for the execution of the next INPUT statement across iterations of the DATA step
- is useful when each input line contains values for several observations

```
Chloe IN Travis IL Jennifer IN
Brian IL Mark IN Kurt IN Hannah IL
```

```
data work.kids10;
  infile 'kids10.dat';
  input name $ state $ @@;
run;
```

NOTE: 2 records were read from the infile 'kids10.dat'.
The minimum record length was 30.
The maximum record length was 34.
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.KIDS10 has 7 observations and 2 variables.
Double Trailing @

Which statement is false concerning the double trailing @?

A. The double trailing @ must be the first item in the INPUT statement.
B. The double trailing @ is useful when one record contains several observations.
C. The double trailing @ holds a record for the next INPUT statement across iterations of the DATA step.
D. SAS releases the record that is held by a double trailing @ if the pointer moves past the end of the input record.

Refer to Exercise 3 for Chapter 3 in Appendix A.

3.4 Reading Microsoft Excel Files

SAS/ACCESS LIBNAME Statement

With the SAS/ACCESS Interface to PC File Formats, the LIBNAME statement can be used to access Microsoft Excel workbooks.

```
libname myxls 'customers.xls';
proc contents data=myxls._all_; run;
```

This enables you to reference a worksheet directly in a DATA step or SAS procedure, and to read from and write to an Excel worksheet as if it were a SAS data set.
SAS/ACCESS LIBNAME Statement

How do you reference the females worksheet in the Excel workbook customer based on the following LIBNAME statement?

```sas
libname myxls 'customers.xls';
```

A. `myxls.females$

B. `customer.females`

C. `myxls.'females$'n`

D. `customer.'females'n`

SAS/ACCESS LIBNAME Statement

Worksheet names appear with a dollar sign at the end of the name.

```
The CONTENTS Procedure
Directory
Libref   MYXLS
Engine   EXCEL
Physical Name customers.xls
User     Admin

#  Name  Type  Type
1  Females$  DATA  TABLE
2  Males$  DATA  TABLE
```
SAS/ACCESS LIBNAME Statement

SAS name literals enable special characters to be included in data set names.

A SAS name literal is a name token that is expressed as a string within quotation marks, followed immediately by the letter n.

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>MYXL.$'Females$n</th>
<th>Observations</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member Type</td>
<td>DATA</td>
<td>Variables</td>
<td>6</td>
</tr>
<tr>
<td>Engine</td>
<td>EXCEL</td>
<td>Indexes</td>
<td>0</td>
</tr>
<tr>
<td>Created</td>
<td>.</td>
<td>Observation Length</td>
<td>0</td>
</tr>
<tr>
<td>Last Modified</td>
<td>.</td>
<td>Deleted Observations</td>
<td>0</td>
</tr>
<tr>
<td>Protection</td>
<td></td>
<td>Compression</td>
<td>NO</td>
</tr>
<tr>
<td>Data Set Type</td>
<td></td>
<td>Sorted</td>
<td>NO</td>
</tr>
<tr>
<td>Label</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Representation</td>
<td>Default</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encoding</td>
<td>Default</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SAS/ACCESS LIBNAME Statement

If SAS has a library reference name assigned to an Excel workbook, the workbook cannot be opened in Excel.

```
libname myxls 'customers.xls';
proc contents data=myxls._all_; run;
proc print data=myxls.'females$n'; run;
```

```ncode
data work.usfemales;
  set myxls.'females$n;
  where country='US';
run;
```
3.4 Reading Microsoft Excel Files

**SAS/ACCESS LIBNAME Statement**

Which statement disassociates the `MYXLS` library reference name?

A. `libname myxls end;`
B. `libname myxls clear;`
C. `libname myxls close;`
D. `libname myxls disassociate;`

**SAS/ACCESS LIBNAME Statement**

The default Excel engine can be specified when the bit count of SAS and Microsoft Office are the same (both are 32-bit or both are 64-bit).

`libname myxls excel 'customers.xls';`

The PC Files Server engine must be specified when the bit counts differ.

`libname myxls pcfiles path='customers.xls';`

Refer to Exercise 4 for Chapter 3 in Appendix A.
### 3.5 Answers to Questions

<table>
<thead>
<tr>
<th>Question Slide Number</th>
<th>Answer</th>
</tr>
</thead>
</table>
| 4                     | 1. kids.dat  
                        | 2. work.kids |
| 9                     | C.     |
| 15                    | A.     |
| 19                    | D.     |
| 27                    | C.     |
| 33                    | The variable **hobby1** is being read in as numeric. |
| 36                    | A.     |
| 44                    | B.     |
| 49                    | A.     |
| 51                    | D.     |
| 55                    | B.     |
| 65                    | B.     |
| 67                    | A.     |
| 74                    | C.     |
| 83                    | A.     |
| 88                    | Yes    |
| 93                    | D.     |
| 97                    | A.     |
| 103                   | C.     |
| 108                   | B.     |
Chapter 4  Creating Variables

4.1  Creating Variables with the Assignment Statement ........................................... 4-3

4.2  Creating Variables Conditionally ........................................................................ 4-8

4.3  Creating Accumulator Variables ........................................................................ 4-19

4.4  Answers to Questions ........................................................................................ 4-25
4.1 Creating Variables with the Assignment Statement

**Assignment Statement**

The assignment statement evaluates an expression and stores the result in a new variable or an existing variable.

Examples:

```plaintext
name = 'Jane Doe';
revenue = 157900;
date = '10MAY2007'd;
total = price * quantity;
product = upcase(product);
average = mean(jan, feb, mar);
```

Assignment statements evaluate the expression on the right side of the equal sign and store the result in the variable that is specified on the left side of the equal sign.

**Q**

Which assignment statement is overwriting a variable used in the expression?

1. `name = 'Jane Doe';`
2. `revenue = 157900;`
3. `date = '10MAY2007'd;`
4. `total = price * quantity;`
5. `product = upcase(product);`
6. `average = mean(jan, feb, mar);`
Expression

The expression is a sequence of operands and operators that form a set of instructions that produce a value.

- **Operands** are
  - constants (character or numeric)
  - variables (character or numeric).

- **Operators** are
  - symbols that represent an arithmetic calculation or concatenation
  - a SAS function.

Operands

A constant is a number or a character string that indicates a fixed value.

```plaintext
name = 'Jane Doe';
revenue = 157900;
date = '10MAY2007';
```

Character constants must be enclosed in quotation marks.

Character constants are enclosed in quotation marks, but names of variables are not enclosed in quotation marks.

```plaintext
total = price * quantity;
cityst = city !! state;
product = upcase(product);
average = mean(jan, feb, mar);
```
Operators

- Arithmetic operators indicate that an arithmetic calculation is performed.
- A concatenation operator concatenates character values.
- A SAS function performs a computation or system manipulation on arguments and returns a value.

```plaintext
total = price * quantity;
cityst = city !! state;
product = upcase(product);
average = mean(jan, feb, mar);
```

Arithmetic Operators

Possible arithmetic operators:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>exponentiation</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
</tr>
<tr>
<td>+</td>
<td>addition</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
</tr>
</tbody>
</table>

- If a missing value is an operand for an arithmetic operator, the result is a missing value.
Example

```sas
data newprice;
  set golf.supplies;
  saleprice = price * 0.75;
  saletype = '25% off';
  format price saleprice dollar8.2;
run;
```

Which statement is true concerning the new variables?

A. `saleprice` and `saletype` are numeric (8 bytes).
B. `saleprice` and `saletype` are character (7 bytes).
C. `saleprice` is numeric (8 bytes) and `saletype` is character (7 bytes).
D. `saleprice` is numeric (8 bytes) and `saletype` is character (8 bytes).

Refer to Exercise 1 for Chapter 4 in Appendix A.
**DROP and KEEP Statements**

- The DROP statement specifies the names of the variables to omit from the output data set.
- The KEEP statement specifies the names of the variables to write to the output data set.

```sas
data newprice;
  set golf.supplies;
  drop type price;
  saleprice = price * 0.75;
  saletype = '25% off';
  format saleprice dollar8.2;
run;
```

**Partial Data Set**

<table>
<thead>
<tr>
<th>mg</th>
<th>type</th>
<th>price</th>
<th>saleprice</th>
<th>saletype</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crew</td>
<td>$40</td>
<td>$30</td>
<td>25% off</td>
</tr>
<tr>
<td>2</td>
<td>Crew</td>
<td>$49</td>
<td>$37</td>
<td>25% off</td>
</tr>
<tr>
<td>3</td>
<td>Crew</td>
<td>$70</td>
<td>$52</td>
<td>25% off</td>
</tr>
</tbody>
</table>

**Placement of statement is irrelevant; statement is applied at output time.**

**DROP= and KEEP= Options**

Which program is equivalent to the previous program?

A. ```sas
   data newprice(drop=type price);
   set golf.supplies;
   saleprice = price * 0.75;
   saletype = '25% off';
   format saleprice dollar8.2;
run;
```

B. ```sas
   data newprice;
   set golf.supplies(drop=type price);
   saleprice = price * 0.75;
   saletype = '25% off';
   format saleprice dollar8.2;
run;
```
**DROP= and KEEP= Options**

The DROP= data set option excludes the variables for
- processing if in the SET statement
- writing to the output data set if in the DATA statement.

The KEEP= data set option specifies the variables for
- processing if in the SET statement
- writing to the output data set if in the DATA statement.

```sas
data newprice(drop=price);
  set golf.supplies(keep=mfg price);
  saleprice = price * 0.75;
  saletype = '25% off';
  format saleprice dollar8.2;
run;
```

**4.2 Creating Variables Conditionally**

**IF-THEN / ELSE Statements**

- The IF-THEN statement executes *a statement* for observations that meet specific conditions.
- The optional ELSE statement gives an alternative action if the THEN clause is not executed.

```sas
data newprice;
  set golf.supplies;
  if mfg='Crew' then saleprice=price*0.75;
  else if mfg='Hi-fly' then saleprice=price*0.70;
  else if mfg='White' then saleprice=price*0.90;
  format price saleprice dollar8.2;
run;
```
Expression

The expression is any valid SAS expression and is a required argument.
SAS evaluates the expression in an IF-THEN statement to produce a result that is either nonzero, zero, or missing.
A nonzero and nonmissing result causes the expression to be true; a result of zero or missing causes the expression to be false.

Expression

Examples:

- city in ('New York','Boston')
- upcase(last)='SMITH'
- city='Atlanta' and salary<50000
- salary ge 125000
- birth='12DEC1999'd
- revenue ne goal

Which expression is invalid in the IF-THEN statement?

A. if total then ...
B. if 12000<=revenue<=25000 then ...
C. if city='Reno' state='NV' then ...
D. if salary>75000 or bonus<5000 then ...
Statement

```sas
if mfg='Crew' then saleprice=price*0.75;
```

The statement can be any executable SAS statement.

Examples:

- `status = 'Unknown'
- `count + 1
- `total = sum(num1, num2, num3)
- `delete
- `anniversary = '15AUG2006'd
- `output

ELSE Statements

```sas
if mfg='Crew' then saleprice=price*0.75;
else if mfg='Hi-fly' then saleprice=price*0.70;
else if mfg='White' then saleprice=price*0.90;
```

Yes or No: Is the word ELSE required each time in the above example?
ELSE Statements

if mfg='Crew' then saleprice=price*0.75;
else if mfg='Hi-fly' then saleprice=price*0.70;
else if mfg='White' then saleprice=price*0.90;

Using IF-THEN statements without the ELSE statement causes SAS to evaluate all IF-THEN statements.

Using IF-THEN statements with the ELSE statement causes SAS to execute the IF-THEN statements until SAS encounters the first true statement. Subsequent IF-THEN statements are not evaluated.

Based on the partial program above, what is the result if an observation has a value of mfg='X-treme'?

A. The observation will be deleted.
B. The observation will have a missing value for saleprice.
C. The observation will have a saleprice value equal to price multiplied by 0.75.
D. The observation will have a saleprice value equal to price multiplied by 0.90.
ELSE Statements

```plaintext
if mfg='Crew' then saleprice=price*0.75;
else if mfg='Hi-fly' then saleprice=price*0.70;
else saleprice=price*0.90;
```

The final ELSE statement can be coded without an IF-THEN statement to direct all previous false conditions into the final condition.

IF-THEN DO / ELSE DO Statements

The IF-THEN DO statement executes a group of statements for observations that meet specific conditions.

```plaintext
... if mfg='Crew' then do;
   pct=0.75;
   saleprice = price * pct;
   saletype = '25% off';
end;
else if mfg='Hi-fly' then do;
   pct=0.70;
   saleprice = price * pct;
   saletype = '30% off';
end;
else do;
   pct=0.90;
   saleprice = price * pct;
   saletype = '10% Storewide Sale';
end;
...  
```

Refer to Exercise 2 for Chapter 4 in Appendix A.
IF-THEN DO / ELSE DO Statements

```sas
data newprice;
  infile 'raw-data-file';
  input mfg $ type $ price;
  length saletype $ 18;
  if mfg='Crew' then do;
    pct=0.75;
    saleprice = price * pct;
    saletype = '25% off';
  end;
  else if mfg='Hi-fly' then do;
    pct=0.70;
    saleprice = price * pct;
    saletype = '30% off';
  end;
  else do;
    pct=0.90;
    saleprice = price * pct;
    saletype = '10% Storewide Sale';
  end;
  format price saleprice dollar8.2;
run;
```

Conditional statements can be used in DATA steps that read raw data files or data sets.

Three variables are being created. pct and saleprice are numeric (8 bytes). saletype is character (18 bytes).
data newprice;
infile 'raw-data-file';
input mfg $ type $ price;
length saletype $ 18;
if mfg='Crew' then do;
pct=0.75;
saleprice = price * pct;
saletype = '25% off';
end;
else if mfg='Hi-fly' then do;
pct=0.70;
saleprice = price * pct;
saletype = '30% off';
end;
else do;
pct=0.90;
saleprice = price * pct;
saletype = '10% Storewide Sale';
end;
format price saleprice dollar8.2;
run;

All previous false conditions fall into the final condition.

Which syntax is valid for IF-THEN DO statements?

A. if payclass='monthly' then do amt=salary;
else if payclass='hourly' then do amt=hrlywage*40;

B. if 12000<=revenue<=24000 then do; x='goal' end;
else if revenue<12000 then do; x='below' end;
else if revenue>24000 then do; x='above' end;

C. if salary>=100000 then do;
category='Exec'; range='Above 100K'; end;
else if salary<100000 then do;
category='Non-Exec'; range='Below 100K'; end;

D. if mon in ('JUN', 'JUL', 'AUG') then do;
status='SUMMER' end;
else if mon in ('MAR', 'APR', 'MAY') then do;
status='SPRING' end;
else do;
status='FALL OR WINTER';
SELECT / WHEN / OTHERWISE Statements

An alternative to IF-THEN statements is SELECT / WHEN / OTHERWISE statements.

- The SELECT statement begins a SELECT group.
- SELECT groups contain WHEN statements that identify SAS statements that are executed when a particular condition is true.
- A SELECT group must use at least one WHEN statement.
- An optional OTHERWISE statement specifies a statement to be executed if no WHEN condition is met.
- An END statement ends a SELECT group.

```sas
data newprice;
set golf.supplies;
select(mfg);
when('Crew') saleprice=price*0.75;
when('Hi-fly') saleprice=price*0.70;
when('White') saleprice=price*0.90;
end;
format price saleprice dollar8.2;
run;
```

```sas
data newprice;
set golf.supplies;
if mfg='Crew' then
  saleprice=price*0.75;
else if mfg='Hi-fly' then
  saleprice=price*0.70;
else if mfg='White' then
  saleprice=price*0.90;
format price saleprice dollar8.2;
run;
```
**SELECT / WHEN / OTHERWISE Statements**

```
select(mfg);
  when('Crew') saleprice=price*0.75;
  when('Hi-fly') saleprice=price*0.70;
  when('White') saleprice=price*0.90;
end;
```

Based on the partial program above, what is the result if an observation has a value of `mfg='X-treme'`?

**SELECT / WHEN / OTHERWISE Statements**

A null OTHERWISE statement prevents SAS from issuing an error message when all WHEN conditions are false.

```
select(mfg);
  when('Crew') saleprice=price*0.75;
  when('Hi-fly') saleprice=price*0.70;
  when('White') saleprice=price*0.90;
otherwise;
end;
```

```
select(mfg);
  when('Crew') saleprice=price*0.75;
  when('Hi-fly') saleprice=price*0.70;
  when('White') saleprice=price*0.90;
end;
```

ERROR: Unsatisfied WHEN clause and no OTHERWISE clause at line 618 column 3.
mfg=X-treme type=Strata price=$10.60
saleprice=. _ERROR_=1 _N_=10

```
select(mfg);
  when('Crew') saleprice=price*0.75;
  when('Hi-fly') saleprice=price*0.70;
  when('White') saleprice=price*0.90;
end;
```

no ERROR
4.2 Creating Variables Conditionally

... if mfg='Crew' then do;
  pct=0.75;
  saleprice = price * pct;
  saletype = '25% off';
end;
else if mfg='Hi-fly' then do;
  pct=0.70;
  saleprice = price * pct;
  saletype = '30% off';
end;
else do;
  pct=0.90;
  saleprice = price * pct;
  saletype = '10% Storewide Sale';
end;
...

select(mfg);
when('Crew') do;
  pct=0.75;
  saleprice = price * pct;
  saletype = '25% off';
end;
when('Hi-fly') do;
  pct=0.70;
  saleprice = price * pct;
  saletype = '30% off';
end;
otherwise do;
  pct=0.90;
  saleprice = price * pct;
  saletype='10% Storewide Sale';
end;
end;

SELECT group

data newprice;
  infile 'raw-data-file';
  input mfg $ type $ price;
  length saletype $ 18;
  select(mfg);
    when('Crew') do;
      pct=0.75;
      saleprice = price * pct;
      saletype = '25% off';
    end;
    when('Hi-fly') do;
      pct=0.70;
      saleprice = price * pct;
      saletype = '30% off';
    end;
    otherwise do;
      pct=0.90;
      saleprice = price * pct;
      saletype='10% Storewide Sale';
    end;
  end;
  format price saleprice dollar8.2;
run;

All previous false conditions fall into the final condition.
SELECT / WHEN / OTHERWISE Statements

Additional examples:

```sas
select;
   when('JUN', 'JUL', 'AUG') and temp>70) status='SUMMER';
when('MAR', 'APR', 'MAY') status='SPRING';
otherwise status='FALL OR WINTER';
end;
```

```sas
select;
   when('monthly') amt=salary;
   when('hourly') amt=hrlywage*40;
otherwise;
end;
```

```sas
select;
   when(12000<=revenue<=24000) target='goal';
   when(revenue<12000) target='below';
   when(revenue>24000) target='above';
otherwise;
end;
```

```sas
data new;
   set rev;
   length target $ 5;
select;
   when (12000<=revenue<=24000) target='goal';
   when (revenue<12000) target='below';
   when (revenue>24000) target='above';
otherwise;
end;
run;
```

What is the value of target when revenue=24000?
A. missing
B. goal
C. below
D. above
4.3 Creating Accumulator Variables

**Accumulator Variables**

An accumulator variable is a variable that adds on an expression.

Partial Output

<table>
<thead>
<tr>
<th>County</th>
<th>Population 2000</th>
<th>Total Population</th>
<th>Total Counties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Androscoggin</td>
<td>103,793</td>
<td>103,793</td>
<td>1</td>
</tr>
<tr>
<td>Aroostook</td>
<td>73,938</td>
<td>177,731</td>
<td>2</td>
</tr>
<tr>
<td>Cumberland</td>
<td>265,612</td>
<td>443,343</td>
<td>3</td>
</tr>
<tr>
<td>Franklin</td>
<td>29,467</td>
<td>472,810</td>
<td>4</td>
</tr>
<tr>
<td>Hancock</td>
<td>51,791</td>
<td>524,601</td>
<td>5</td>
</tr>
<tr>
<td>Kennebec</td>
<td>117,114</td>
<td>641,715</td>
<td>6</td>
</tr>
<tr>
<td>Knox</td>
<td>39,618</td>
<td>681,333</td>
<td>7</td>
</tr>
<tr>
<td>Lincoln</td>
<td>33,616</td>
<td>714,949</td>
<td>8</td>
</tr>
<tr>
<td>Oxford</td>
<td>54,755</td>
<td>769,704</td>
<td>9</td>
</tr>
<tr>
<td>Penobscot</td>
<td>144,919</td>
<td>914,623</td>
<td>10</td>
</tr>
</tbody>
</table>

**Sum Statement**

The sum statement adds the result of an expression to an accumulator variable.

Examples:

- `TotCounties + 1;`
- `TotPopulation + Population2000;`
**Sum Statement**

The accumulator variable has the following characteristics:
- must be a numeric variable
- is automatically set to 0 before SAS reads the first observation
- is retained from one iteration to the next

The expression is defined with the following features:
- is any SAS expression
- is evaluated and the result added to the accumulator variable
- is ignored if missing

---

**Sum Statement**

The sum statement is equivalent to using the RETAIN statement and the SUM function.

```sas
retain TotPopulation 0;
TotPopulation = sum(TotPopulation, Population2000);
```

- The RETAIN statement causes a variable to retain its value from one iteration of the DATA step to the next and specifies an initial value for the variable.
- The SUM function returns the sum of the nonmissing arguments.
Sum and RETAIN Statements

Yes or No: Is the RETAIN statement ever needed with the sum statement?

RETAIN Statement

To initialize an accumulator variable to a value other than zero, include the accumulator variable in a RETAIN statement with an initial value.

data FiveYearPop;
  set FiveYearPop;
  retain year 1945;
  year+5;
  run;

<table>
<thead>
<tr>
<th>Population</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>914950</td>
</tr>
<tr>
<td>2</td>
<td>940841</td>
</tr>
<tr>
<td>3</td>
<td>976669</td>
</tr>
<tr>
<td>4</td>
<td>993235</td>
</tr>
<tr>
<td>5</td>
<td>997357</td>
</tr>
<tr>
<td>6</td>
<td>1062640</td>
</tr>
<tr>
<td>7</td>
<td>1125027</td>
</tr>
<tr>
<td>8</td>
<td>1183950</td>
</tr>
<tr>
<td>9</td>
<td>1227326</td>
</tr>
</tbody>
</table>
Accumulator Variable for BY Groups

In order to create an accumulator variable for BY groups, the beginning and end of each BY group must be determined.

<table>
<thead>
<tr>
<th>County</th>
<th>Town</th>
<th>Population</th>
<th>Total Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>Androscoggin</td>
<td>Auburn</td>
<td>23203</td>
<td>23203</td>
</tr>
<tr>
<td>Androscoggin</td>
<td>Lewiston</td>
<td>35890</td>
<td>58893</td>
</tr>
<tr>
<td>Cumberland</td>
<td>Brunswick</td>
<td>21172</td>
<td>21172</td>
</tr>
<tr>
<td>Cumberland</td>
<td>Portland</td>
<td>64249</td>
<td>85421</td>
</tr>
<tr>
<td>Cumberland</td>
<td>Scarborough</td>
<td>16970</td>
<td>102391</td>
</tr>
<tr>
<td>Cumberland</td>
<td>South Portland</td>
<td>23324</td>
<td>126715</td>
</tr>
<tr>
<td>Kennebec</td>
<td>Augusta</td>
<td>18560</td>
<td>18560</td>
</tr>
<tr>
<td>Penobscot</td>
<td>Bangor</td>
<td>31473</td>
<td>31473</td>
</tr>
<tr>
<td>York</td>
<td>Biddeford</td>
<td>20942</td>
<td>20942</td>
</tr>
<tr>
<td>York</td>
<td>Sanford</td>
<td>20806</td>
<td>41748</td>
</tr>
</tbody>
</table>

BY-Group Processing

In the DATA step, SAS identifies the beginning and end of each BY group by creating two temporary variables for each BY variable: the FIRST. and LAST. variables.

```
data TopCounties;
  set Top10Town;
  by County;
rung;
```

These temporary variables are available for DATA step programming but are not added to the output data set.
BY-Group Processing

- The **FIRST.** variable is set to 1 when an observation is the first in a BY group. Otherwise, it equals 0.
- The **LAST.** variable is set to 1 when an observation is the last in a BY group. Otherwise, it equals 0.

<table>
<thead>
<tr>
<th>County</th>
<th>Population</th>
<th>first.County</th>
<th>last.County</th>
</tr>
</thead>
<tbody>
<tr>
<td>Androscoggin</td>
<td>23203</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Androscoggin</td>
<td>35690</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Cumberland</td>
<td>21172</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Cumberland</td>
<td>64249</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cumberland</td>
<td>16970</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cumberland</td>
<td>23324</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Kennebec</td>
<td>18560</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Which of the following is **false**?

A. **First.County**=1 and **Last.County**=0 means the observation is the first one in the BY group.
B. **First.County**=0 and **Last.County**=1 means the observation is the last one in the BY group.
C. **First.County**=0 and **Last.County**=0 means the observation is the first and the last one in the BY group.
D. **First.County**=0 and **Last.County**=0 means the observation is neither the first nor the last one in the BY group.
BY-Group Processing

The following program resets the accumulator variable at the beginning of each BY group and outputs only at the end of each BY group.

```
data TopCounties;
  set Top10Town;
  by County;
  if first.County then TotalPopulation=0;
  TotalPopulation+Population;
  if last.County=1;
  keep County TotalPopulation;
run;
```

```
<table>
<thead>
<tr>
<th>County</th>
<th>TotalPopulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Androscoggin</td>
</tr>
<tr>
<td>2</td>
<td>Cumberland</td>
</tr>
<tr>
<td>3</td>
<td>Kennebec</td>
</tr>
<tr>
<td>4</td>
<td>Penobscot</td>
</tr>
<tr>
<td>5</td>
<td>York</td>
</tr>
</tbody>
</table>
```

Multiple BY-Group Variables

A FIRST. variable and a LAST. variable is created for each variable in the BY statement.

```
data CityDonate;
  set Donations;
  by State City;
run;
```

Data must be sorted or indexed by State and City.

First.State, Last.State, First.City, and Last.City are created.

When you use more than one variable in the BY statement, a change in the primary variable forces the LAST. variable=1 for the secondary variable.

Refer to Exercise 3 for Chapter 4 in Appendix A.
Multiple BY-Group Variables

```sas
data CityDonate;
  set Donations;
  by State City;
  if first.City=1 then TotalDonation=0;
  TotalDonation+Donation;
  if last.City=1;
run;
```

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>donation</th>
<th>TotalDonation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC</td>
<td>Charlotte</td>
<td>4000</td>
<td>15000</td>
</tr>
<tr>
<td>NC</td>
<td>Greenville</td>
<td>3000</td>
<td>9000</td>
</tr>
<tr>
<td>SC</td>
<td>Greenville</td>
<td>2000</td>
<td>7000</td>
</tr>
<tr>
<td>SC</td>
<td>Pelzer</td>
<td>5000</td>
<td>5000</td>
</tr>
</tbody>
</table>

### 4.4 Answers to Questions

<table>
<thead>
<tr>
<th>Question Slide Number</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6.</td>
</tr>
<tr>
<td>11</td>
<td>C.</td>
</tr>
<tr>
<td>17</td>
<td>A.</td>
</tr>
<tr>
<td>24</td>
<td>C.</td>
</tr>
<tr>
<td>27</td>
<td>No</td>
</tr>
<tr>
<td>30</td>
<td>B.</td>
</tr>
<tr>
<td>39</td>
<td>C.</td>
</tr>
<tr>
<td>43</td>
<td>ERROR: Unsatisfied WHEN clause and no OTHERWISE clause.</td>
</tr>
<tr>
<td>49</td>
<td>B.</td>
</tr>
<tr>
<td>56</td>
<td>Yes</td>
</tr>
<tr>
<td>62</td>
<td>C.</td>
</tr>
</tbody>
</table>
# Chapter 5  Manipulating Data

5.1 Using Functions to Manipulate Data ................................................................. 5-3

5.2 Converting Character and Numeric Data ....................................................... 5-34

5.3 Processing Data with DO Loops ..................................................................... 5-41

5.4 Processing Data with Arrays ........................................................................ 5-51

5.5 Answers to Questions .................................................................................... 5-59
5.1 Using Functions to Manipulate Data

Functions

A SAS function performs a computation or system manipulation on arguments and returns a value.

SAS functions can be used in DATA step programming statements, in a WHERE expression, and in the REPORT procedure.

<table>
<thead>
<tr>
<th>Functions by Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character</strong></td>
</tr>
<tr>
<td><strong>Date and Time</strong></td>
</tr>
<tr>
<td><strong>Truncation</strong></td>
</tr>
<tr>
<td><strong>Descriptive</strong></td>
</tr>
<tr>
<td><strong>Statistics</strong></td>
</tr>
<tr>
<td><strong>Special</strong></td>
</tr>
</tbody>
</table>

Character Functions

<table>
<thead>
<tr>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBSTR</td>
</tr>
<tr>
<td>CAT</td>
</tr>
<tr>
<td>LOWCASE</td>
</tr>
<tr>
<td>SCAN</td>
</tr>
<tr>
<td>CATS</td>
</tr>
<tr>
<td>UPCASE</td>
</tr>
<tr>
<td>LEFT</td>
</tr>
<tr>
<td>CATT</td>
</tr>
<tr>
<td>PROPCASE</td>
</tr>
<tr>
<td>RIGHT</td>
</tr>
<tr>
<td>CATX</td>
</tr>
<tr>
<td>FIND</td>
</tr>
<tr>
<td>TRIM</td>
</tr>
<tr>
<td>TRANWRD</td>
</tr>
<tr>
<td>COMPRESS</td>
</tr>
<tr>
<td>STRIP</td>
</tr>
<tr>
<td>LENGTH</td>
</tr>
<tr>
<td>COMPBL</td>
</tr>
</tbody>
</table>

Refer to Exercise 1 for Chapter 5 in Appendix A.
SUBSTR Function

The SUBSTR function used to the right of the equal sign extracts a substring from an argument.

\[
\text{Features} = \text{substr(VIN, 4, 5)};
\]

The SUBSTR function can be used to the left of the equal sign to replace character value constants.

SUBSTR Function

If the SUBSTR function returns a value to a variable that was not yet assigned a length, by default, the variable length is determined by the length of the first argument.

<table>
<thead>
<tr>
<th>VIN (17 bytes)</th>
<th>Assignment Statement Using SUBSTR Function</th>
<th>New Variable (17 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1F1JF27W86J178227</td>
<td>Make = substr(VIN, 2, 1);</td>
<td>F</td>
</tr>
<tr>
<td>Features = substr(VIN, 4, 5);</td>
<td>JF27W</td>
<td></td>
</tr>
<tr>
<td>SequenceNumber = substr(VIN, 12);</td>
<td>178227</td>
<td></td>
</tr>
</tbody>
</table>

If you omit the length, SAS extracts the remainder of the expression.
**SUBSTR Function**

The tenth position of a vehicle identification number represents the model year.

Which assignment statement correctly extracts only the tenth position of the vehicle identification number?

A. `ModelYear = substr(VIN,1);`

B. `ModelYear = substr(VIN,10);`

C. `ModelYear = substr(VIN,1,10);`

D. `ModelYear = substr(VIN,10,1);`

---

**LENGTH Function**

The `LENGTH` function returns the length of a non-blank character string, excluding trailing blanks, and returns 1 for a blank character string.

`Level=strsubr(Title, length(Title), 1);`

In this example, the `Level` variable is equal to the last character of the `Title` variable.

<table>
<thead>
<tr>
<th>Name</th>
<th>Title</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>Trainer 1</td>
<td>1</td>
</tr>
<tr>
<td>Pat</td>
<td>Consultant 3</td>
<td>3</td>
</tr>
<tr>
<td>Tim</td>
<td>Analyst 2</td>
<td>2</td>
</tr>
<tr>
<td>Lou</td>
<td>Consultant 2</td>
<td>2</td>
</tr>
</tbody>
</table>
SCAN Function

The \textit{SCAN function} selects a given word from a character expression.

\begin{align*}
\text{Six} & = \text{scan}(\text{Sentence} \ , \ 6 \ , \ ' ' ) ; \\
\end{align*}

If the SCAN function returns a value to a variable that was not yet assigned a length, by default, the variable is assigned a length of 200.

\begin{tabular}{|c|c|}
\hline
\textbf{Sentence (34 bytes)} & \\
\hline
This is a test, a very short test. & \\
\hline
\hline
\textbf{Assignment Statement Using the SCAN Function} & \textbf{New Variable (200 bytes)} \\
\hline
Six = \text{scan}(\text{Sentence},6,' '); & \text{very} \\
\hline
Two = \text{scan}(\text{Sentence},2,'','); & \text{a very short test.} \\
\hline
\end{tabular}

leading blank
SCAN Function

What must be added to the following program to control the byte size of the variables Six and Two?

```sas
data test;
  Sentence= 'This is a test, a very short test.';
  Six=scan(Sentence,6,' ');
  Two=scan(Sentence,2,',');
run;
```

SCAN Function

If you omit the delimiter, a default list of delimiters is used.

ASCII environment: blank . < ( + & ! $ * ) ; ^ - / , % | 
EBCDIC environment: blank . < ( + & ! $ * ) ¬ - / , % | ¢

<table>
<thead>
<tr>
<th>Sentence (36 bytes)</th>
<th>This+is an(ex-tremely)<strong>crazy</strong>test!</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Assignment Statement Using the SCAN Function</th>
<th>New Variable (200 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two = scan(Sentence,2);</td>
<td>is</td>
</tr>
<tr>
<td>Four = scan(Sentence,4);</td>
<td>ex</td>
</tr>
</tbody>
</table>
SCAN Function

- Leading delimiters before the first word in the character string do not affect the SCAN function.
- If there are two or more contiguous delimiters, the SCAN function treats them as one.
- If \( n \) is greater than the number of words in the character string, the SCAN function returns a blank value.
- If \( n \) is negative, the SCAN function selects the word in the character string starting from the end of the string.

Q

What are the new variable values?

<table>
<thead>
<tr>
<th>Assignment Statement Using the SCAN Function</th>
<th>New Variable (200 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>One = scan(Sentence,1);</td>
<td></td>
</tr>
<tr>
<td>Six = scan(Sentence,6);</td>
<td></td>
</tr>
<tr>
<td>Eight = scan(Sentence,8);</td>
<td></td>
</tr>
<tr>
<td>MinusTwo = scan(Sentence,-2);</td>
<td></td>
</tr>
<tr>
<td>Two = scan(Sentence,2,'*');</td>
<td></td>
</tr>
</tbody>
</table>
LEFT and RIGHT Functions

- The **LEFT function** left-aligns a character expression.

  LEFT returns an argument with leading blanks moved to the end of the value.

- The **RIGHT function** right-aligns a character expression.

  RIGHT returns an argument with trailing blanks moved to the start of the value.

If the LEFT or RIGHT function returns a value to a variable that was not yet assigned a length, the variable length is determined by the length of the argument.

<table>
<thead>
<tr>
<th>Var (13 bytes)</th>
<th>Assignment Statement Using the LEFT or RIGHT Function</th>
<th>New Variable (13 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 leading blanks</td>
<td>NewVar1 = left(Var); ZOOLOGY</td>
<td>6 trailing blanks</td>
</tr>
<tr>
<td>ZOOLOGY</td>
<td>NewVar2 = right(Var); 6 leading blanks</td>
<td>3 trailing blanks</td>
</tr>
</tbody>
</table>
LEFT and RIGHT Functions

What is the value of the new variable?

<table>
<thead>
<tr>
<th>Var (13 bytes)</th>
<th>Assignment Statement Using the LEFT or RIGHT Function</th>
<th>New Variable (13 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 leading blanks</td>
<td>NewVar3 = substr(left(Var),1,3);</td>
<td></td>
</tr>
<tr>
<td>ZOOLOGY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 trailing blanks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Concatenation Operator

The concatenation operator (|| - two vertical bars, \|| - two broken vertical bars, or !! - two exclamation points) concatenates character values.

```plaintext
FullName = First || Middle || Last;
```

- The length of the resulting variable is the sum of the lengths of each variable or constant in the concatenation operation, unless you use a LENGTH statement to specify a different length for the new variable.
- The concatenation operator does not trim leading or trailing blanks. If variables are padded with trailing blanks, use the TRIM function to trim trailing blanks from values before concatenating them.
Concatenation Operator

```
data name;
  length Name $ 20 First Middle Last $ 10;
  Name = 'Jones, Mary Ann, Sue';
  First = left(scan(Name,2,',',));
  Middle = left(scan(Name,3,',',));
  Last = scan(name,1,',',);
  FullName = First || Middle || Last;
run;
```

<table>
<thead>
<tr>
<th>First</th>
<th>Middle</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary Ann</td>
<td>Sue</td>
<td>Jones</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FullName (30 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary Ann Sue Jones</td>
</tr>
</tbody>
</table>

2 blanks 7 blanks 5 blanks

TRIM Function

The TRIM function removes trailing blanks from character expressions and returns one blank if the expression is missing.

```
FullName = trim(First) || trim(Middle) || Last;
```

- The TRIM function is useful for concatenating because concatenation does not remove trailing blanks.
- If the TRIM function returns a value to a variable that was not yet assigned a length, by default, the variable length is determined by the length of the argument.
TRIM Function

data name;
  length Name $ 20 First Middle Last $ 10;
  Name = 'Jones, Mary Ann, Sue';
  First = left(scan(Name,2,'',''));
  Middle = left(scan(Name,3,'',''));
  Last = scan(name,1,'','');
  FullName = trim(First)||trim(Middle)||Last;
run;

<table>
<thead>
<tr>
<th>First</th>
<th>Middle</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10 bytes)</td>
<td>(10 bytes)</td>
<td>(10 bytes)</td>
</tr>
<tr>
<td>Mary Ann</td>
<td>Sue</td>
<td>Jones</td>
</tr>
<tr>
<td>14 blanks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FullName</th>
</tr>
</thead>
<tbody>
<tr>
<td>(30 bytes)</td>
</tr>
<tr>
<td>Mary AnnSueJones</td>
</tr>
</tbody>
</table>

TRIM Function

The following program is submitted:

data name;
  length Name $ 20 First Middle Last $ 10;
  Name = 'Jones, Mary Ann, Sue';
  First = left(scan(Name,2,'',''));
  Middle = left(scan(Name,3,'',''));
  Last = scan(name,1,'','');
  Name1=trim(Middle);
  Name2=trim(First)||' '||trim(Middle)||' '||Last;
run;

What is the byte size of Name1 and Name2?
A. Name1=3 and Name2=30
B. Name1=3 and Name2=32
C. Name1=10 and Name2=30
D. Name1=10 and Name2=32
STRIP Function

The STRIP function removes leading and trailing blanks from character expressions.

```sas
data name;
  length Name $ 20 Last First Middle $ 10;
  Name = 'Jones, Mary Ann, Sue';
  Last = scan(name,1,',');
  First = scan(Name,2,',');
  Middle = scan(Name,3,',');
  Name2=strip(First)||' '||strip(Middle)||' '||Last;
run;
```

This example does not use the LEFT function when creating the First and Middle variables.

CAT Functions

The following functions concatenate character strings:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT</td>
<td>Does not remove leading or trailing blanks.</td>
</tr>
<tr>
<td>CATS</td>
<td>Removes leading and trailing blanks.</td>
</tr>
<tr>
<td>CATT</td>
<td>Removes trailing blanks.</td>
</tr>
<tr>
<td>CATX</td>
<td>Removes leading and trailing blanks and inserts separators.</td>
</tr>
</tbody>
</table>

```sas
FullName1 = cat(First, Middle, Last);
FullName2 = cats(First, Middle, Last);
FullName3 = catt(First, Middle, Last);
FullName4 = catx(' ', First, Middle, Last);
```
data name;
    length First Middle Last $ 10;
    First = 'Tony';
    Middle = ' Albert';
    Last = 'Smith';
    Name1 = cat(First, Middle, Last);
    Name2 = cats(First, Middle, Last);
    Name3 = catt(First, Middle, Last);
    Name4 = catx(' ', First, Middle, Last);
run;

First (10 bytes) Middle (10 bytes) Last (10 bytes)
Tony Albert Smith

New Variables (200 bytes)
Name1 Tony Albert Smith CAT does not remove blanks.
Name2 TonyAlbertSmith CATS removes leading and trailing blanks.
Name3 Tony AlbertSmith CATT removes trailing blanks.
Name4 Tony Albert Smith CATX is CATS plus adds separators.

---

**CAT Functions**

The following program is submitted:

data location;
    length City State $ 15 CityState $ 30;
    City= 'Princeton';
    State= 'New Jersey';
    <insert statement here>
run;

Which statement will give the CityState variable a value of Princeton, New Jersey?

A. CityState = cats(City, State);
B. CityState = cats(', ', City, State);
C. CityState = catx(', ', City, State);
D. CityState = catx(' ', City, State);
5.1 Using Functions to Manipulate Data

**TRANWRD Function**

The *TRANWRD function* replaces or removes all occurrences of a word in a character string.

The target specifies the string searched for in the source.

```
NewSentence1=tranwrd(Sentence,'hard','easy');
```

The replacement specifies the string that replaces the target.

If the TRANWRD function returns a value to a variable that was not yet assigned a length, by default, the variable is assigned a length of 200.

Sentence (31 bytes)

Functions are very hard to use.

<table>
<thead>
<tr>
<th>Assignment Statement Using the TRANWRD Function</th>
<th>New Variable (200 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewSentence1 = tranwrd(Sentence,'hard','easy');</td>
<td>Functions are very easy to use.</td>
</tr>
<tr>
<td>NewSentence2 = tranwrd(Sentence,'hard','difficult');</td>
<td>Functions are very difficult to use.</td>
</tr>
</tbody>
</table>
TRANWRD Function

The following program is submitted:

```sas
data function;
    Sentence='Functions are very hard to use.';
    NewSentence3=tranwrd(Sentence,'HARD','FUN');
run;
```

What is the result of `NewSentence3`?

A. Functions are very hard to use.
B. Functions are very HARD to use.
C. Functions are very fun to use.
D. Functions are very FUN to use.

FIND Function

The `FIND` function searches a character expression for a string of characters.

```sas
num = find(string,substring);
```

- The FIND function searches the string, from left to right, for the first occurrence of the substring, and returns the position in the string of the substring’s first character.
- If the substring is not found in the string, the FIND function returns a value of 0.
- If there are multiple occurrences of the substring, the FIND function returns only the position of the first occurrence.
FIND Function

```plaintext
num = find(string, substring);
```

<table>
<thead>
<tr>
<th>string</th>
<th>substring</th>
<th>num</th>
</tr>
</thead>
<tbody>
<tr>
<td>character 37 bytes</td>
<td>character 4 bytes</td>
<td>numeric 8 bytes</td>
</tr>
<tr>
<td>How much WOOD would a woodchuck chuck</td>
<td>WOOD</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>wood</td>
<td>23</td>
</tr>
</tbody>
</table>

The following program is submitted:

```plaintext
data tonguetwister;
  string='How much WOOD would a woodchuck chuck';
  num=find(string,'wood ');
run;
```

What is the value of `num`?
A. 0  
B. 10  
C. 14  
D. 23
FIND Function

The following program is submitted:

```sas
data tonguetwister;
  string='How much WOOD would a woodchuck chuck';
  num=find(string,' wo');
run;
```

What is the value of `num`?
A. 0  
B. 10  
C. 14  
D. 23

FIND Function

The FIND function can have modifiers and specify a starting position.

```sas
num5 = find(string,'WOOD ','it',15);
```

modifiers

starting position
FIND Function

A modifier is a character constant, variable, or expression that specifies one or more modifiers. The following modifiers can be in uppercase or lowercase:
- i - ignores character case during the search.
- t - trims trailing blanks from string and substring.

FIND Function

The starting position is an integer that specifies the position at which the search should start and the direction of the search.
- Greater than 0 - starts the search at the starting position, and the direction of the search is to the right. If the starting position is greater than the length of the string, the FIND function returns a value of 0.
- Less than 0 - starts the search at the starting position, and the direction of the search is to the left. If the starting position is greater than the length of the string, the search starts at the end of the string.
FIND Function

num5 = find(string,'WOOD ','it',15);

string
character
37 bytes

num5
numeric
8 bytes

How much WOOD would a woodchuck chuck

Ignore case and trim trailing blanks.

Start at position 15 and search to the right.

Case Functions

The LOWCASE function converts all letters in an argument to lowercase.

name1 = lowcase(name);

The UPCASE function converts all letters in an argument to uppercase.

name2 = upcase(name);

Refer to Exercise 2 for Chapter 5 in Appendix A.
Case Functions

The *PROPCASE function* converts all words in an argument to proper case.

\[ \text{name3} = \text{propcase(name)}; \]

- The PROPCASE function first converts all letters to lowercase letters and then converts the first character of words to uppercase.
- The first character of a word is the first letter of a string or any letter preceded by a default list of delimiters.
  Default Delimiter List:  blank  /  - (  .  tab
- Delimiters can be specified as a second argument, instead of using the default list.

\[ \text{name4} = \text{propcase(name,' ')}; \]

---

Case Functions

If the case functions return a value to a variable that was not yet assigned a length, by default, the variable length is determined by the length of the first argument.

<table>
<thead>
<tr>
<th>Name (16 bytes)</th>
<th>Assignment Statement Using Case Functions</th>
<th>New Variable (16 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane SMITH-JONES</td>
<td>name1 = lowcase(name); Jane Smith-Jones</td>
<td></td>
</tr>
<tr>
<td></td>
<td>name2 = upcase(name); JANE SMITH-JONES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>name3 = propcase(name); Jane Smith-Jones</td>
<td></td>
</tr>
<tr>
<td></td>
<td>name4 = propcase(name,' '); Jane Smith-jones</td>
<td></td>
</tr>
</tbody>
</table>
Case Functions

The following program is submitted:

```sas
data example;
  Var='r&d, u.s. division (not puerto rico)';
  NewVar=propcase(Var);
run;
```

The following is the desired value of **NewVar**:

**R&D, U.S. Division (Not Puerto Rico)**

Yes or No: Does the program create the desired value of **NewVar**?

---

COMPBL and COMPRESS Functions

- The *COMPBL function* removes multiple blanks in a character string by translating each occurrence of two or more consecutive blanks into a single blank.
- The *COMPRESS function* returns a character string with specified characters removed from the string.

<table>
<thead>
<tr>
<th>var (13 bytes)</th>
<th>Assignment Statement</th>
<th>New Variable (13 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC - DEF GH</td>
<td>Var1=compbl(Var);</td>
<td>ABC - DEF GH</td>
</tr>
<tr>
<td></td>
<td>Var2=compress(Var);</td>
<td>ABC-DEFGH</td>
</tr>
<tr>
<td></td>
<td>Var3=compress(Var,'-');</td>
<td>ABC  DEF GH</td>
</tr>
<tr>
<td></td>
<td>Var4=compress(Var,'- ');</td>
<td>ABCDEFGH</td>
</tr>
</tbody>
</table>
Compressing Characters

<table>
<thead>
<tr>
<th>Var (13 bytes)</th>
<th>Assignment Statement</th>
<th>Var5 (13 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC DEF GH</td>
<td>Var5=expression;</td>
<td>ABC DEF GH</td>
</tr>
</tbody>
</table>

Which expression creates the desired Var5 variable?

A. compbl(compress(Var,'-'))

B. compress(compbl(Var),'-')

Date Functions

<table>
<thead>
<tr>
<th>Date</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WEEKDAY</td>
<td>YEAR</td>
</tr>
<tr>
<td>DAY</td>
<td>TODAY or DATE</td>
</tr>
<tr>
<td>MONTH</td>
<td>MDY</td>
</tr>
<tr>
<td>QTR</td>
<td>YRDIF</td>
</tr>
</tbody>
</table>
### Date Functions

The SAS date value for Wednesday, June 13, 1962, is 894.

```sas
data birthday;
  BirthDate=894;
  BirthWeekDay=weekday(BirthDate);
  BirthDay=day(BirthDate);
  BirthMonth=month(BirthDate);
  BirthQtr=qtr(BirthDate);
  BirthYear=year(BirthDate);
output;
run;
```

What is the value of `BirthQtr`?

<table>
<thead>
<tr>
<th>Birth Date</th>
<th>Birth WeekDay</th>
<th>Birth Day</th>
<th>Birth Month</th>
<th>Birth Qtr</th>
<th>Birth Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>894</td>
<td>4</td>
<td>13</td>
<td>6</td>
<td>1</td>
<td>1962</td>
</tr>
</tbody>
</table>

#### Date Functions

- The `WEEKDAY function` returns the day of the week (1=Sunday, … 7=Saturday) from a SAS date value.
- The `DAY function` returns the day of the month (1-31) from a SAS date value.
- The `MONTH function` returns the month (1-12) from a SAS date value.
- The `QTR function` returns the quarter of the year (1-4) from a SAS date value.
- The `YEAR function` returns the four-digit year from a SAS date value.
**Date Functions**

The following program is submitted:

```sas
data birthday;
  BirthDate='13JUN1962'd;
  BirthWeekDay=weekday(BirthDate);
  output;
run;
```

Which of the following statements is true?

A. **BirthDate** and **BirthWeekDay** are both character.
B. **BirthDate** and **BirthWeekDay** are both numeric.
C. **BirthDate** is character and **BirthWeekDay** is numeric.
D. **BirthDate** is numeric and **BirthWeekDay** is character.

---

**TODAY and DATE Functions**

The **TODAY** and **DATE** functions returns the current date as a SAS date value.

- A SAS date value is the number of days since January 1, 1960.

If today were December 15, 2007:

<table>
<thead>
<tr>
<th>Current (Numeric 8 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Current = today();</code></td>
</tr>
<tr>
<td><code>Current = date();</code></td>
</tr>
<tr>
<td>17515</td>
</tr>
</tbody>
</table>
MDY Function

The MDY function returns a SAS date value from month (integer 1 through 12), day (integer 1 through 31), and year (two-digit integer or four-digit integer) values.

<table>
<thead>
<tr>
<th>Date (Numeric 8 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date = mdy(12, 15, 2007);</td>
</tr>
<tr>
<td>Date = mdy(m, d, y);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>m</th>
<th>d</th>
<th>y</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>27</td>
<td>1982</td>
<td>8121</td>
</tr>
</tbody>
</table>

YRDIF Function

The YRDIF function returns the difference in years between two dates.

```sas
age = yrdif(birth, '02feb2007'd, 'AGE');
```

If the value of basis is AGE, then YRDIF computes the age. The age computation takes into account leap years.
YRDIF Function

Given the following assignment statement:

\[
\text{age=yrdif('05MAY1999'd,'10NOV1999'd,'AGE');}
\]

What is an approximate value of \text{age}?
Truncation Functions

<table>
<thead>
<tr>
<th>Truncation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEIL</td>
</tr>
<tr>
<td>FLOOR</td>
</tr>
<tr>
<td>INT</td>
</tr>
<tr>
<td>ROUND</td>
</tr>
</tbody>
</table>

Refer to Exercise 3 for Chapter 5 in Appendix A.

CEIL and FLOOR Functions

The **CEIL function** returns the smallest integer that is *greater* than or equal to the argument.
The **FLOOR function** returns the largest integer that is *less* than or equal to the argument.

<table>
<thead>
<tr>
<th>num</th>
<th>ceil(num)</th>
<th>floor(num)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.75</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>-2.75</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>23.1234</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>-23.1234</td>
<td>-23</td>
<td>-24</td>
</tr>
</tbody>
</table>

If the argument is within 1E-12 of an integer, the function returns that integer.
5.1 Using Functions to Manipulate Data

### CEIL and FLOOR Functions

The following program is submitted:

``` Sas
data trunc;
  num=29;  A=ceil(num);  B=floor(num);
  output;
run;
```

What is the value of A and B in the final data set?

A. A=29 and B=29  
B. A=30 and B=28  
C. A=30 and B=29  
D. A=29 and B=28

### INT Function

The *INT function* returns the integer value.

<table>
<thead>
<tr>
<th>num</th>
<th>int(num)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.75</td>
<td>2</td>
</tr>
<tr>
<td>-2.75</td>
<td>-2</td>
</tr>
<tr>
<td>23.1234</td>
<td>23</td>
</tr>
<tr>
<td>-23.1234</td>
<td>-23</td>
</tr>
</tbody>
</table>

If the argument is within 1E-12 of an integer, the INT function returns that integer.
**INT Function**

Yes or No: Are both of the following statements true?

- The INT function has the same result as the FLOOR function if the value of the argument is positive.
- The INT function has the same result as the CEIL function if the value of the argument is negative.

<table>
<thead>
<tr>
<th>num</th>
<th>ceil(num)</th>
<th>floor(num)</th>
<th>int(num)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.75</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>-2.75</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
</tr>
<tr>
<td>23.1234</td>
<td>24</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>-23.1234</td>
<td>-23</td>
<td>-24</td>
<td>-23</td>
</tr>
</tbody>
</table>

**ROUND Function**

The *ROUND function* rounds the first argument to the nearest integer when the second argument is omitted.

<table>
<thead>
<tr>
<th>num</th>
<th>round(num)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.75</td>
<td>3</td>
</tr>
<tr>
<td>-2.75</td>
<td>-3</td>
</tr>
<tr>
<td>23.1234</td>
<td>23</td>
</tr>
<tr>
<td>-23.1234</td>
<td>-23</td>
</tr>
</tbody>
</table>
### ROUND Function

The ROUND function rounds the first argument to the nearest multiple of the second argument.

```sas
data rounding;
  d1 = round(1234.56789, 100);
  d2 = round(1234.56789, 10);
  d3 = round(1234.56789, 1);
  d4 = round(1234.56789, .1);
  d5 = round(1234.56789, .01);
  d6 = round(1234.56789, .001);
run;
```

<table>
<thead>
<tr>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>d4</th>
<th>d5</th>
<th>d6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>1230</td>
<td>1235</td>
<td>1234.6</td>
<td>1234.57</td>
<td>1234.568</td>
</tr>
</tbody>
</table>

### ROUND Function

The following program is submitted:

```sas
data round;
  num=986.151;
  C=round(num,100);  D=round(num,.01);
output;
run;
```

What is the value of C and D in the final data set?

A. C=900 and D=986.2
B. C=1000 and D=986.2
C. C=900 and D=986.15
D. C=1000 and D=986.15
Descriptive Statistics Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAX</strong></td>
<td>Returns the largest value.</td>
</tr>
<tr>
<td><strong>MEAN</strong></td>
<td>Returns the arithmetic mean (average).</td>
</tr>
<tr>
<td><strong>MIN</strong></td>
<td>Returns the smallest value.</td>
</tr>
<tr>
<td><strong>SUM</strong></td>
<td>Returns the sum of the nonmissing arguments.</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td>Returns the number of nonmissing numeric values.</td>
</tr>
<tr>
<td><strong>NMISS</strong></td>
<td>Returns the number of missing numeric values.</td>
</tr>
<tr>
<td><strong>CMISS</strong></td>
<td>Returns the number of missing numeric and character values.</td>
</tr>
</tbody>
</table>

```sas
data math;
var1=2;
var2=6;
var3=.;
var4=4;
maximum = max(var1,var2,var3,var4);
average = mean(var1,var2,var3,var4);
minimum = min(var1,var2,var3,var4);
total = sum(var1,var2,var3,var4);
run;
```

<table>
<thead>
<tr>
<th>maximum</th>
<th>average</th>
<th>minimum</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
<td>13</td>
</tr>
</tbody>
</table>
5.1 Using Functions to Manipulate Data

**Descriptive Statistics Functions**

The argument list can consist of a variable list, which is preceded by OF.

```sas
data math;
  var1=2;
  var2=6;
  var3=.;
  var4=4;
  maximum = max(of var1-var4);
  average = mean(of var1-var4);
  minimum = min(of var1-var4);
  total = sum(of var1-var4);
run;
```

<table>
<thead>
<tr>
<th>maximum</th>
<th>average</th>
<th>minimum</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4</td>
<td>2</td>
<td>13</td>
</tr>
</tbody>
</table>

**SAS Variable Lists**

<table>
<thead>
<tr>
<th>Numbered range lists</th>
<th>x1-xn</th>
<th>Specifies all variables from x1 to xn inclusive. You can begin with any number and end with any number as long as you do not violate the rules for user-supplied variable names and the numbers are consecutive.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name range lists</td>
<td>x--a</td>
<td>Specifies all variables ordered as they are ordered in the program data vector, from x to a inclusive.</td>
</tr>
<tr>
<td></td>
<td>x-numeric-a</td>
<td>Specifies all numeric variables from x to a inclusive.</td>
</tr>
<tr>
<td></td>
<td>x-character-a</td>
<td>Specifies all character variables from x to a inclusive.</td>
</tr>
<tr>
<td>Name prefix lists</td>
<td>REV:</td>
<td>Specifies all the variables that begin with REV, such as REVJAN, REVFEB, and REVMAR.</td>
</tr>
<tr>
<td>Special SAS name lists</td>
<td><em>ALL</em></td>
<td>Specifies all variables that are already defined in the current DATA step.</td>
</tr>
<tr>
<td></td>
<td><em>NUMERIC</em></td>
<td>Specifies all numeric variables that are already defined in the current DATA step.</td>
</tr>
<tr>
<td></td>
<td><em>CHARACTER</em></td>
<td>Specifies all character variables that are already defined in the current DATA step.</td>
</tr>
</tbody>
</table>
Descriptive Statistics Functions

The following program is submitted:

```sas
data math;
  var1=30;
  var2=15;
  var3=10;
  total = sum(var1-var3);
run;
```

What is the value of `total`?
A. 0  
B. 20  
C. 40  
D. 55

5.2 Converting Character and Numeric Data

Converting Character and Numeric Data

Data can be converted with the following two methods:
- automatic conversion
- explicit conversion with a SAS function

<table>
<thead>
<tr>
<th>Special</th>
<th>INPUT</th>
<th>PUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>character-to-numeric</td>
<td>numeric-to-character</td>
<td></td>
</tr>
</tbody>
</table>
Automatic Character-to-Numeric Conversion

Automatic character-to-numeric conversion happens when a character value is used in a numeric context. For example:

- assignment to a numeric variable
  ```
  num=char;
  ```
- an arithmetic operation
  ```
  num2=num1+char;
  ```
- logical comparison with a numeric value
  ```
  if num>char;
  ```
- a function that takes numeric arguments
  ```
  num2=mean(num1,char);
  ```

Automatic Character-to-Numeric Conversion

Automatic character-to-numeric conversion uses the W. informat
- produces a numeric missing if the character value does not conform to the W. informat
- writes a message to the SAS log stating that the conversion occurred.

```sas
819  data numeric;
820    num1=5;
821    char='6';
822    num2=num1+char;
823  run;
```

NOTE: Character values have been converted to numeric values at the places given by: (Line):(Column).

```
822:13
```

NOTE: The data set WORK.NUMERIC has 1 observations and 3 variables.
Automatic Numeric-to-Character Conversion

Automatic numeric-to-character conversion happens when a numeric value is used in a character context. For example:

- assignment to a character variable
  ```
  char=num;
  ```
- a concatenation operation
  ```
  char2=char1||num;
  ```
- a function that takes character arguments
  ```
  char=substr(num,3,1);
  ```

Automatic Numeric-to-Character Conversion

Automatic numeric-to-character conversion uses the BEST12. format, right-aligns the resulting character value, and writes a message to the SAS log stating that the conversion occurred.

```plaintext
828  data character;
829    num=1234567;
830    char=substr(num,3,1);
831  run;

NOTE: Numeric values have been converted to character values at the places given by: (Line):(Column).
830:15
NOTE: The data set WORK.CHARACTER has 1 observations and 2 variables.
```

Refer to Exercise 4 for Chapter 5 in Appendix A.
### Explicit Conversion Using SAS Functions

Explicit conversion using a SAS function
- produces desirable results
- does not write a message to the SAS log stating that the conversion occurred.

<table>
<thead>
<tr>
<th>Special</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>PUT</td>
</tr>
<tr>
<td>character-to-numeric</td>
<td>numeric-to-character</td>
</tr>
</tbody>
</table>

#### Explicit Conversion Using SAS Functions

The INPUT function converts a character value to a numeric value.
- The second argument is a numeric informat.
- If the INPUT function returns a value to a variable that was not yet assigned a length, by default, the variable length is 8 bytes.

The PUT function converts a numeric value to a character value.
- The second argument is a numeric format.
- If the PUT function returns a value to a variable that was not yet assigned a length, by default, the variable length is determined by the width of the format.
### INPUT Function

<table>
<thead>
<tr>
<th>Value of Character Variable</th>
<th>Value of Numeric Variable (8 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>162400 input('162400',6.)</td>
<td>162400</td>
</tr>
<tr>
<td>$162,400 input('$162,400',comma8.)</td>
<td>162400</td>
</tr>
<tr>
<td>49275.937 input('49275.937',9.)</td>
<td>49275.937</td>
</tr>
<tr>
<td>+24 input('+24',3.)</td>
<td>24</td>
</tr>
<tr>
<td>-73.5 input('-73.5',5.)</td>
<td>-73.5</td>
</tr>
<tr>
<td>01234 input('01234',5.)</td>
<td>1234</td>
</tr>
<tr>
<td>52E3 input('52E3',4.)</td>
<td>52000</td>
</tr>
<tr>
<td>01/01/1960 input('01/01/1960',mmddyy10.)</td>
<td>0</td>
</tr>
</tbody>
</table>

### PUT Function

<table>
<thead>
<tr>
<th>Value of Numeric Variable (8 bytes)</th>
<th>Value of Character Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>162400 put(162400,dollar8.);</td>
<td>$162,400</td>
</tr>
<tr>
<td>49275.937 put(49275.937,comma10.3);</td>
<td>49,275.937</td>
</tr>
<tr>
<td>-73.5 put(-73.5,5.1);</td>
<td>-73.5</td>
</tr>
<tr>
<td>52E3 put(52E3,5.);</td>
<td>52000</td>
</tr>
<tr>
<td>0 put(0,date9.);</td>
<td>01JAN1960</td>
</tr>
</tbody>
</table>
5.2 Converting Character and Numeric Data

**Data Set Personnel:**

<table>
<thead>
<tr>
<th>Hired</th>
<th>First</th>
<th>Last</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>27MAR2003</td>
<td>Samantha</td>
<td>Jones</td>
<td>44444444</td>
</tr>
<tr>
<td>01SEP2006</td>
<td>Timothy</td>
<td>Peters</td>
<td>99999999</td>
</tr>
</tbody>
</table>

**Hired** is character.  
**SSN** is numeric.

The following program is submitted:

```sas
data NewPersonnel;
  set Personnel;
  NewHired = input(Hired, date9.);
  TempSSN = put(SSN, 9.);
  NewSSN = catx('-', substr(TempSSN, 1, 3), substr(TempSSN, 4, 2), substr(TempSSN, 6));
run;
```

**Data Set NewPersonnel:**

<table>
<thead>
<tr>
<th>Hired</th>
<th>First</th>
<th>Last</th>
<th>SSN</th>
<th>NewHired</th>
<th>TempSSN</th>
<th>NewSSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>27MAR2003</td>
<td>Samantha</td>
<td>Jones</td>
<td>44444444</td>
<td>15731</td>
<td>44444444</td>
<td>444-444444</td>
</tr>
<tr>
<td>01SEP2006</td>
<td>Timothy</td>
<td>Peters</td>
<td>99999999</td>
<td>17065</td>
<td>99999999</td>
<td>099-999999</td>
</tr>
</tbody>
</table>

The **SSNw.** format could be used to display the numeric **SSN** with dashes.

**INPUT and PUT Functions**

**Data Set Personnel:**

<table>
<thead>
<tr>
<th>Hired</th>
<th>First</th>
<th>Last</th>
<th>SSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>27MAR2003</td>
<td>Samantha</td>
<td>Jones</td>
<td>44444444</td>
</tr>
<tr>
<td>01SEP2006</td>
<td>Timothy</td>
<td>Peters</td>
<td>99999999</td>
</tr>
</tbody>
</table>

**Hired** is character.  
**SSN** is numeric.

The following program is submitted:

```sas
data NewPersonnel;
  set Personnel;
  Hired = input(Hired, date9.);
  SSN = put(SSN, 9.);
run;
```

Yes or No: Does this modified program create a numeric **Hired** and a character **SSN**?
INPUT and PUT Functions

After a variable type is established, it cannot be changed.

Hired=input(Hired,date9.);

cannot be numeric because already defined as character

defined in input data set as character

INPUT and PUT Functions

Solution:

data NewPersonnel;
set Personnel
(rename=(Hired=TempH SSN=TempS));
Hired=input(TempH,date9.);
SSN=put(TempS,9.);
drop TempH TempS;
run;

- The RENAME= data set option changes the names of the original variables with unwanted data types.
- The DROP statement excludes the original variables with unwanted data types from the output SAS data set.
5.3 Processing Data with DO Loops

DO Loops

The DO loop executes statements between DO and END repetitively based on the value of an index variable.

```
data training;
  miles=30;
  do weeks = 1 to 6 by 1;
    miles+miles*0.04;
  end;
run;
```

- Sally ran 30 miles per week.
- She plans to increase her mileage by 4% each week for six weeks.

Index Variable

The index variable names a variable whose value governs execution of the DO group.

```
data training;
  miles=30;
  index variable
  do weeks = 1 to 6 by 1;
    miles+miles*0.04;
  end;
run;
```

- The index variable argument is required.
- Unless you specify to drop it, the index variable is included in the data set that is being created.
The specification denotes an expression or a series of expressions.

```plaintext
do weeks = 1 to 6 by 1;
   miles+miles*0.04;
end;
```

Possible specifications:
- `start TO stop BY increment`
- `start1, start2, ...
- `WHILE(expression)`
- `UNTIL(expression)`

- `start` specifies the initial value of the index variable.
- `stop` is an optional value that specifies the ending value of the index variable.
- `increment` is an optional value that specifies a positive or negative number to control the incrementing of the index variable.
  If no increment is specified, the index variable is increased by 1.
### Specification

Which of the following is **not** a valid DO statement?

A. `do 2 to 10 by 2;`

B. `do year=2006 to 2000 by -2;`

C. `do count=1 to num2+num3 by num4;`

D. `do date='15JUN2007'd to '31DEC2007'd;`

### DO Loop Output

The following program is submitted:

``` SAS
data training;
miles=30;
do weeks = 1 to 6;
miles+miles*0.04;
end;
run;
proc print data=training noobs;
run;
```

What is the result?

A. 

<table>
<thead>
<tr>
<th>miles</th>
<th>weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>31.2000</td>
<td>1</td>
</tr>
<tr>
<td>32.4480</td>
<td>2</td>
</tr>
<tr>
<td>33.7459</td>
<td>3</td>
</tr>
<tr>
<td>35.0958</td>
<td>4</td>
</tr>
<tr>
<td>36.4996</td>
<td>5</td>
</tr>
<tr>
<td>37.9596</td>
<td>6</td>
</tr>
</tbody>
</table>

B. 

<table>
<thead>
<tr>
<th>miles</th>
<th>weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>37.9596</td>
<td>6</td>
</tr>
</tbody>
</table>

C. 

<table>
<thead>
<tr>
<th>miles</th>
<th>weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>37.9596</td>
<td>7</td>
</tr>
</tbody>
</table>
DO Loop Output

The value of increment is evaluated before the execution of the loop.

data training;
miles=30;
do weeks = 1 to 6;
   miles+miles*0.04;
end;
output;
run;

Define start, stop, and increment values. Set INDEX=start.

Is INDEX out of range?

YES

NO

Execute statements in loop

INDEX=INDEX+increment
DO Loop with SET Statement

```sas
data training;
set runners;
do weeks=1 to 6;
miles+miles*pct;
output;
end;
run;
```

1. How many times does SAS loop through the DATA step?

2. How many times does SAS loop through the DO loop per each DATA step iteration?

3. How many observations are created?

Three observations are in the data set `work.runners`.

```sas
%viewtable('work.runners', columns=(name miles pct));
```

<table>
<thead>
<tr>
<th>name</th>
<th>miles</th>
<th>pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jill</td>
<td>24</td>
<td>0.05</td>
</tr>
<tr>
<td>Ray</td>
<td>28</td>
<td>0.04</td>
</tr>
<tr>
<td>Mark</td>
<td>30</td>
<td>0.05</td>
</tr>
</tbody>
</table>
```
DO Loop with INFILE/INPUT Statements

Yes or No: Can a DO loop be in a DATA step that is reading in a raw data file?

```sas
data training;
  infile 'raw-data-file';
  input name $ miles pct;
  do weeks=1 to 6;
    miles+miles*pct;
    output;
  end;
run;
```

Nested DO Loops

DO loops can be nested.

<table>
<thead>
<tr>
<th>miles</th>
<th>pct</th>
<th>month</th>
<th>weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>30.9000</td>
<td>0.03</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>31.8270</td>
<td>0.03</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>32.7818</td>
<td>0.03</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>33.7853</td>
<td>0.03</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>35.1159</td>
<td>0.04</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>36.5205</td>
<td>0.04</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>37.9813</td>
<td>0.04</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>39.5006</td>
<td>0.04</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>41.4755</td>
<td>0.05</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>43.5494</td>
<td>0.05</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>45.7269</td>
<td>0.05</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>48.0192</td>
<td>0.05</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- Sally runs 30 miles per week.
- She plans to increase her mileage weekly for three months.

3% weekly increase for month 1
4% weekly increase for month 2
5% weekly increase for month 3
### Nested DO Loops

The following program is submitted with no OUTPUT statement:

```sas
data training;
  miles=30;
pct=0.02;
do month=1 to 3;
pct+0.01;
do weeks = 1 to 4;
miles+miles*pct;
output;
end;
end;
run;
```

How many observations are created?

A. 0  B. 1  C. 3  D. 12

12 observations
Additional Specifications

The specification in the DO statement can be a series of items separated by commas.

- The items can be either all numeric or all character constants, or might be variables.
- Character constants must be enclosed in quotation marks.
- The DO group is executed once for each value in the list.

Examples:

- do month = 'JAN', 'FEB', 'MAR';
- do count = 2, 3, 5, 7, 11, 13, 17;
- do i = var1, var2, var3;
- do date = '01JAN2007'd, '25APR2007'd;

Sally runs 30 miles per week.

Thirty days from now she plans to increase her mileage by 10%.

One hundred days from now she plans to increase her mileage again by 10%.

```
data training;
miles=30;
do date=today()+30, today()+100;
miles+miles*0.10;
output;
end;
r
run;
```

<table>
<thead>
<tr>
<th>miles</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>33.0</td>
<td>27MAY2007</td>
</tr>
<tr>
<td>36.3</td>
<td>05AUG2007</td>
</tr>
</tbody>
</table>
5.3 Processing Data with DO Loops

Conditional Specifications

The DO statement can execute statements repetitively while a condition is true or until a condition is true.

**DO WHILE(expression)**

- Executes while a condition is true
- Is evaluated at the **top** of the loop
- Does not execute if the expression is false the first time that it is evaluated.

**DO UNTIL(expression)**

- Executes until a condition is true
- Is evaluated at the **bottom** of the loop
- Is executed at least once.

---

Sally plans to increase her mileage by 4% each week until she runs 50+ miles per week.

```
data training;
miles=30;
do while(miles<=50);	week+1;	miles+miles*0.04;	output;	end;
run;
```

```
data training;
miles=30;
do until(miles>50);	week+1;	miles+miles*0.04;	output;	end;
run;
```

<table>
<thead>
<tr>
<th>week</th>
<th>miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>31.2</td>
</tr>
<tr>
<td>2</td>
<td>32.45</td>
</tr>
<tr>
<td>3</td>
<td>33.75</td>
</tr>
<tr>
<td>4</td>
<td>35.1</td>
</tr>
<tr>
<td>5</td>
<td>36.5</td>
</tr>
<tr>
<td>6</td>
<td>37.96</td>
</tr>
<tr>
<td>7</td>
<td>38.48</td>
</tr>
<tr>
<td>8</td>
<td>41.00</td>
</tr>
<tr>
<td>9</td>
<td>42.7</td>
</tr>
<tr>
<td>10</td>
<td>44.41</td>
</tr>
<tr>
<td>11</td>
<td>46.18</td>
</tr>
<tr>
<td>12</td>
<td>48.03</td>
</tr>
<tr>
<td>13</td>
<td>49.95</td>
</tr>
<tr>
<td>14</td>
<td>51.98</td>
</tr>
</tbody>
</table>
```
Conditional Specifications

Yes or No: Will the DO loop in the following program execute?

```sas
data training;
miles=45;
do until(miles>40);
  week+1;
miles+miles*0.04;
output;
end;
run;
```

Combined Specifications

A WHILE(expression) or UNTIL(expression) specification can be combined with the start-to-stop-by-increment specification.

```sas
data training;
miles=30;
do weeks=1 to 30 until(miles>50);
miles+miles*0.02;
output;
end;
run;
```

Sally plans to increase her mileage by 2% each week until she runs 50+ miles per week or until she reaches 30 weeks.

In a DO UNTIL loop, the condition is checked before the index variable is incremented.

In a DO WHILE loop, the condition is checked after the index variable is incremented.
Combined Specifications

The following program is submitted:

```sas
data training;
miles=30;
do weeks=1 to 30 until(miles>50);
miles+miles*0.02;
output;
end;
run;
```

The `training` data set has 26 observations with the last observation resembling the following:

<table>
<thead>
<tr>
<th>miles</th>
<th>weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>50.202543431</td>
<td>26</td>
</tr>
</tbody>
</table>

What ended the DO loop?

A. `weeks=1 to 30`  
B. `until(miles>50)`

---

5.4 Processing Data with Arrays

**Arrays**

An array is a temporary grouping of SAS variables that are arranged in a particular order and identified by an array name.

- Arrays exist only for the duration of the current DATA step.
- Arrays are referenced by the array name and a subscript.
- The array name is not a variable.

An array is only a convenient way of temporarily identifying a group of variables. Arrays are often referenced in DO loops because more than one element in an array must be processed.
**Arrays**

Examples:

<table>
<thead>
<tr>
<th>Arrays</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>run</td>
<td>week1-week4</td>
</tr>
<tr>
<td>health</td>
<td>Height Weight BlPres Pulse Chol</td>
</tr>
</tbody>
</table>

**Defining an Array**

An ARRAY statement defines elements of an array.

```sas
array run{4} week1-week4;
array health{5} Height Weight BlPres Pulse Chol;
```

- The number of elements must be enclosed in parentheses (), braces {}, or brackets [].
- Variables defined in a given array must be all character or all numeric.
Referencing an Array

To reference an array that was previously defined in the same DATA step, use an array reference.

```plaintext
run{1} run{2} run{3} run{4}
week1 week2 week3 week4
health{1} health{2} health{3} health{4} health{5}
Height Weight BlPres Pulse Chol
```

Defining and Referencing an Array

```plaintext
data Increase;
  set Weekly;
  array run{4} week1 - week4;
  do week = 1 to 4;
    run{week} = run{week}*1.10;
  end;
  drop week;
run;
```

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Weekly</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
Defining and Referencing an Array

```sas
data Stats2(drop=i);
  set Stats;
  count=0;
  array height weight blpres pulse chol;
  do i = 1 to 5;
    if health{i} = 'AboveAve' then count+1;
  end;
run;
```

Based on the program and the results of the final data set, what should be in the blank space?

A. `i{5}`  
B. `count{5}`  
C. `height{5}`  
D. `health{5}`

---

Defining and Referencing an Array

When an asterisk is used to specify the number of elements, SAS is to determine the subscript by counting the variables in the array.

```sas
data newprices;
  set sashelp.pricedata;
  array parray{*} price:;
  total=sum(of parray{*});
  do num = 1 to dim(parray);
    parray{num} = parray{num}*1.10;
  end;
run;
```

The DIM function in the iterative DO statement returns the number of elements in an array.

---

Refer to Exercise 5 for Chapter 5 in Appendix A.
Creating Numeric Variables with an Array

An array can be based on existing variables or new variables.

```sas
data WeeklyDiff(drop=week);
  set Weekly;
  array run{4} week1 - week4;
  array diff{3} diff21 diff32 diff43;
  do week = 1 to 3;
    diff{week} = run{week+1}-run{week};
  end;
run;
```

Creating Numeric Variables with an Array

SAS creates variable names by concatenating the array name and the numbers 1, 2, 3, … n.

```sas
data WeeklyPct(drop=week);
  set Weekly;
  total=sum(of week1-week4);
  array run{4} week1 - week4;
  array pctwk{4};
  do week = 1 to 4;
    pctwk{week} = run{week}/total;
  end;
  format pctwk1-pctwk4 percent6.;
run;
```
Creating Character Variables with an Array

If an array is based on new character variables, the elements must be defined as character with a byte size.

```sas
data newnames(drop=i);
  set names;
  array first{3};
  array newfirst{3} $ 7;
  do i=1 to 3;
    newfirst{i}=propcase(first{i});
  end;
run;
```

<table>
<thead>
<tr>
<th></th>
<th>first1</th>
<th>first2</th>
<th>first3</th>
<th>newfirst1</th>
<th>newfirst2</th>
<th>newfirst3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Janet</td>
<td>Russel</td>
<td>Cheyl</td>
<td>Janet</td>
<td>Russel</td>
<td>Cheyl</td>
</tr>
</tbody>
</table>

Creating Character Variables with an Array

What is the byte size of `newfirst1-newfirst3` if a byte size is not specified in the ARRAY statement?

A. 0
B. 7 (the byte size of `first1-first3`)
C. 8
D. 18
5.4 Processing Data with Arrays

Initial Values

Initial values can be specified for the corresponding elements in the array.

```sas
data score(drop=x);
  infile 'raw-data-file';
  input name $ q1 $ q2 $ q3 $ q4 $ q5 $;
  array answer{5} q1-q5;
  array correct{5} $ 1 ('A','B','A','D','C');
  score=0;
  do x=1 to 5;
    if answer{x}=correct{x}
      then score+1;
  end;
run;
```

_TEMPERATURE_ Option

The _TEMPERARY_ option is used to create a list of temporary data elements.

```sas
data score(drop=x);
  infile 'raw-data-file';
  input name $ q1 $ q2 $ q3 $ q4 $ q5 $;
  array answer{5} q1-q5;
  array correct{5} $ 1 _temporary_ ('A','B','A','D','C');
  score=0;
  do x=1 to 5;
    if answer{x}=correct{x}
      then score+1;
  end;
run;
```
### _TEMPORARY_ Option

Which of the following statements is valid?

A. \[ \text{array revenue}\{3\} \_\text{temporary}_\text{ } \text{rev1-rev3} (12,15,22); \]

B. \[ \text{array revenue}\{3\} \$ \text{rev1-rev3} \_\text{temporary}_\text{ } ('12','15','22'); \]

C. \[ \text{array revenue}\{3\} \_\text{temporary}_\text{ } (12 15 22); \]

D. \[ \text{array revenue}\{3\} \_\text{temporary}_\text{ } 8 (12 15 22); \]
## 5.5 Answers to Questions

<table>
<thead>
<tr>
<th>Question Slide Number</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>D.</td>
</tr>
<tr>
<td>14</td>
<td>length Six Two $ 22;</td>
</tr>
</tbody>
</table>
| 18                    | • One = This  
                        | • Six = crazy  
                        | • Eight =  
                        | • MinusTwo = crazy  
                        | • Two = crazy |
| 22                    | ZOO (with 10 trailing blanks) |
| 28                    | D.     |
| 33                    | D.     |
| 37                    | A.     |
| 41                    | A.     |
| 43                    | C.     |
| 54                    | No     |
| 57                    | A.     |
| 60                    | BirthQtr = 2 |
| 63                    | B.     |
| 69                    | 0.5178 |
| 75                    | A.     |
| 78                    | Yes    |
| 82                    | D.     |
| 87                    | B.     |
| 103                   | No     |
| 112                   | A.     |
| 114                   | C.     |

(Continued on the next page.)
<table>
<thead>
<tr>
<th>Question Slide Number</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>118</td>
<td>1. 3</td>
</tr>
<tr>
<td></td>
<td>2. 6</td>
</tr>
<tr>
<td></td>
<td>3. 18</td>
</tr>
<tr>
<td>121</td>
<td>Yes</td>
</tr>
<tr>
<td>125</td>
<td>B.</td>
</tr>
<tr>
<td>131</td>
<td>Yes</td>
</tr>
<tr>
<td>134</td>
<td>B.</td>
</tr>
<tr>
<td>143</td>
<td>D.</td>
</tr>
<tr>
<td>151</td>
<td>C.</td>
</tr>
<tr>
<td>155</td>
<td>C.</td>
</tr>
</tbody>
</table>
# Chapter 6 Generating Reports

6.1 Creating Detail Reports with the PRINT Procedure ..............................................6-3

6.2 Creating Formats with the FORMAT Procedure .................................................6-22

6.3 Creating Frequency Tables with the FREQ Procedure ...................................6-30

6.4 Creating Summary Reports with the MEANS Procedure ...............................6-35

6.5 Directing Reports to External Files with ODS .................................................6-40

6.6 Answers to Questions .......................................................................................6-46
# 6.1 Creating Detail Reports with the PRINT Procedure

**PRINT Procedure**

The *PRINT procedure* creates a report of the variables and observations in a SAS data set. You can create a variety of reports ranging from a simple listing to a highly customized report that groups the data and calculates totals and subtotals for numeric variables.

### Partial Output

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bangkok</td>
<td>Boot</td>
<td>$9,576.00</td>
<td>$1,996.00</td>
</tr>
<tr>
<td>Bangkok</td>
<td>Sandal</td>
<td>$15,087.00</td>
<td>$3,230.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Boot</td>
<td>$160,589.00</td>
<td>$60,712.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Sandal</td>
<td>$21,483.00</td>
<td>$4,978.00</td>
</tr>
<tr>
<td></td>
<td>Region</td>
<td>$206,735.00</td>
<td>$70,916.00</td>
</tr>
</tbody>
</table>

*Created by Tony Smith, Chicago, IL*

---

**PRINT Procedure**

Which *two* of the following items *cannot* be accomplished with the PRINT procedure?

1. produce detail reports
2. produce summary reports
3. sort data values by one or more variables
4. produce column totals for numeric variables
5. replace variable values with formatted values
6. replace variable names with descriptive labels
7. choose only observations that meet a condition
8. select specific variables and control the order in which the variables appear
Example of the PRINT Procedure

```latex
options nodate nonumber ps=30 ls=64;
proc print data=sashelp.shoes noobs split='*';
var subsidiary product inventory sales;
where product='Boot' or product='Sandal';
sum inventory sales;
by region;
pageby region;
label inventory='Total*Inventory'
sales='Total*Sales';
format inventory sales dollar14.2;
title 'Boot and Sandal Report';
footnote 'Created by Tony Smith';
footnote2 'Chicago, IL';
run;
```

Refer to Exercise 1 for Chapter 6 in Appendix A.

**NOOBS Option**

The *NOOBS* option suppresses the column in the output that identifies each observation by number.

```latex
proc print data=sashelp.shoes noobs split='*';
```

By default, the PRINT procedure gives an observation column.
6.1  Creating Detail Reports with the PRINT Procedure

NOOBS Option

Boot and Sandal Report

Without NOOBS:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>Bangkok</td>
<td>Boot</td>
<td>$9,576.00</td>
<td>$1,996.00</td>
</tr>
<tr>
<td>59</td>
<td>Bangkok</td>
<td>Sandal</td>
<td>$15,087.00</td>
<td>$3,230.00</td>
</tr>
<tr>
<td>62</td>
<td>Seoul</td>
<td>Boot</td>
<td>$160,589.00</td>
<td>$60,712.00</td>
</tr>
<tr>
<td>65</td>
<td>Seoul</td>
<td>Sandal</td>
<td>$21,483.00</td>
<td>$4,978.00</td>
</tr>
</tbody>
</table>

Region=Asia

Total: $206,735.00 $70,916.00

Partial Output

With NOOBS:

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bangkok</td>
<td>Boot</td>
<td>$9,576.00</td>
<td>$1,996.00</td>
</tr>
<tr>
<td>Bangkok</td>
<td>Sandal</td>
<td>$15,087.00</td>
<td>$3,230.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Boot</td>
<td>$160,589.00</td>
<td>$60,712.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Sandal</td>
<td>$21,483.00</td>
<td>$4,978.00</td>
</tr>
</tbody>
</table>

Region=Asia

Subsidiary: Bangkok, Seoul
Product: Boot, Sandal
Total: $206,735.00 $70,916.00

VAR Statement

The VAR statement selects variables that appear in the report and determines the order of the variables.

```sql
var subsidiary product inventory sales;
```

By default, the PRINT procedure displays all variables in the order that the variables are stored in the data set.
**VAR Statement**

The following list report is requested:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Product</th>
<th>Stores</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Boot</td>
<td>12</td>
<td>$769</td>
</tr>
<tr>
<td>2</td>
<td>Men's Casual</td>
<td>4</td>
<td>$2,284</td>
</tr>
<tr>
<td>3</td>
<td>Men's Dress</td>
<td>7</td>
<td>$2,433</td>
</tr>
<tr>
<td>4</td>
<td>Sandal</td>
<td>10</td>
<td>$1,861</td>
</tr>
</tbody>
</table>

Which VAR statement creates the desired output?

A. `var product stores returns;`

B. `var product, stores, returns;`

C. `var obs product stores returns;`

D. `var obs, product, stores, returns;`

**WHERE Statement**

The *WHERE statement* subsets the input data set by specifying certain conditions that each observation must meet before it is available for the report.

`where product='Boot' or product='Sandal';`

- The WHERE statement does not alter the original data set.
- Use only one WHERE statement in a step unless you use a WHERE SAME AND or WHERE ALSO statement with a WHERE statement.
- Character values are case sensitive.
WHERE Statement

Which WHERE statement does not produce results that are identical to those of the other three WHERE statements?

A. `WHERE product='Boot' or 'Sandal';`
B. `WHERE product in ('Boot' 'Sandal');`
C. `WHERE product in ('Boot','Sandal');`
D. `WHERE product='Boot' or product='Sandal';`

WHERE Statement

Examples:
- `WHERE sales > 100000;`
- `WHERE sales eq .;`
- `WHERE name = 'Smith';`
- `WHERE name = ' ';`
- `WHERE sales ge 100000 and name = 'Smith';`
- `WHERE sales ge 100000 or name = 'Smith';`
- `WHERE revenue >= 150 and revenue <= 999;`
- `WHERE revenue between 150 and 999;`
- `WHERE revenue not between 150 and 999;`
- `WHERE month contains 'uary';`
- `WHERE birthdate > '11JUL1968';`
WHERE Statement

The following SAS program is submitted:

```
proc print data=sales;
  where month=2;
  where sales>100;
run;
```

Which of the following statements is true regarding the program?
A. Only the first WHERE statement will be used.
B. Only the second WHERE statement will be used.
C. Both WHERE statements will be used with a logical OR between the statements.
D. Both WHERE statements will be used with a logical AND between the statements.

SUM Statement

The SUM statement totals values of numeric variables.

```
sum inventory sales;
```

Partial Output

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bangkok</td>
<td>Boot</td>
<td>$9,576.00</td>
<td>$1,996.00</td>
</tr>
<tr>
<td>Bangkok</td>
<td>Sandal</td>
<td>$15,087.00</td>
<td>$3,230.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Boot</td>
<td>$160,589.00</td>
<td>$60,712.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Sandal</td>
<td>$21,483.00</td>
<td>$4,978.00</td>
</tr>
<tr>
<td>Region</td>
<td></td>
<td>$206,735.00</td>
<td>$70,916.00</td>
</tr>
</tbody>
</table>

The SUM statement always gives grand totals and gives subtotals if used with a BY statement.
### BY Statement

The **BY statement** produces a separate section of the report for each BY group.

```
by region;
```

Partial Output

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bangkok</td>
<td>Boot</td>
<td>$9,676.00</td>
<td>$1,996.00</td>
</tr>
<tr>
<td>Bangkok</td>
<td>Sandal</td>
<td>$15,087.00</td>
<td>$3,230.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Boot</td>
<td>$160,589.00</td>
<td>$60,712.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Sandal</td>
<td>$21,483.00</td>
<td>$4,978.00</td>
</tr>
<tr>
<td>Region</td>
<td></td>
<td>$206,735.00</td>
<td>$70,916.00</td>
</tr>
</tbody>
</table>

Data must be indexed or sorted to use a BY statement.

---

## SUM and BY Statements

### Partial Output

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copenhagen</td>
<td>Boot</td>
<td>$4,657.00</td>
<td>$1,663.00</td>
</tr>
<tr>
<td>Geneva</td>
<td>Boot</td>
<td>$171,030.00</td>
<td>$41,341.00</td>
</tr>
<tr>
<td>Geneva</td>
<td>Sandal</td>
<td>$3,529.00</td>
<td>$736.00</td>
</tr>
<tr>
<td>Heidelberg</td>
<td>Boot</td>
<td>$301,779.00</td>
<td>$65,010.00</td>
</tr>
<tr>
<td>Heidelberg</td>
<td>Sandal</td>
<td>$4,618.00</td>
<td>$977.00</td>
</tr>
<tr>
<td>Lisbon</td>
<td>Boot</td>
<td>$341,911.00</td>
<td>$76,349.00</td>
</tr>
<tr>
<td>Lisbon</td>
<td>Sandal</td>
<td>$24,253.00</td>
<td>$1,650.00</td>
</tr>
<tr>
<td>London</td>
<td>Boot</td>
<td>$289,527.00</td>
<td>$54,449.00</td>
</tr>
<tr>
<td>London</td>
<td>Sandal</td>
<td>$11,111.00</td>
<td>$5,317.00</td>
</tr>
<tr>
<td>Madrid</td>
<td>Boot</td>
<td>$1,027.00</td>
<td>$1,179.00</td>
</tr>
<tr>
<td>Paris</td>
<td>Boot</td>
<td>$41,506.00</td>
<td>$19,196.00</td>
</tr>
<tr>
<td>Paris</td>
<td>Sandal</td>
<td>$23,816.00</td>
<td>$1,520.00</td>
</tr>
<tr>
<td>Rome</td>
<td>Boot</td>
<td>$209,271.00</td>
<td>$36,244.00</td>
</tr>
<tr>
<td>Rome</td>
<td>Sandal</td>
<td>$4,611.00</td>
<td>$1,249.00</td>
</tr>
<tr>
<td>Region</td>
<td></td>
<td>$1,432,646.00</td>
<td>$307,380.00</td>
</tr>
</tbody>
</table>

---

### Region=Western Europe

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copenhagen</td>
<td>Boot</td>
<td>$4,657.00</td>
<td>$1,663.00</td>
</tr>
<tr>
<td>Geneva</td>
<td>Boot</td>
<td>$171,030.00</td>
<td>$41,341.00</td>
</tr>
<tr>
<td>Geneva</td>
<td>Sandal</td>
<td>$3,529.00</td>
<td>$736.00</td>
</tr>
<tr>
<td>Heidelberg</td>
<td>Boot</td>
<td>$301,779.00</td>
<td>$65,010.00</td>
</tr>
<tr>
<td>Heidelberg</td>
<td>Sandal</td>
<td>$4,618.00</td>
<td>$977.00</td>
</tr>
<tr>
<td>Lisbon</td>
<td>Boot</td>
<td>$341,911.00</td>
<td>$76,349.00</td>
</tr>
<tr>
<td>Lisbon</td>
<td>Sandal</td>
<td>$24,253.00</td>
<td>$1,650.00</td>
</tr>
<tr>
<td>London</td>
<td>Boot</td>
<td>$289,527.00</td>
<td>$54,449.00</td>
</tr>
<tr>
<td>London</td>
<td>Sandal</td>
<td>$11,111.00</td>
<td>$5,317.00</td>
</tr>
<tr>
<td>Madrid</td>
<td>Boot</td>
<td>$1,027.00</td>
<td>$1,179.00</td>
</tr>
<tr>
<td>Paris</td>
<td>Boot</td>
<td>$41,506.00</td>
<td>$19,196.00</td>
</tr>
<tr>
<td>Paris</td>
<td>Sandal</td>
<td>$23,816.00</td>
<td>$1,520.00</td>
</tr>
<tr>
<td>Rome</td>
<td>Boot</td>
<td>$209,271.00</td>
<td>$36,244.00</td>
</tr>
<tr>
<td>Rome</td>
<td>Sandal</td>
<td>$4,611.00</td>
<td>$1,249.00</td>
</tr>
<tr>
<td>Region</td>
<td></td>
<td>$1,432,646.00</td>
<td>$307,380.00</td>
</tr>
</tbody>
</table>

### Region=Asia

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bangkok</td>
<td>Boot</td>
<td>$9,676.00</td>
<td>$1,996.00</td>
</tr>
<tr>
<td>Bangkok</td>
<td>Sandal</td>
<td>$15,087.00</td>
<td>$3,230.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Boot</td>
<td>$160,589.00</td>
<td>$60,712.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Sandal</td>
<td>$21,483.00</td>
<td>$4,978.00</td>
</tr>
<tr>
<td>Region</td>
<td></td>
<td>$206,735.00</td>
<td>$70,916.00</td>
</tr>
</tbody>
</table>
The *PAGEBY statement* puts each separate section of a BY group on separate pages.

```
pageby region;
```

The PAGEBY statement must name a variable that appears in the BY statement.

**Partial Output**

---

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al-Khobar</td>
<td>Boot</td>
<td>$44,658.00</td>
<td>$15,062.00</td>
</tr>
<tr>
<td>Al-Khobar</td>
<td>Sandal</td>
<td>$15,343.00</td>
<td>$1,800.00</td>
</tr>
<tr>
<td>Dubai</td>
<td>Boot</td>
<td>$493,259.00</td>
<td>$90,972.00</td>
</tr>
<tr>
<td>Dubai</td>
<td>Sandal</td>
<td>$58,985.00</td>
<td>$17,492.00</td>
</tr>
<tr>
<td>Tel Aviv</td>
<td>Boot</td>
<td>$222,165.00</td>
<td>$65,248.00</td>
</tr>
<tr>
<td>Tel Aviv</td>
<td>Sandal</td>
<td>$71,094.00</td>
<td>$16,314.00</td>
</tr>
<tr>
<td>Region</td>
<td></td>
<td>$814,504.00</td>
<td>$206,468.00</td>
</tr>
</tbody>
</table>

---

---

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bangkok</td>
<td>Boot</td>
<td>$9,576.00</td>
<td>$1,996.00</td>
</tr>
<tr>
<td>Bangkok</td>
<td>Sandal</td>
<td>$15,087.00</td>
<td>$3,230.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Boot</td>
<td>$160,589.00</td>
<td>$60,712.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Sandal</td>
<td>$21,483.00</td>
<td>$4,978.00</td>
</tr>
<tr>
<td>Region</td>
<td></td>
<td>$206,735.00</td>
<td>$70,916.00</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Budapest</td>
<td>Boot</td>
<td>$317,515.00</td>
<td>$74,102.00</td>
</tr>
<tr>
<td>Budapest</td>
<td>Sandal</td>
<td>$4,894.00</td>
<td>$1,814.00</td>
</tr>
<tr>
<td>Moscow</td>
<td>Boot</td>
<td>$329,604.00</td>
<td>$67,476.00</td>
</tr>
<tr>
<td>Prague</td>
<td>Boot</td>
<td>$341,326.00</td>
<td>$86,215.00</td>
</tr>
<tr>
<td>Warsaw</td>
<td>Boot</td>
<td>$363,358.00</td>
<td>$78,992.00</td>
</tr>
<tr>
<td>Region</td>
<td></td>
<td>$1,361,041.00</td>
<td>$310,501.00</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al-Khobar</td>
<td>Boot</td>
<td>$44,658.00</td>
<td>$15,062.00</td>
</tr>
<tr>
<td>Al-Khobar</td>
<td>Sandal</td>
<td>$13,343.00</td>
<td>$1,800.00</td>
</tr>
<tr>
<td>Dubai</td>
<td>Boot</td>
<td>$493,259.00</td>
<td>$90,972.00</td>
</tr>
<tr>
<td>Dubai</td>
<td>Sandal</td>
<td>$58,985.00</td>
<td>$17,492.00</td>
</tr>
<tr>
<td>Tel Aviv</td>
<td>Boot</td>
<td>$222,165.00</td>
<td>$65,248.00</td>
</tr>
<tr>
<td>Tel Aviv</td>
<td>Sandal</td>
<td>$71,094.00</td>
<td>$16,314.00</td>
</tr>
<tr>
<td>Region</td>
<td></td>
<td>$814,504.00</td>
<td>$206,468.00</td>
</tr>
</tbody>
</table>
6.1 Creating Detail Reports with the PRINT Procedure

BY and PAGEBY Statements

--- Region=United States Subsidiary=Chicago ---
--- Region=United States Subsidiary=New York ---
--- Region=United States Subsidiary=Seattle ---

Which statements give the desired layout?

A. `by region;
   pageby subsidiary;`

B. `by region subsidiary;
   pageby region;`

C. `by region subsidiary;
   pageby subsidiary;`

D. `by region subsidiary;
   pageby region subsidiary;`

ID Statement

The `ID statement` specifies the variable(s) to print at the beginning of each row instead of an observation number.

```
id region;
by region;
```

When used with a BY statement, the ID statement eliminates the BY line and suppresses repetitious printing of the BY variable(s).

Partial Output

```
<table>
<thead>
<tr>
<th>Region</th>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Western Europe</td>
<td>Copenhagen</td>
<td>Boot</td>
<td>$4,657.00</td>
<td>$1,663.00</td>
</tr>
<tr>
<td></td>
<td>Geneva</td>
<td>Boot</td>
<td>$171,030.00</td>
<td>$41,341.00</td>
</tr>
<tr>
<td></td>
<td>Geneva</td>
<td>Sandal</td>
<td>$3,529.00</td>
<td>$736.00</td>
</tr>
<tr>
<td></td>
<td>Heidelberg</td>
<td>Boot</td>
<td>$301,779.00</td>
<td>$65,610.00</td>
</tr>
<tr>
<td></td>
<td>Heidelberg</td>
<td>Sandal</td>
<td>$4,618.00</td>
<td>$977.00</td>
</tr>
</tbody>
</table>
```
The **LABEL statement** assigns descriptive labels to variable names.

```
label inventory='Total*Inventory'
sales='Total*Sales';
```

### Partial Output

<table>
<thead>
<tr>
<th>Obs</th>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>Bangkok</td>
<td>Boot</td>
<td>$9,576.00</td>
<td>$1,996.00</td>
</tr>
<tr>
<td>59</td>
<td>Bangkok</td>
<td>Sandal</td>
<td>$15,087.00</td>
<td>$3,230.00</td>
</tr>
<tr>
<td>62</td>
<td>Seoul</td>
<td>Boot</td>
<td>$160,589.00</td>
<td>$60,712.00</td>
</tr>
<tr>
<td>65</td>
<td>Seoul</td>
<td>Sandal</td>
<td>$21,483.00</td>
<td>$4,978.00</td>
</tr>
<tr>
<td></td>
<td>Region</td>
<td></td>
<td>$206,735.00</td>
<td>$70,916.00</td>
</tr>
</tbody>
</table>

A label can be up to 256 characters.

### LABEL Statement

A **LABEL statement** must also be accompanied by an option in the PROC PRINT statement in order for the labels to appear.

Which options are used in conjunction with the LABEL statement?

1. APPEAR
2. DESCRIPTION
3. LABEL
4. NOW
5. PRINT
6. SPLIT=
6.1 Creating Detail Reports with the PRINT Procedure

**LABEL Statement**

The LABEL option enforces variables' labels as column headings.

```
proc print data=sashelp.shoes noobs label;
```

The SPLIT= option specifies the split character, which controls line breaks in column headers and implies the use of labels.

```
proc print data=sashelp.shoes noobs split='*';
```

---

**FORMAT Statement**

The FORMAT statement associates formats to variable values.

```
format inventory sales dollar14.2;
```

Partial Output

<table>
<thead>
<tr>
<th>Obs</th>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>Bangkok</td>
<td>Boot</td>
<td>$9,576.00</td>
<td>$1,996.00</td>
</tr>
<tr>
<td>59</td>
<td>Bangkok</td>
<td>Sandal</td>
<td>$15,087.00</td>
<td>$3,230.00</td>
</tr>
<tr>
<td>62</td>
<td>Seoul</td>
<td>Boot</td>
<td>$160,589.00</td>
<td>$60,712.00</td>
</tr>
<tr>
<td>65</td>
<td>Seoul</td>
<td>Sandal</td>
<td>$21,483.00</td>
<td>$4,978.00</td>
</tr>
<tr>
<td>-----</td>
<td>------------</td>
<td>---------</td>
<td>-----------------</td>
<td>------------</td>
</tr>
<tr>
<td>57</td>
<td>Bangkok</td>
<td></td>
<td>$206,735.00</td>
<td>$70,916.00</td>
</tr>
</tbody>
</table>

A format is an instruction that SAS uses to write data values.
**FORMAT Statement**

Formats have the following form:

<$>format<w>.<d>

- `<$>` indicates a character format
- `format` is the format name
- `<w>` is the total width (including decimal places and special characters)
- `<d>` is the required delimiter
- `<.` is the number of decimal places

**FORMAT Statement**

<table>
<thead>
<tr>
<th>Stored Value</th>
<th>Format</th>
<th>Displayed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Washington</td>
<td>$4.</td>
<td>Wash</td>
</tr>
<tr>
<td>1234.4567</td>
<td>8.0</td>
<td>1234</td>
</tr>
<tr>
<td>1234.4567</td>
<td>8.2</td>
<td>1234.46</td>
</tr>
<tr>
<td>1234.4567</td>
<td>comma8.2</td>
<td>1,234.46</td>
</tr>
<tr>
<td>1234.4567</td>
<td>dollar9.2</td>
<td>$1,234.46</td>
</tr>
</tbody>
</table>
### FORMAT Statement

**Partial Output**

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copenhagen</td>
<td>Boot</td>
<td>$4,657.00</td>
<td>$1,663.00</td>
</tr>
<tr>
<td>Geneva</td>
<td>Boot</td>
<td>$3,529.00</td>
<td>$736.00</td>
</tr>
<tr>
<td>Geneva</td>
<td>Sandal</td>
<td>$301,779.00</td>
<td>$65,610.00</td>
</tr>
<tr>
<td>Heidelberg</td>
<td>Boot</td>
<td>$4,618.00</td>
<td>$977.00</td>
</tr>
<tr>
<td>Heidelberg</td>
<td>Sandal</td>
<td>$341,911.00</td>
<td>$76,349.00</td>
</tr>
<tr>
<td>Lisbon</td>
<td>Boot</td>
<td>$289,527.00</td>
<td>$54,449.00</td>
</tr>
<tr>
<td>Lisbon</td>
<td>Sandal</td>
<td>$11,111.00</td>
<td>$5,217.00</td>
</tr>
<tr>
<td>Madrid</td>
<td>Boot</td>
<td>$1,027.00</td>
<td>$1,179.00</td>
</tr>
<tr>
<td>Paris</td>
<td>Boot</td>
<td>$41,506.00</td>
<td>$19,196.00</td>
</tr>
<tr>
<td>Paris</td>
<td>Sandal</td>
<td>$23,816.00</td>
<td>$1,520.00</td>
</tr>
<tr>
<td>Rome</td>
<td>Boot</td>
<td>$209,271.00</td>
<td>$36,244.00</td>
</tr>
<tr>
<td>Rome</td>
<td>Sandal</td>
<td>$4,611.00</td>
<td>$1,249.00</td>
</tr>
</tbody>
</table>

### Q

What minimum widths are needed to complete the FORMAT statement for this desired output?

```plaintext
format inventory dollar___2 sales dollar___2;
```

### FORMAT Statement

The following program is submitted:

```
proc print data=sales;
  format sales dollar7.2;
run;
```

What is the result if the format width for `dollar` is too narrow to represent the value?

A. The program fails execution due to errors.
B. SAS supplies a missing value to the `sales` variable.
C. SAS fits the value into the space available in the best way that it can.
D. The program runs with warnings and eliminates the `sales` variable.
### FORMAT Statement

<table>
<thead>
<tr>
<th>Stored Value</th>
<th>Format</th>
<th>Displayed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>17332</td>
<td>mmdyy6.</td>
<td>061507</td>
</tr>
<tr>
<td>17332</td>
<td>mmdyy8.</td>
<td>06/15/07</td>
</tr>
<tr>
<td>17332</td>
<td>mmdyy10.</td>
<td>06/15/2007</td>
</tr>
<tr>
<td>17332</td>
<td>date7.</td>
<td>15JUN07</td>
</tr>
<tr>
<td>17332</td>
<td>date9.</td>
<td>15JUN2007</td>
</tr>
<tr>
<td>17332</td>
<td>ddmmyy8.</td>
<td>15/06/07</td>
</tr>
<tr>
<td>17332</td>
<td>wordate.</td>
<td>June 15, 2007</td>
</tr>
<tr>
<td>17332</td>
<td>weekdate.</td>
<td>Friday, June 15, 2007</td>
</tr>
<tr>
<td>17332</td>
<td>monyy7.</td>
<td>JUN2007</td>
</tr>
</tbody>
</table>

### Desired report:

<table>
<thead>
<tr>
<th>Obs</th>
<th>birth</th>
<th>hired</th>
<th>retired</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>02/17/1941</td>
<td>05/01/1976</td>
<td>31DEC2006</td>
</tr>
</tbody>
</table>

What two corrections must be made to the following statement to obtain the desired report?

```sql
format birth hired mmdyy8. retired date9;
```
LABEL and FORMAT Statements

LABEL and FORMAT statements assigned in a PROC step are considered temporary attributes (apply only for the duration of the step). LABEL and FORMAT statements assigned in a DATA step are considered permanent attributes (stored in the descriptor portion).

### Alphabetic List of Variables and Attributes

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Format</th>
<th>Informat</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Inventory</td>
<td>Num</td>
<td>8</td>
<td>DOLLAR12.</td>
<td>DOLLAR12.</td>
<td>Total Inventory</td>
</tr>
<tr>
<td>2</td>
<td>Product</td>
<td>Char</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Region</td>
<td>Char</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Returns</td>
<td>Num</td>
<td>8</td>
<td>DOLLAR12.</td>
<td>DOLLAR12.</td>
<td>Total Returns</td>
</tr>
<tr>
<td>5</td>
<td>Sales</td>
<td>Num</td>
<td>8</td>
<td>DOLLAR12.</td>
<td>DOLLAR12.</td>
<td>Total Sales</td>
</tr>
<tr>
<td>4</td>
<td>Stores</td>
<td>Num</td>
<td>8</td>
<td></td>
<td></td>
<td>Number of Stores</td>
</tr>
<tr>
<td>3</td>
<td>Subsidiary</td>
<td>Char</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### TITLE Statement

The TITLE statement specifies up to 10 lines of text at the top of output.

```plaintext
title 'Boot and Sandal Report';
```

**Partial Output**

```
Boot and Sandal Report
----------------- Region=Asia --------------------------

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bangkok</td>
<td>Boot</td>
<td>$9,576.00</td>
<td>$1,996.00</td>
</tr>
<tr>
<td>Bangkok</td>
<td>Sandal</td>
<td>$15,087.00</td>
<td>$3,230.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Boot</td>
<td>$160,589.00</td>
<td>$60,712.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Sandal</td>
<td>$21,483.00</td>
<td>$4,978.00</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>-----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Region</td>
<td></td>
<td>$206,735.00</td>
<td>$70,916.00</td>
</tr>
</tbody>
</table>
```

TITLE is the same as TITLE1.
FOOTNOTE Statement

The FOOTNOTE statement specifies up to 10 lines of text at the bottom of output.

```
footnote 'Created by Tony Smith';
footnote2 'Chicago, IL';
```

Partial Output

<table>
<thead>
<tr>
<th>Subsidiary</th>
<th>Product</th>
<th>Total Inventory</th>
<th>Total Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bangkok</td>
<td>Boot</td>
<td>$9,576.00</td>
<td>$1,996.00</td>
</tr>
<tr>
<td>Bangkok</td>
<td>Sandal</td>
<td>$15,087.00</td>
<td>$3,230.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Boot</td>
<td>$160,589.00</td>
<td>$60,712.00</td>
</tr>
<tr>
<td>Seoul</td>
<td>Sandal</td>
<td>$21,483.00</td>
<td>$4,978.00</td>
</tr>
<tr>
<td>Region</td>
<td></td>
<td>$206,735.00</td>
<td>$70,916.00</td>
</tr>
</tbody>
</table>

FOOTNOTE is the same as FOOTNOTE1.

TITLE and FOOTNOTE Statements

The TITLE and FOOTNOTE statements are global statements, which means that the statements stay in effect until they are canceled or changed, or you end your SAS session.

The code `title;` cancels all titles.
The code `footnote;` cancels all footnotes.

TITLEn or FOOTNOTEn
  - replaces a previous title or footnote with the same number
  - cancels all titles or footnotes with higher numbers.
TITLE and FOOTNOTE Statements

The following SAS program is submitted:

```sas
proc print data=shoes1;
  title1 'Shoe Store';
  title2 'Report One';
  title3 'Accounting';
run;
proc print data=shoes2;
  title2 'Report Two';
run;
```

What titles appear in the second procedure output?

A. Report Two
B. Shoe Store Report Two
C. Report Two Accounting
D. Shoe Store Report Two Accounting

OPTIONS Statement

The *OPTIONS statement* changes the value of one or more SAS system options.

```sas
options nodate nonumber ps=30 ls=64;
```

The OPTIONS statement is a global statement, which means that the options remain in effect until they are canceled or changed, or you end your SAS session.

The OPTIONS statement is not usually included in a step.

Some system options change the appearance of a report.
OPTIONS Statement

<table>
<thead>
<tr>
<th>System Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>NODATE</td>
</tr>
<tr>
<td>NODTRESET</td>
<td>DTRESET</td>
</tr>
<tr>
<td>NUMBER</td>
<td>NONUMBER</td>
</tr>
<tr>
<td>PAGENO=n</td>
<td>Specifies a beginning page number for the next page of output.</td>
</tr>
<tr>
<td>LS=n</td>
<td>LINESIZE=n</td>
</tr>
<tr>
<td>PS=n</td>
<td>PAGESIZE=n</td>
</tr>
</tbody>
</table>

The following SAS program is submitted.

```sas
options nodate nonumber;
proc print data=shoes1;
run;
options pageno=1;
proc print data=shoes2;
run;
```

What is the result of the second report?
A. The second report has a date and no page number.
B. The second report has a date and a page number of 1.
C. The second report has no date and no page number.
D. The second report has no date and a page number of 1.
Common Statements

The following statements have the same function in a number of Base SAS procedures:
- WHERE
- LABEL
- FORMAT
- TITLE
- FOOTNOTE
- OPTIONS

Refer to Exercise 2 for Chapter 6 in Appendix A.
6.2 Creating Formats with the FORMAT Procedure

**FORMAT Procedure**

The *FORMAT procedure* enables you to define your own formats for variable values.

```plaintext
proc format;
  value $gender 'F' = 'Female'
                'M' = 'Male'
                other = 'Miscoded';
  value wtrange low - <100 = 'Under 100 lbs'
                   100 - high = '100+ lbs';
run;
```

Formats determine how variable values are printed.

Refer to Exercise 3 for Chapter 6 in Appendix A.

**Naming Convention**

Format names must be

- 32 characters or less
- different than the name of a format supplied by SAS.

Character formats start with a $, followed by a letter or underscore.

```plaintext
proc format;
  value $gender 'F' = 'Female'
                'M' = 'Male'
                other = 'Miscoded';
  value wtrange low - <100 = 'Under 100 lbs'
                   100 - high = '100+ lbs';
run;
```

Numerical formats start with a letter or underscore.
Single Values

On the left side of the equal sign, you can have single values. Character values should be enclosed in quotation marks.

```
proc format;
  value $gender 'F' = 'Female'
     'M' = 'Male'
     other = 'Miscoded';
  value wtrange low - <100 = 'Under 100 lbs'
     100 - high = '100+ lbs';
run;
```

You can have multiple single values with commas separating the values.

```
value $gender 'F','FEM','FEMALE' = 'Female'
     'M','MAL','MALE' = 'Male';
```

Single Values

A format needs to be created for a numeric variable where 0 means NO, 1 means YES, 3 and 4 means UNSURE, and . means MISSING.

```
proc format;
  value code '0' = 'NO'
    '1' = 'YES'
    '3','4' = 'UNSURE'
    .   = 'MISSING';
run;
```

What is the error in the PROC FORMAT step?
Ranges

On the left side of the equal sign, you can have ranges.

```sas
proc format;
  value $gender 'F' = 'Female'
                  'M' = 'Male'
           other = 'Miscoded';
  value wtrange low - <100 = 'Under 100 lbs'
                    100 - high = '100+ lbs';
run;
```

For character ranges, each string should be enclosed in quotation marks (example: 'A' – 'Z').

Ranges

Given the following code:

```sas
proc format;
  value group  0 - 50  = 'First Half'
                   51 - 100 = 'Second Half';
run;
```

What will be the formatted value of 50.5?
A. 50.5
B. First Half
C. Second Half
D. missing value
Ranges

The less than (<) sign excludes values from ranges.
- Put < after the value if you exclude the first value in a range.
- Put < before the value if you exclude the last value in a range.

<table>
<thead>
<tr>
<th>Range</th>
<th>Includes 50</th>
<th>Includes 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>50 - 100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50 - &lt; 100</td>
<td>Includes 50</td>
<td>Excludes 100</td>
</tr>
<tr>
<td>50 &lt; - 100</td>
<td>Excludes 50</td>
<td>Includes 100</td>
</tr>
<tr>
<td>50 &lt; - &lt; 100</td>
<td>Excludes 50</td>
<td>Excludes 100</td>
</tr>
</tbody>
</table>

Given the following code:

```sas
proc format;
  value group  0 - < 50 = 'First Half'
                  50 - 100 = 'Second Half';
run;
```

What will be the formatted value of 50.5?
A. 50.5
B. First Half
C. Second Half
D. missing value
Keywords

- OTHER matches all values that do not match any other value or range.
- LOW encompasses lowest possible value.
  LOW does not include missing for numeric variables.
  LOW does include missing for character variables.
- HIGH encompasses highest possible value.
- LOW - HIGH encompasses all values.

```sas
proc format;
  value $gender 'F' = 'Female'
                 'M' = 'Male'
                 other = 'Miscoded';
  value wtrange low - <100 = 'Under 100 lbs'
                   100 - high = '100+ lbs';
run;
```

Q

Keywords

What is wrong with the keyword in the following example?

```sas
proc format;
  value $gender 'F' = 'Female'
                   'M' = 'Male'
                   'other' = 'Miscoded';
run;
```
Formatted Values

On the right side of the equal sign, you have the formatted values.

- Formatted values
  - are typically quoted strings
  - can be up to 32,767 characters.

```sas
proc format;
  value $gender 'F' = 'Female'
                   'M' = 'Male'
                  other = 'Miscoded';
  value wtrange low - <100 = 'Under 100 lbs'
                 100 - high = '100+ lbs';
run;
```

Creating and Using Formats

Create format (no period in format name):

```sas
proc format;
  value $gender 'F' = 'Female'
                   'M' = 'Male'
                  other = 'Miscoded';
  value wtrange low - <100 = 'Under 100 lbs'
                 100 - high = '100+ lbs';
run;
```

Use format (period in format name):

```sas
options nodate nonumber ps=30 ls=64;
proc print data=sashelp.class;
  var name sex weight;
  format sex $gender. weight wtrange.;
run;
```
Creating and Using Formats

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Desired Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obs</td>
<td>name</td>
</tr>
<tr>
<td>1</td>
<td>Jane</td>
</tr>
<tr>
<td>2</td>
<td>Tom</td>
</tr>
<tr>
<td>3</td>
<td>Mark</td>
</tr>
<tr>
<td>4</td>
<td>Sue</td>
</tr>
</tbody>
</table>

```
proc format;
  value $testfmt 'a' = 'Excellent'
    'b' = 'Good'
    'c' = 'Average'
    'd','f' = 'Needs Improvement'
    ' ' = 'Incomplete'
    other = 'Miscoded';
run;
```

```
proc print data=test;
  format test $testfmt;
run;
```

What two issues cause this program not to give the desired results?

Permanent Formats

By default, user-defined formats are stored in the `work.formats` catalog and exist only for the duration of the SAS session.

```
proc format library = sasuser.myfmts;
  value $gender 'F'  = 'Female'
       'M'  = 'Male'
       other = 'Miscoded';
run;
```

The LIBRARY= option is used to create permanent formats. LIBRARY= can point to either a library or a catalog. If only a library is specified, then SAS uses the catalog name FORMATS.
Format System Options

Which system option specifies the order in which format catalogs are searched?

A. FMTCATALOG
B. FMTERR
C. FMTORDER
D. FMTSEARCH

Format System Options

- The FMTERR system option (default) specifies that when SAS cannot find a specified variable format, it generates an error message and stops processing.
- The NOFMTERR replaces missing formats with the $w$ or $w$. default format and continues processing.

```plaintext
options nofmterr
   fmtsearch=(sasuser sasuser.myfmts);
```

- The FMTSEARCH= system option specifies the order in which format catalogs are searched. When a library is specified without a catalog, SAS uses FORMATS as the default catalog name. The default value for FMTSEARCH is (WORK LIBRARY).
6.3 Creating Frequency Tables with the FREQ Procedure

FREQ Procedure
The *FREQ* procedure produces one-way to *n*-way frequency tables. By default, the procedure generates one-way frequency tables for all data set variables.

```
proc freq data=sashelp.orsales;
run;
```

The data set *sashelp.orsales* has eight variables. Therefore, eight one-way frequency tables are created.

TABLES Statement
The *TABLES* statement requests one-way to *n*-way frequency tables and statistics for those tables.

```
proc freq data=sashelp.orsales;
tables product_line year;
run;
```
Two-Way Tables

An asterisk between two variables produces a two-way table.

```sql
proc freq data=sashelp.orsales;
  where product_line in ('Outdoors','Sports');
  tables product_line * year ;
run;
```

Table of Product_Line by Year

<table>
<thead>
<tr>
<th>Product_Line (Product Line)</th>
<th>Year (Year)</th>
<th>Frequency</th>
<th>Percent</th>
<th>Row Pct</th>
<th>Col Pct</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outdoors</td>
<td>1999</td>
<td>28</td>
<td>6.25</td>
<td>25.00</td>
<td>25.00</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>28</td>
<td>6.25</td>
<td>25.00</td>
<td>25.00</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>2001</td>
<td>28</td>
<td>6.25</td>
<td>25.00</td>
<td>25.00</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>2002</td>
<td>28</td>
<td>6.25</td>
<td>25.00</td>
<td>25.00</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>448</td>
</tr>
</tbody>
</table>

| Sports                      | 1999        | 84        | 18.75   | 75.00   | 75.00   | 336   |
|                             | 2000        | 84        | 18.75   | 75.00   | 75.00   | 336   |
|                             | 2001        | 84        | 18.75   | 75.00   | 75.00   | 336   |
|                             | 2002        | 84        | 18.75   | 75.00   | 75.00   | 336   |
|                             | Total       |           |         |         |         | 448   |

<table>
<thead>
<tr>
<th>Total</th>
<th>112</th>
<th>112</th>
<th>112</th>
<th>112</th>
<th>100.00</th>
<th>448</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>25.00</td>
<td>25.00</td>
<td>25.00</td>
<td>25.00</td>
<td>100.00</td>
<td>100.00</td>
</tr>
</tbody>
</table>
Two-Way Tables

The following program is submitted:

```sas
proc freq data=sashelp.orsales ;
  tables quarter product_category year*product_line;
run;
```

Which of the following is true regarding the program?

A. One frequency table is produced.
B. Two frequency tables are produced.
C. Three frequency tables are produced.
D. Four frequency tables are produced.

---

CROSSLIST Option

The CROSSLIST option displays two-way tables in column format, instead of cell format.

```sas
proc freq data=sashelp.orsales ;
  where product_line in ('Outdoors','Sports');
  tables product_line*year / crosslist ;
run;
```

Options in the TABLES statement must come after a forward slash.

---

Copyright © 2013, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.
### The FREQ Procedure

#### Table of Product Line by Year

<table>
<thead>
<tr>
<th>Product Line</th>
<th>Year</th>
<th>Frequency</th>
<th>Percent</th>
<th>Row Percent</th>
<th>Column Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outdoors</td>
<td>1999</td>
<td>28</td>
<td>6.25</td>
<td>25.00</td>
<td>25.00</td>
</tr>
<tr>
<td></td>
<td>2000</td>
<td>28</td>
<td>6.25</td>
<td>25.00</td>
<td>25.00</td>
</tr>
<tr>
<td></td>
<td>2001</td>
<td>28</td>
<td>6.25</td>
<td>25.00</td>
<td>25.00</td>
</tr>
<tr>
<td></td>
<td>2002</td>
<td>28</td>
<td>6.25</td>
<td>25.00</td>
<td>25.00</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>112</td>
<td>25.00</td>
<td>100.00</td>
<td></td>
</tr>
</tbody>
</table>

| Sports       | 1999 | 84        | 18.75   | 25.00       | 75.00          |
|              | 2000 | 84        | 18.75   | 25.00       | 75.00          |
|              | 2001 | 84        | 18.75   | 25.00       | 75.00          |
|              | 2002 | 84        | 18.75   | 25.00       | 75.00          |
| **Total**    |      | 336       | 75.00   | 100.00      |                |

| Total        | 1999 | 112       | 25.00   | 100.00      |                |
|              | 2000 | 112       | 25.00   | 100.00      |                |
|              | 2001 | 112       | 25.00   | 100.00      |                |
|              | 2002 | 112       | 25.00   | 100.00      |                |
| **Total**    |      | 448       | 100.00  |             |                |

### Statistics

**Default Statistics**
- One-Way Tables: Frequency, Percent, Cumulative Frequency, Cumulative Percent
- Two-Way Tables: Frequency, Percent, Row Percent, Column Percent

**Options to Eliminate Statistics**
- One-Way Tables: NOPERCENT, NOCUM
- Two-Way Tables: NOFREQ, NOPERCENT, NOROW, NOCOL

```sas
tables product_line*year / options;
```
Statistics

The following frequency table is desired:

<table>
<thead>
<tr>
<th>Product Line</th>
<th>Frequency</th>
<th>Cumulative Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outdoors</td>
<td>112</td>
<td>112</td>
</tr>
<tr>
<td>Sports</td>
<td>336</td>
<td>448</td>
</tr>
</tbody>
</table>

Which statement creates the desired report?

A. `tables product_line;`
B. `tables product_line / nocum;`
C. `tables product_line / nopercent;`
D. `tables product_line / nocum nopercent;`

NLEVELS Option

The NLEVELS option displays the number of levels for all TABLES variables.

```
proc freq data=sashelp.orsales nlevels;
  tables year quarter
    product_line product_category;
run;
```

This output will appear before the four one-way frequency tables.

Refer to Exercise 4 for Chapter 6 in Appendix A.
6.4 Creating Summary Reports with the MEANS Procedure

**MEANS Procedure**

The *MEANS procedure* computes descriptive statistics for variables across all observations and within groups of observations.

```sas
proc means data=sashelp.prdsale
   maxdec=2 mean stddev;
   var predict actual;
   class country year;
run;
```

### Statistics

Which statistics does the MEANS procedure produce by default?

A. MEAN, STDDEV, SUM
B. MEAN, STDDEV, MIN, MAX
C. N, MEAN, STDDEV, MIN, MAX
D. N, MEAN, STDDEV, SUM, MIN, MAX
Statistics

By default, the MEANS procedure creates a report with N (number of nonmissing values), MEAN, STDDEV, MIN, and MAX.

```
proc means data=sashelp.prdsale;
run;
```

The MEANS Procedure

<table>
<thead>
<tr>
<th>Variable</th>
<th>Label</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTUAL</td>
<td>Actual Sales</td>
<td>1440</td>
<td>507.17847</td>
<td>287.03130</td>
<td>3.00000</td>
<td>1000.00</td>
</tr>
<tr>
<td>PREDICT</td>
<td>Predicted Sales</td>
<td>1440</td>
<td>490.48264</td>
<td>285.76679</td>
<td>0.00000</td>
<td>1000.00</td>
</tr>
<tr>
<td>QUARTER</td>
<td>Quarter</td>
<td>1440</td>
<td>2.50000</td>
<td>1.11842</td>
<td>1.00000</td>
<td>4.00000</td>
</tr>
<tr>
<td>YEAR</td>
<td>Year</td>
<td>1440</td>
<td>1993.50</td>
<td>0.50017</td>
<td>1993.00</td>
<td>1994.00</td>
</tr>
<tr>
<td>MONTH</td>
<td>Month</td>
<td>1440</td>
<td>12403.00</td>
<td>210.629</td>
<td>12054.00</td>
<td>12753.00</td>
</tr>
</tbody>
</table>

```
proc means data=sashelp.prdsale sum range;
run;
```

The MEANS Procedure

<table>
<thead>
<tr>
<th>Variable</th>
<th>Label</th>
<th>Sum</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTUAL</td>
<td>Actual Sales</td>
<td>730337.00</td>
<td>997.00000000</td>
</tr>
<tr>
<td>PREDICT</td>
<td>Predicted Sales</td>
<td>706295.00</td>
<td>1000.00</td>
</tr>
<tr>
<td>QUARTER</td>
<td>Quarter</td>
<td>3600.00</td>
<td>3.00000000</td>
</tr>
<tr>
<td>YEAR</td>
<td>Year</td>
<td>2870640.00</td>
<td>1.00000000</td>
</tr>
<tr>
<td>MONTH</td>
<td>Month</td>
<td>17860320.00</td>
<td>699.00000000</td>
</tr>
</tbody>
</table>
6.4 Creating Summary Reports with the MEANS Procedure  6-37

MAXDEC= Option

The MAXDEC= option in the PROC MEANS statement specifies the maximum number of decimal places to display the statistics in the output.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Label</th>
<th>N</th>
<th>Mean</th>
<th>Std Dev</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTUAL</td>
<td>Actual Sales</td>
<td>1440</td>
<td>507.178472</td>
<td>287.0313</td>
<td>3.00</td>
<td>1000.00</td>
</tr>
<tr>
<td>PREDICT</td>
<td>Predicted Sales</td>
<td>1440</td>
<td>490.48264</td>
<td>285.767</td>
<td>0.00</td>
<td>1000.00</td>
</tr>
<tr>
<td>QUARTER</td>
<td>Quarter</td>
<td>1440</td>
<td>2.5000</td>
<td>1.11842</td>
<td>1.00</td>
<td>4.00</td>
</tr>
<tr>
<td>YEAR</td>
<td>Year</td>
<td>1440</td>
<td>1993.50</td>
<td>0.50</td>
<td>1993.00</td>
<td>1994.00</td>
</tr>
<tr>
<td>MONTH</td>
<td>Month</td>
<td>1440</td>
<td>12403.00</td>
<td>210.63</td>
<td>12054.00</td>
<td>12753.00</td>
</tr>
</tbody>
</table>

Without MAXDEC=

VAR Statement

The VAR statement identifies the analysis variables and specifies their order in the results.

```
proc means data=sashelp.prdsale maxdec=2 n mean;
  var predict actual;
run;
```

The MEANS procedure analyzes all numeric variables if you omit the VAR statement.
CLASS Statement

The CLASS statement specifies one or more variables that the procedure uses to group the data.

```sas
proc means data=sashelp.prdsale
maxdec=2 n mean;
var predict actual;
class country year;
run;
```

CLASS Statement

Adding a CLASS statement adds the N Obs column.

<table>
<thead>
<tr>
<th>Country</th>
<th>Year</th>
<th>N Obs</th>
<th>Variable</th>
<th>Label</th>
<th>N</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANADA</td>
<td>1993</td>
<td>240</td>
<td>PREDICT</td>
<td>Predicted Sales</td>
<td>240</td>
<td>497.20</td>
</tr>
<tr>
<td></td>
<td>1994</td>
<td>240</td>
<td>ACTUAL</td>
<td>Actual Sales</td>
<td>240</td>
<td>504.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GERMANY</td>
<td>1993</td>
<td>240</td>
<td>PREDICT</td>
<td>Predicted Sales</td>
<td>240</td>
<td>488.00</td>
</tr>
<tr>
<td></td>
<td>1994</td>
<td>240</td>
<td>ACTUAL</td>
<td>Actual Sales</td>
<td>240</td>
<td>530.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U.S.A.</td>
<td>1993</td>
<td>240</td>
<td>PREDICT</td>
<td>Predicted Sales</td>
<td>240</td>
<td>491.50</td>
</tr>
<tr>
<td></td>
<td>1994</td>
<td>240</td>
<td>ACTUAL</td>
<td>Actual Sales</td>
<td>240</td>
<td>484.57</td>
</tr>
</tbody>
</table>

- Number of observations for each unique combination of the class variables
- Number of nonmissing values for the analysis variable
CLASS Statement

For a given data set, there are 20 observations with a Country value of GERMANY. Of those 20 observations, only 15 observations have a value for PREDICT.

Which output is correct?

A.  

<table>
<thead>
<tr>
<th>Country</th>
<th>Obs</th>
<th>Variable</th>
<th>Label</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>GERMANY</td>
<td>15</td>
<td>PREDICT</td>
<td>Predicted Sales</td>
<td>20</td>
</tr>
</tbody>
</table>

B.  

<table>
<thead>
<tr>
<th>Country</th>
<th>Obs</th>
<th>Variable</th>
<th>Label</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>GERMANY</td>
<td>20</td>
<td>PREDICT</td>
<td>Predicted Sales</td>
<td>15</td>
</tr>
</tbody>
</table>

Refer to Exercise 5 for Chapter 6 in Appendix A.

UNIVARIATE Procedure

Another procedure for creating summary reports is the UNIVARIATE procedure.

```sas
proc univariate data=sashelp.air;
  var air;
run;
```

The UNIVARIATE procedure can display the following sections of output:
- Moments
- Basic Statistical Measures
- Tests for Locations
- Quantiles
- Extreme Observations
- Missing Values
UNIVARIATE Procedure

By default, the *Extreme Observations* section includes the five lowest and five highest values for the analysis variable and the corresponding observation numbers.

```sas
proc univariate data=sashelp.air nextrobs=2;
 var air;
 id date;
run;
```

### Extreme Observations

<table>
<thead>
<tr>
<th>Value</th>
<th>DATE</th>
<th>Obs</th>
<th>Value</th>
<th>DATE</th>
<th>Obs</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>NOV49</td>
<td>11</td>
<td>606</td>
<td>AUG60</td>
<td>140</td>
</tr>
<tr>
<td>112</td>
<td>JAN49</td>
<td>1</td>
<td>622</td>
<td>JUL60</td>
<td>139</td>
</tr>
</tbody>
</table>

6.5 Directing Reports to External Files with ODS

Output Delivery System (ODS)

ODS statements enable you to create output in a variety of forms.
Output Delivery System

Which ODS destination is used to direct reports to the Output window?

A. LISTING
B. PRINT
C. OUTPUT
D. WINDOW

ODS HTML Statement

The ODS HTML statement opens or closes the HTML destination, which produces HyperText Markup Language files that are viewable with a web browser.

```ods html file='gnp.html' style=default;
proc print data=sashelp.gnp;
  ... run;
proc means data=sashelp.gnp;
  ... run;
ods html close;```
What is wrong with the following program?

```
ods html file='rent.pdf';
proc print data=sashelp.rent;
  title 'Rent Report';
ods close;
run;
```
**ODS RTF and PDF Statements**

- The **ODS RTF statement** opens or closes the RTF destination, which produces Rich Text Format files that are viewable with a word processor.
- The **ODS PDF statement** opens or closes the PDF destination, which produces Portable Document Format files that are viewable with an Adobe product.

```ods rtf file='shoes.rtf';
ods pdf file='shoes.pdf';
proc freq data=sashelp.shoes;
... run;
ods rtf close;
ods pdf close;```
Multiple ODS Destinations

Which statement can be used to close multiple ODS destinations?

A. `ods close;`
B. `ods all close;`
C. `ods _all_ close;`
D. `ods _destination_ close;`

The ODS _ALL_ CLOSE statement closes all open destinations including the LISTING destination.

```sas
ods listing;
ods html file='shoes.html';
ods rtf file='shoes.rtf';
ods pdf file='shoes.pdf';
proc print data=sashelp.shoes;
... run;
ods _all_ close;
ods Listing;
```
Destinations Used with Excel

Which destination cannot create a file that can be opened in Excel?

A. CSVALL
B. EXCEL
C. EXCELP
D. MSOFFICE2K

Destinations Used with Excel

- The CSVALL destination creates a CSV (comma-separated value) file.
- The EXCELP destination creates an XML (Extensible Markup Language) file.
- The MSOFFICE2K destination creates an HTML (HyperText Markup Language) file.

ods csvall file='shoes.csv';
ods tagsets.excelxp file='shoes.xml';
ods msoffice2k file='shoes.html';
proc print data=sashelp.shoes;
   ...
run;
ods _all_ close;

Refer to Exercise 6 for Chapter 6 in Appendix A.
# 6.6 Answers to Questions

<table>
<thead>
<tr>
<th>Question Slide Number</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2. and 3.</td>
</tr>
<tr>
<td>13</td>
<td>A.</td>
</tr>
<tr>
<td>16</td>
<td>A.</td>
</tr>
<tr>
<td>19</td>
<td>B.</td>
</tr>
<tr>
<td>26</td>
<td>C.</td>
</tr>
<tr>
<td>30</td>
<td>3. and 6.</td>
</tr>
<tr>
<td>36</td>
<td><code>format inventory dollar14.2 sales dollar13.2;</code></td>
</tr>
<tr>
<td>38</td>
<td>C.</td>
</tr>
</tbody>
</table>
| 41                    | - The MMDDYY8. format must be the MMDDYY10. format.  
- A period must be added to the end of the DATE9 format. |
| 47                    | B. |
| 51                    | C. |
| 62                    | `ERROR: The quoted string '0' is not acceptable to a numeric format or informat.` |
| 65                    | A. |
| 68                    | C. |
| 71                    | Keywords do not use quotation marks. |
| 75                    | - Character values are case sensitive.  
- Missing period, $testfmt. |
| 78                    | D. |
| 86                    | C. |
| 91                    | C. |
| 98                    | C. |
| 106                   | B. |
| 114                   | A. |

(Continued on the next page.)
<table>
<thead>
<tr>
<th>Question Slide Number</th>
<th>Answer</th>
</tr>
</thead>
</table>
| 118                   | • The extension of the file must be appropriate to the destination.  
                        | • HTML is missing in the statement to close the file.  
                        | • The RUN statement must be before the statement that closes the file. |
| 122                   | C.                                                      |
| 125                   | B.                                                      |
Chapter 7  Additional Information

7.1  More Specifics about the SAS Base Programming Exam ........................................7-3

7.2  Additional Preparation Resources ............................................................................7-6

7.3  Test-Taking Strategies .............................................................................................7-9
7.1 More Specifics about the SAS Base Programming Exam

SAS Certification Website

Remember to check the SAS certification website prior to taking your exam for up-to-date information.

http://support.sas.com/certify

SAS Base Programming for SAS®

After completing the exam, you receive your score.

PASSING SCORE: % YOUR SCORE: 100% GRADE: Pass

<table>
<thead>
<tr>
<th>Section Analysis</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessing Data</td>
<td>100%</td>
</tr>
<tr>
<td>Creating Data Structures</td>
<td>100%</td>
</tr>
<tr>
<td>Managing Data</td>
<td>100%</td>
</tr>
<tr>
<td>Generating Reports</td>
<td>100%</td>
</tr>
<tr>
<td>Handling Errors</td>
<td>100%</td>
</tr>
</tbody>
</table>

The score report will display the percentage of items in each section that you answered correctly for your exam. These section scores are calculated on a per section basis and cannot be used in determining your total score.
GRADE: Pass

If you pass your exam and meet all requirements for this credential, you will receive an e-mail from SAS with instructions providing access to your certificate and logo.

These e-mails are sent to the e-mail address provided at exam registration.

Please allow at least one week from your exam date to receive your e-mail.

SAS Foundation Credentials and Exams

After you pass the SAS Base Programming Exam, consider taking other certification exams to earn additional SAS Foundation credentials.

<table>
<thead>
<tr>
<th>Credentials</th>
<th>Certification Exams</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ SAS Certified Base Programmer for SAS®9</td>
<td>➢ SAS Base Programming for SAS®9</td>
</tr>
<tr>
<td>SAS Certified Advanced Programmer for SAS®9</td>
<td>➢ SAS Base Programming for SAS®9</td>
</tr>
<tr>
<td></td>
<td>➢ SAS Advanced Programming for SAS®9</td>
</tr>
<tr>
<td>SAS Certified Clinical Trials Programmer Using SAS®9</td>
<td>➢ Clinical Trials Programming Using SAS®9 OR</td>
</tr>
<tr>
<td></td>
<td>➢ SAS Base Programming for SAS®9</td>
</tr>
<tr>
<td></td>
<td>➢ Clinical Trials Programming Using SAS®9 – Accelerated Version</td>
</tr>
</tbody>
</table>
Recertification

- Recertification is not required for individuals holding versioned certification credentials.
- SAS Certified Base Programmer for SAS®9 is a versioned certification credential.

Retaking the Exam

- A candidate can retake an exam five times in a 12-month period.
- A candidate must wait a minimum of 14 calendar days between attempts.
- Exams that do not comply with the retake examination policy will be considered invalid.
- Exam retakes require payment of the full exam fee and are not discounted.
7.2 Additional Preparation Resources

Additional Preparation Resources

Taking this review course is not a guarantee that you will pass the exam.

This review course provides practice in multiple-choice and short-answer questions, familiarizes you with SAS terminology, and refreshes your mind on topics learned but not used.

In addition, this course helps you to determine in what areas you need additional preparation and experience.

Exam Preparation Options

The following are some of the ways a candidate can prepare for the SAS Base Programming Exam:

- Instructor-led Training
- Self-paced e-learning
- Certification Prep Guide
- Online Documentation
Instructor-led Training

Self-paced e-Learning

http://support.sas.com/training

http://support.sas.com/training/elearn
Online Documentation

Recommended documentation:
- SAS® 9.3 Language Reference: Concepts
- Step-by-Step Programming with Base SAS® Software
- Base SAS® 9.3 Procedures Guide
- Base SAS® 9.3 Procedures Guide: Statistical Procedures
- SAS® 9.3 Data Set Options: Reference
- SAS® 9.3 Formats and Informats: Reference
- SAS® 9.3 Functions and CALL Routines: Reference
- SAS® 9.3 Statements: Reference
- SAS® 9.3 System Options: Reference, Second Edition
- SAS/ACCESS® 9.3 Interface to PC Files: Reference

7.3 Test-Taking Strategies

Before the Exam
- Extend studying and reviewing sessions over days or weeks.
- Do not cram the night before.
- Practice answering multiple choice SAS questions.
- Practice exam questions under timed conditions.
- Determine a plan for how you will use the allotted exam time.
- Get a full night's sleep before the exam.
- Arrive early to the exam and take a moment to relax.
- Listen attentively to the instructions given by the staff.
- Read the exam directions carefully.
During the Exam

- Maintain a positive attitude.
- Read each question carefully and thoroughly.
- Formulate your answer before reading the options.
- Eliminate unlikely options first.
- Be sure to read all options before selecting one.
- Pace yourself so that you have enough time to answer every question.
- Leave no questions unanswered.

During the Exam

- Rely on your first impression.
- Do not be afraid to change an answer if you feel strongly about it.
- Do not be discouraged if you cannot answer a question.
- Skip questions that you cannot answer, and return to those questions after completing the remainder of the exam.
- Plan to finish early and have time for review.
- Return to difficult questions that you marked for review.
Appendix A Exercises and Solutions

A.1 Exercises ........................................................................................................................ A-3

  Chapter 1.......................................................................................................................... A-3
  Chapter 2.......................................................................................................................... A-4
  Chapter 3.......................................................................................................................... A-7
  Chapter 4.......................................................................................................................... A-11
  Chapter 5.......................................................................................................................... A-14
  Chapter 6.......................................................................................................................... A-18

A.2 Solutions ...................................................................................................................... A-24

  Chapter 1.......................................................................................................................... A-24
  Chapter 2.......................................................................................................................... A-25
  Chapter 3.......................................................................................................................... A-26
  Chapter 4.......................................................................................................................... A-27
  Chapter 5.......................................................................................................................... A-29
  Chapter 6.......................................................................................................................... A-31
A.1 Exercises

Chapter 1

1. Fundamental Concepts

Answer TRUE or FALSE to the following sentences.

a. ___________ The two types of steps that can make up a SAS program are DATA and PROC.

b. ___________ A DATA step must use a SAS data set as input.

c. ___________ A statement always ends in a colon.

d. ___________ A global statement stays in effect for only the subsequent step.

e. ___________ The LIBNAME statement assigns a logical name to a SAS data library.

f. ___________ Data sets are referenced using a four-level name.

g. ___________ Data sets located in the Sasuser data library are considered temporary.

h. ___________ A variable name and the name of a data set can be up to 32 characters long.

i. ___________ By default, a variable name can contain special characters such as a dash (−).

j. ___________ A numeric variable is stored as 32 bytes by default.

k. ___________ A numeric variable can be stored with digits, decimal point, comma, minus sign, and E for scientific notation.

l. ___________ A character variable is stored as 1 to 32,767 bytes.

m. ___________ A SAS date value represents the number of days between January 1, 1960, and a specific date.

n. ___________ A missing numeric value is represented with a zero.

o. ___________ A missing character value is represented with a blank.

p. ___________ The DESCRIPTOR procedure views the descriptor portion of a SAS data set.

q. ___________ A statement that starts with an asterisk is a SAS comment.

r. ___________ The SAS log contains messages starting with the words NOTE, SUGGESTION, and ERROR.
Chapter 2

1. Input and Output Data Sets

The SAS data set company.sales has three variables (product, price, and quantity). A new data set work.sales must be created. The new data set needs to contain two variables (product and total). The variable total is the result of price multiplied by quantity.

Complete the following program based on the previous scenario:

```
data ________________;
set ________________;
keep ________________;
total = ________________;
run;
```

2. Multiple Data Sets

The SAS data set sashelp.class has five variables (name, sex, age, height, and weight) and 19 observations (9 observations with sex='F' and 10 observations with sex='M').

Answer the questions based on the previous information and the following program:

```
data work.female(drop=height)
   work.everyone(keep=name weight height);
set sashelp.class;
if sex='F' then output work.female;
output work.everyone;
run;
```

a. What is the input data set? __________________________

b. How many output data sets are being created? __________________________

c. How many observations are in work.female? __________________________

d. How many observations are in work.everyone? __________________________

e. What variables are in work.female? __________________________
3. WHERE and Subsetting IF Statements

Below is a partial view of the work.sales data set:

<table>
<thead>
<tr>
<th>DATE</th>
<th>STATE</th>
<th>PRODUCT</th>
<th>actual</th>
<th>predict</th>
</tr>
</thead>
<tbody>
<tr>
<td>575</td>
<td>14184</td>
<td>DESK</td>
<td>1852</td>
<td>2043</td>
</tr>
<tr>
<td>576</td>
<td>14214</td>
<td>DESK</td>
<td>1211</td>
<td>2146</td>
</tr>
<tr>
<td>577</td>
<td>13515</td>
<td>SOFA</td>
<td>1151</td>
<td>465</td>
</tr>
<tr>
<td>578</td>
<td>13546</td>
<td>SOFA</td>
<td>1630</td>
<td>103</td>
</tr>
</tbody>
</table>

A new data set, work subset, must be created. The new data set should contain only observations with a state equal to Texas, a date less than January 1, 1998, and a difference greater than 1000.

Complete the following program as efficiently as possible based on the previous scenario:

```sas
data subset;
  set sales;
  where ____________________________;
  difference=actual-predict;
  if ____________________________;
run;
```

4. SORT Procedure

Answer the questions based on the following program:

```sas
proc sort data=sashelp.shoes
  out=shoes;
  by descending region product;
run;
```

a. What is the input data set? ____________________________
b. What is the output data set? ____________________________
c. Where is the output data set stored? ____________________________
d. How many variables are used to sort the data set? ____________________________
e. Does the DESCENDING option apply to the region variable? ____________________________
f. Does the DESCENDING option apply to the product variable? ____________________________
g. Which variable is considered the primary sort variable? ____________________________
h. What other statements can be added to the SORT procedure? ____________________________
i. Does the SORT procedure create a report? ____________________________
5. DATA Step Merge

Below is the input data set `work.employees`:

<table>
<thead>
<tr>
<th>Viewtable: Work.employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Below is the input data set `work.salaries`:

<table>
<thead>
<tr>
<th>Viewtable: Work.Salaries</th>
</tr>
</thead>
<tbody>
<tr>
<td>idnum</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
</tbody>
</table>

Below is the output data set `work.empsal`:

<table>
<thead>
<tr>
<th>Viewtable: Work.Empsal</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Add the appropriate statement to the following program to create the output data set based on the input data sets:

```sas
data empsal;
  by idnum;
  if emp=1 and sal=1;
run;
```
1. **Program Data Vector (PDV)**

```
input state $ 1-2
   @5 date mmddyy10. @18 populat comma15.
  @35 city1 $ city2 $;
```

Given the previous INPUT statement, create the PDV that is created at compile time. Include the variable name, the variable type (char or num), and the variable byte size.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Column, Formatted, and List Input

Below is the raw data file **kids7.dat**:

```
----|----10---|----20---|----30---|----41---|
Tina Smith 15$1,322!Tim!Tammy
Michael McMurray 12$532!Mary
Jackie Jones 14$1,000!Jim!John!Joyce
```

Below is the desired output data set, **work.kids7**:

<table>
<thead>
<tr>
<th>VIEWTABLE: Work.Kids7</th>
</tr>
</thead>
<tbody>
<tr>
<td>first  last  age  savings  sibling1  sibling2  sibling3</td>
</tr>
<tr>
<td>1  Tina  Smith  15  1322  Tim  Tammy</td>
</tr>
<tr>
<td>2  Michael  McMurray  12  532  Mary</td>
</tr>
<tr>
<td>3  Jackie  Jones  14  1000  Jim  John  Joyce</td>
</tr>
</tbody>
</table>

Complete the following program based on the raw data file **kids7.dat** and the desired output data set **work.kids7**. Use column input for **first**, **last**, and **age**. Use list input for **savings**, **sibling1**, **sibling2**, and **sibling3**.

```
data work.kids7;
  infile 'kids7.dat'
     input
       first  last  age  savings  sibling1  sibling2  sibling3
     run;
```
3. Definitions Applying to Raw Data Files

Place the appropriate letter before each item.

_____ DLM= _____ : MODIFIER
_____ DSD _____ /
_____ MISSOVER _____ #N
_____ INFORMAT

a. A line-pointer control that advances the pointer to column 1 of the next input record

b. An option that prevents an INPUT statement from reading a new input data record if it does not find values in the current input line for all the variables in the statement

c. An option that specifies a delimiter to be used for LIST input

d. Applies an informat to the field and ignores the width

e. An option that treats two consecutive delimiters as a missing value

f. A line-pointer control that advances the pointer to column 1 of record N

g. An instruction that SAS uses to read data values into a variable
4. Reading Excel Files

The Excel workbook named **products.xls** contains four worksheets. Each sheet contains two columns: **category** and **name**.

Find the five mistakes in the following program:

```plaintext
libname prod 'products';

proc contents data=prod.all;
run;

data work.golf;
  set prod.'sports$';
  where category='Golf';
run;
libname clear;
proc print data=work.golf;
run;
```
Chapter 4

1. Assignment Statements

Answer TRUE or FALSE to the following comments based on the given assignment statement:

\[
\text{total} = \text{num1} + \text{num2} + \text{num3};
\]

- The variable `total` is a numeric variable.
- The values `num1`, `num2`, and `num3` are constants.
- If `num2` is missing, then `total` will be missing.

\[
\text{fullname} = \text{'Ms. or Mr.' || name};
\]

- The variable `fullname` is a numeric variable.
- The value 'Ms. or Mr.' is a constant.
- The value `||` is an operator.

\[
\text{birthdate} = \text{'12FEB1992'd};
\]

- The variable `birthdate` is a character variable.
- The byte size of `birthdate` is 9 bytes.
- The value '12FEB1992'd is an operator.

\[
\text{phonenumber} = \text{'888-999-0000'};
\]

- The variable `phonenumber` is a character variable.
- The byte size of `phonenumber` is 8 bytes.
- The value '888-999-0000' is an operator.
2. IF-THEN DO / ELSE DO Statements

Answer the questions based on the following program:

```sas
data newprice;
  infile 'raw-data-file';
  input mfg $ type $ price;
  length saletype $ 18;
  if mfg='Crew' then do;
    pct=0.75;
    saleprice = price * pct;
    saletype = '25% off';
  end;
  else if mfg='Hi-fly' then do;
    pct=0.70;
    saleprice = price * pct;
    saletype = '30% off';
  end;
  else do;
    pct=0.90;
    saleprice = price * pct;
    saletype = '10% Storewide Sale';
  end;
  format price saleprice dollar8.2;
run;
```

a. How many DO blocks are in the program? 3

b. How many variables will be assigned values if an expression is true? 3

c. How many of those variables are numeric? 2

d. What is the byte size of `pct`? 8

e. What would be the byte size of `saletype` if the LENGTH statement were not part of the program? 18

f. How many ELSE statements are in the program? 2

g. Why is the word ELSE used? To indicate an alternative action.

h. Why are the DO blocks needed? To perform actions under different conditions.

i. What stops each DO block? The end statement.

j. Will the value of `pct` ever be missing in the data set? No.
3. Multiple BY-Group Variables

Given the following statement:

```plaintext
by state city;
```

Fill in the following table with the correct **Total Donation** for each BY group and the **FIRST.** and **LAST.** values:

<table>
<thead>
<tr>
<th>State</th>
<th>City</th>
<th>Donation</th>
<th>Total Donation</th>
<th>first. State</th>
<th>last. State</th>
<th>first. City</th>
<th>last. City</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC</td>
<td>Charlotte</td>
<td>2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NC</td>
<td>Charlotte</td>
<td>9000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NC</td>
<td>Charlotte</td>
<td>4000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NC</td>
<td>Greenville</td>
<td>6000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NC</td>
<td>Greenville</td>
<td>3000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>Greenville</td>
<td>5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>Greenville</td>
<td>2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC</td>
<td>Pelzer</td>
<td>5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The variable **Total Donation** represents a running total within each BY group.
Chapter 5

1. Character Functions

Place the appropriate letter before each function.

_____ CATS _____ PROPCASE
_____ CATX _____ RIGHT
_____ COMPBL _____ SCAN
_____ COMPRESS _____ STRIP
_____ FIND _____ SUBSTR
_____ LEFT _____ TRANWRD
_____ LENGTH _____ TRIM
_____ LOWCASE _____ UPCASE

a. Concatenates character strings and removes leading and trailing blanks

b. Selects a given word from a character expression

c. Replaces or removes all occurrences of a word in a character string

d. Converts all letters in an argument to lowercase

e. Right-aligns a character expression

f. Removes multiple blanks from a character string

g. Searches a character expression for a string of characters with the capability of ignoring case and trimming trailing blanks

h. Extracts a substring from an argument

i. Converts all letters in an argument to uppercase

j. Returns a character string with specified characters removed from the original string

k. Removes trailing blanks from character expressions

l. Returns an integer that represents the position of the rightmost non-blank character in a string

m. Left-aligns a character expression

n. Concatenates character strings, removes leading and trailing blanks, and inserts separators

o. Converts all words in an argument to proper case

p. Removes leading and trailing blanks from character expressions
2. FIND Function

The following program is submitted:

```sas
data tonguetwister;
  length string $ 37;
  string='How much WOOD would a woodchuck chuck';
  num1=find(string,'wood');
  num2=find(string,'wood','i');
  num3=find(string,'wood ','t');
  num4=find(string,'wood ','i',15);
  num5=find(string,'WOOD ','it',15);
  num6=find(string,'WOOD ','it',-15);
  num7=find(string,'wood',40);
  num8=find(string,'WOOD','i',-40);
run;
```

Fill in the following table with the correct value of `num1` through `num8`:

<table>
<thead>
<tr>
<th>num1</th>
<th>num2</th>
<th>num3</th>
<th>num4</th>
<th>num5</th>
<th>num6</th>
<th>num7</th>
<th>num8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Truncation Functions

Fill in the following table based on the value of `num` and the truncation function:

<table>
<thead>
<tr>
<th>num</th>
<th>ceil(num)</th>
<th>floor(num)</th>
<th>int(num)</th>
<th>round(num)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-2.75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23.1234</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-23.1234</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## 4. Automatic Data Conversions

Fill in the following tables with the converted value assuming automatic conversion:

<table>
<thead>
<tr>
<th>Value of Character Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>162400</td>
</tr>
<tr>
<td>$162,400</td>
</tr>
<tr>
<td>49275.937</td>
</tr>
<tr>
<td>+24</td>
</tr>
<tr>
<td>-73.5</td>
</tr>
<tr>
<td>01234</td>
</tr>
<tr>
<td>52E3</td>
</tr>
<tr>
<td>01/01/1960</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value of Numeric Variable (8 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>162400</td>
</tr>
<tr>
<td>49275.937</td>
</tr>
<tr>
<td>-73.5</td>
</tr>
<tr>
<td>52E3</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>

Automatic character-to-numeric conversion using the `W.` informat

Automatic numeric-to-character conversion using the `BEST12.` format
5. DO Loops and Arrays

The following is the input data set, Weekly:

<table>
<thead>
<tr>
<th>name</th>
<th>week1</th>
<th>week2</th>
<th>week3</th>
<th>week4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jack</td>
<td>25</td>
<td>32</td>
<td>48</td>
</tr>
<tr>
<td>2</td>
<td>Susan</td>
<td>10</td>
<td>12</td>
<td>10</td>
</tr>
</tbody>
</table>

The following is the desired output data set, WeeklyRotate:

<table>
<thead>
<tr>
<th>name</th>
<th>week</th>
<th>miles</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jack</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>Jack</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>Jack</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>Jack</td>
<td>33</td>
</tr>
<tr>
<td>5</td>
<td>Susan</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Susan</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>Susan</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>Susan</td>
<td>10</td>
</tr>
</tbody>
</table>

Complete the following program to create the desired output data set:

```sas
data WeeklyRotate;
  set Weekly;
  array run [5];
  do ;
    miles = ;
    output;
  ;
  drop week1 - week4;
run;
```
Chapter 6

1. PRINT Procedure

Place the appropriate letter before each item.

_____ BY Statement     _____ PAGEBY Statement
_____ FOOTNOTE Statement  _____ SUM Statement
_____ FORMAT Statement   _____ TITLE Statement
_____ LABEL Statement     _____ VAR Statement
_____ NOOBS Option       _____ WHERE Statement
_____ OPTIONS Statement

a. Puts each separate section of a BY group on separate pages
b. Changes the value of one or more SAS system options
c. Suppresses the column in the output that identifies each observation by number
d. Specifies up to 10 lines of text at the top of output
e. Subsets the input data set by specifying certain conditions that each observation must meet
f. Assigns descriptive labels to variable names
g. Produces a separate section of the report for each BY group
h. Selects variables that appear in the report and determines the variables order
i. Specifies up to 10 lines of text at the bottom of output
j. Associates formats to variable values
k. Totals values of numeric variables
2. PRINT Procedure

The following is the desired report (partial output):

<table>
<thead>
<tr>
<th>Country</th>
<th>Year</th>
<th>Predicted Sales</th>
<th>Actual Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mexico</td>
<td>1995</td>
<td>$282</td>
<td>$189</td>
</tr>
<tr>
<td>Mexico</td>
<td>1995</td>
<td>$276</td>
<td>$266</td>
</tr>
<tr>
<td>Mexico</td>
<td>1995</td>
<td>$807</td>
<td>$241</td>
</tr>
<tr>
<td>Mexico</td>
<td>1995</td>
<td>$782</td>
<td>$451</td>
</tr>
<tr>
<td>Mexico</td>
<td>1995</td>
<td>$381</td>
<td>$126</td>
</tr>
</tbody>
</table>

Find the six syntax mistakes in the following program:

```sas
options nodate numberno ps=30 ls=64;
proc print data=sashelp.prdsal2 nobs headers;
  where country=Mexico;
  var country year predict actual;
  by country;
  sum predict actual;
  label predict='Predicted Sales'
   actual='Actual Sales';
  format predict actual dollar12;
  title 'Predicted versus Actual Sales'
run;
```
3. FORMAT Procedure

Answer the questions based on the following program:

```sas
proc format;
  value $gender 'F'  = 'Female'
                 'M'  = 'Male'
                 other = 'Miscoded';
  value wtrange low - <100 = 'Under 100 lbs'
                 100 - high = '100+ lbs';
run;
proc print data=sashelp.class;
  var name sex weight;
  format sex $gender. weight wtrange.;
run;
```

a. How many formats are created in the PROC FORMAT step? _______________________

b. Is $GENDER a character or numeric format? _______________________

c. Is WTRANGE a character or numeric format? _______________________

d. What is the maximum length of a format name? _______________________

e. What must start a character format name? _______________________

f. What are the three keywords used in the VALUE statements? _______________________

g. Does the less than (<) sign include or exclude values from ranges? _______________________

h. Does the LOW keyword include a numeric missing value? _______________________

i. What special character is used at the end of the format name when you use a format? ________

j. Are these formats temporary or permanent? _______________________

k. Does the PROC FORMAT step refer to the data set and the variable that will use the format? _______________________

4. FREQ Procedure

The following is the desired report:

```
Report One

The FREQ Procedure

Number of Variable Levels

<table>
<thead>
<tr>
<th>Variable</th>
<th>Label</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product_category</td>
<td>Product Category</td>
<td>12</td>
</tr>
</tbody>
</table>

Product Category

<table>
<thead>
<tr>
<th>Product Category</th>
<th>Frequency</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assorted Sports Articles</td>
<td>64</td>
<td>7.02</td>
</tr>
<tr>
<td>Children Sports</td>
<td>176</td>
<td>19.30</td>
</tr>
<tr>
<td>Clothes</td>
<td>240</td>
<td>26.32</td>
</tr>
<tr>
<td>Golf</td>
<td>32</td>
<td>3.51</td>
</tr>
<tr>
<td>Indoor Sports</td>
<td>48</td>
<td>5.26</td>
</tr>
<tr>
<td>Outdoors</td>
<td>112</td>
<td>12.28</td>
</tr>
<tr>
<td>Racket Sports</td>
<td>48</td>
<td>5.26</td>
</tr>
<tr>
<td>Running - Jogging</td>
<td>32</td>
<td>3.51</td>
</tr>
<tr>
<td>Shoes</td>
<td>48</td>
<td>5.26</td>
</tr>
<tr>
<td>Swim Sports</td>
<td>16</td>
<td>1.75</td>
</tr>
<tr>
<td>Team Sports</td>
<td>64</td>
<td>7.02</td>
</tr>
<tr>
<td>Winter Sports</td>
<td>32</td>
<td>3.51</td>
</tr>
</tbody>
</table>
```

Complete the following program to create the desired report:

```
options nodate nonumber ps=50 ls=74;
proc freq data=sashelp.orsales;
   title 'Report One';
run;
```
5. MEANS Procedure

The following is the desired report:

<table>
<thead>
<tr>
<th>State/Province</th>
<th>N Obs</th>
<th>Sum</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illinois</td>
<td>5760</td>
<td>410977</td>
<td>71</td>
</tr>
<tr>
<td>New York</td>
<td>1152</td>
<td>1705493</td>
<td>1445</td>
</tr>
</tbody>
</table>

Complete the following program to create the desired report:

```sas
options nodate nonumber ps=18 ls=74;
proc means data=sashelp.prdsal2;
  where state in ('New York','Illinois');
  label actual='Actual Retail Sales';
  title 'Product Sales Report';
  footnote 'Confidential';
run;
```
6. Output Delivery System

Fill in the blanks in the following program to complete the program and answer the questions:

```sas
ods listing;
ods html _______________ = 'steel.html';
ods _______________ file = 'steel.xml';

proc sort data=sashelp.steel out=steel;
  by date;
run;

proc print data=steel;
run;

proc freq data=steel;
  tables date;
run;

ods _all_ _______________;
ods listing;
```

a. How many destinations are open in the program? ________________________________

b. How many reports are sent to the open destinations? _____________________________
A.2 Solutions

Chapter 1

1. Fundamental Concepts

   a. **TRUE** The two types of steps that can make up a SAS program are DATA and PROC.

   b. **FALSE** A DATA step must use a SAS data set as input.

   c. **FALSE** A statement always ends in a colon.

   d. **FALSE** A global statement stays in effect for only the subsequent step.

   e. **TRUE** The LIBNAME statement assigns a logical name to a SAS data library.

   f. **FALSE** Data sets are referenced using a four-level name.

   g. **FALSE** Data sets located in the `Sasuser` data library are considered temporary.

   h. **TRUE** A variable name and the name of a data set can be up to 32 characters long.

   i. **FALSE** By default, a variable name can contain special characters such as a dash (−).

   j. **FALSE** A numeric variable is stored as 32 bytes by default.

   k. **FALSE** A numeric variable can be stored with digits, decimal point, comma, minus sign, and E for scientific notation.

   l. **TRUE** A character variable is stored as 1 to 32,767 bytes.

   m. **TRUE** A SAS date value represents the number of days between January 1, 1960 and a specific date.

   n. **FALSE** A missing numeric value is represented with a zero.

   o. **TRUE** A missing character value is represented with a blank.

   p. **FALSE** The DESCRIPTOR procedure views the descriptor portion of a SAS data set.

   q. **TRUE** A statement that starts with an asterisk is a SAS comment.

   r. **FALSE** The SAS log contains messages starting with the words NOTE, SUGGESTION, and ERROR.
Chapter 2

1. Input and Output Data Sets

```sas
data work.sales;
  set company.sales;
  keep product total;
  total = price*quantity;
run;
```

2. Multiple Data Sets
   a. What is the input data set? `sashelp.class`
   b. How many output data sets are being created? two
   c. How many observations are in `work.female`? nine
   d. How many observations are in `work.everyone`? 19
   e. What variables are in `work.female`? `name`, `sex`, `age`, and `weight`

3. WHERE and Subsetting IF Statements

```sas
data subset;
  set sales;
  where state='Texas' and date<'01JAN1998'd;
  difference=actual-predict;
  if difference>1000;
run;
```

4. SORT Procedure
   a. What is the input data set? `sashelp.shoes`
   b. What is the output data set? `shoes`
   c. Where is the output data set stored? work
   d. How many variables are used to sort the data set? two
   e. Does the DESCENDING option apply to the `region` variable? yes
   f. Does the DESCENDING option apply to the `product` variable? no
   g. Which variable is considered the primary sort variable? `region`
   h. What other statements can be added to the SORT procedure? FORMAT, LABEL, and WHERE
   i. Does the SORT procedure create a report? no

5. DATA Step Merge

```sas
data empsal;
  merge employees(in=emp rename=(id=idnum))
    salaries(in=sal);
  by idnum;
  if emp=1 and sal=1;
run;
```
Chapter 3

1. Program Data Vector (PDV)

<table>
<thead>
<tr>
<th>Name</th>
<th>state</th>
<th>date</th>
<th>populat</th>
<th>city1</th>
<th>city2</th>
<th><em>ERROR</em></th>
<th><em>N</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>char</td>
<td>num</td>
<td>num</td>
<td>char</td>
<td>char</td>
<td>num</td>
<td>num</td>
</tr>
<tr>
<td>Size</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

2. Column, Formatted, and List Input

```sas
data work.kids7;
  infile 'kids7.dat' dlm='!' missover;
  input first $ 1-9
    last $ 10-19
    age 20-21
    savings : comma8.
    sibling1 $
    sibling2 $
    sibling3 $;
run;
```

3. Definitions Applying to Raw Data Files

- DLM=
- DSD
- MISSOVER
- INFORMAT
- : MODIFIER

4. Reading Excel Files

```sas
libname prod 'products.xls';

proc contents data=prod._all_;
run;

data work.golf;
  set prod.'sports$'n;
  where category='Golf';
run;

libname prod clear;
proc print data=work.golf;
run;
```
Chapter 4

1. Assignment Statements

```
total = num1 + num2 + num3;
```

TRUE  The variable `total` is a numeric variable.

FALSE The values `num1`, `num2`, and `num3` are constants.

TRUE  If `num2` is missing, then `total` will be missing.

```
fullname = 'Ms. or Mr.' || name;
```

FALSE The variable `fullname` is a numeric variable.

TRUE  The value 'Ms. or Mr.' is a constant.

TRUE  The value `||` is an operator.

```
birthdate = '12FEB1992'd;
```

FALSE The variable `birthdate` is a character variable.

FALSE The byte size of `birthdate` is 9 bytes.

FALSE The value '12FEB1992'd is an operator.

```
phonenumber = '888-999-0000';
```

TRUE  The variable `phonenumber` is a character variable.

FALSE The byte size of `phonenumber` is 8 bytes.

FALSE The value '888-999-0000' is an operator.
2. IF-THEN DO / ELSE DO Statements
   a. How many DO blocks are in the program? three
   b. How many variables will be assigned values if an expression is true? three
   c. How many of those variables are numeric? two
   d. What is the byte size of \texttt{pct}? eight
   e. What would be the byte size of \texttt{saletype} if the LENGTH statement were not part of the program? seven
   f. How many ELSE statements are in the program? two
   g. Why is the word ELSE used? Subsequent ELSE statements are not evaluated after a true statement.
   h. Why are the DO blocks needed? executing three statements, not only one statement
   i. What stops each DO block? \texttt{end};
   j. Will the value of \texttt{pct} ever be missing in the data set? no

3. Multiple BY-Group Variables

<table>
<thead>
<tr>
<th>State</th>
<th>City</th>
<th>Donation</th>
<th>Total Donation</th>
<th>first. State</th>
<th>last. State</th>
<th>first. City</th>
<th>last. City</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC</td>
<td>Charlotte</td>
<td>2000</td>
<td>15000</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>NC</td>
<td>Charlotte</td>
<td>9000</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>NC</td>
<td>Charlotte</td>
<td>4000</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NC</td>
<td>Greenville</td>
<td>6000</td>
<td>9000</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>NC</td>
<td>Greenville</td>
<td>3000</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SC</td>
<td>Greenville</td>
<td>5000</td>
<td>7000</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SC</td>
<td>Greenville</td>
<td>2000</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SC</td>
<td>Pelzer</td>
<td>5000</td>
<td>5000</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Chapter 5

1. Character Functions
   a. CATS         o. PROPCASE
   n. CATX         e. RIGHT
   f. COMPBL       b. SCAN
   j. COMPRESS     p. STRIP
   g. FIND         h. SUBSTR
   m. LEFT         c. TRANWRD
   l. LENGTH       k. TRIM
   d. LOWCASE      i. UPCASE

2. FIND Function

<table>
<thead>
<tr>
<th>num1</th>
<th>num2</th>
<th>num3</th>
<th>num4</th>
<th>num5</th>
<th>num6</th>
<th>num7</th>
<th>num8</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>10</td>
<td>23</td>
<td>10</td>
<td>23</td>
<td>10</td>
<td>0</td>
<td>23</td>
</tr>
</tbody>
</table>

3. Truncation Functions

<table>
<thead>
<tr>
<th>num</th>
<th>ceil(num)</th>
<th>floor(num)</th>
<th>int(num)</th>
<th>round(num)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.75</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>-2.75</td>
<td>-2</td>
<td>-3</td>
<td>-2</td>
<td>-3</td>
</tr>
<tr>
<td>23.1234</td>
<td>24</td>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>-23.1234</td>
<td>-23</td>
<td>-24</td>
<td>-23</td>
<td>-23</td>
</tr>
</tbody>
</table>
4. Automatic Data Conversions

<table>
<thead>
<tr>
<th>Value of Character Variable</th>
<th>Value of Numeric Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>162400</td>
<td>162400</td>
</tr>
<tr>
<td>$162,400</td>
<td>.</td>
</tr>
<tr>
<td>49275.937</td>
<td>49275.937</td>
</tr>
<tr>
<td>+24</td>
<td>24</td>
</tr>
<tr>
<td>-73.5</td>
<td>-73.5</td>
</tr>
<tr>
<td>01234</td>
<td>1234</td>
</tr>
<tr>
<td>52E3</td>
<td>52000</td>
</tr>
<tr>
<td>01/01/1960</td>
<td>.</td>
</tr>
</tbody>
</table>

Automatic character-to-numeric conversion using the W. informat

<table>
<thead>
<tr>
<th>Value of Numeric Variable (8 bytes)</th>
<th>Value of Character Variable (12 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>162400</td>
<td>162400</td>
</tr>
<tr>
<td>49275.937</td>
<td>49275.937</td>
</tr>
<tr>
<td>-73.5</td>
<td>-73.5</td>
</tr>
<tr>
<td>52E3</td>
<td>52000</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Automatic numeric-to-character conversion using the BEST12. format

5. DO Loops and Arrays

```r
data WeeklyRotate;
  set Weekly;
  array run{4} week1 - week4;
  do week = 1 to 4;
    miles = run{week};
    output;
  end;
  drop week1 - week4;
run;
```
Chapter 6

1. PRINT Procedure

   g. BY Statement
   a. PAGEBY Statement
   i. FOOTNOTE Statement
   k. SUM Statement
   j. FORMAT Statement
   d. TITLE Statement
   f. LABEL Statement
   h. VAR Statement
   c. NOOBS Option
   e. WHERE Statement
   b. OPTIONS Statement

2. PRINT Procedure

   options nodate number ps=30 ls=64;
   proc print data=sashelp.prdsal2 noobs label;
     where country='Mexico';
     var country year predict actual;
     by country;
     sum predict actual;
     label predict='Predicted Sales'
                   actual='Actual Sales';
     format predict actual dollar12.;
     title 'Predicted versus Actual Sales';
   run;

3. FORMAT Procedure

   a. How many formats are created in the PROC FORMAT step? two ($GENDER and WTRANGE)
   b. Is $GENDER a character or a numeric format? character
   c. Is WTRANGE a character or a numeric format? numeric
   d. What is the maximum length of a format name? 32
   e. What must start a character format name? dollar sign ($)
   f. What are the three keywords used in the VALUE statements? OTHER, LOW, and HIGH
   g. Does the less than (<) sign include or exclude values from ranges? exclude
   h. Does the LOW keyword include a numeric missing value? no
   i. What special character is used at the end of the format name when you use a format? period (.)
   j. Are these formats temporary or permanent? temporary
   k. Does the PROC FORMAT step refer to the data set and the variable that will use the format? no
4. **FREQ Procedure**

```sas
options nodate nonumber ps=50 ls=74;
proc freq data=sashelp.orsales nlevels;
   tables product_category / nocum;
   title 'Report One';
run;
```

5. **MEANS Procedure**

```sas
options nodate nonumber ps=18 ls=74;
proc means data=sashelp.prdsal2 maxdec=0 sum median;
   where state in ('New York','Illinois');
   var actual;
   class state;
   label actual='Actual Retail Sales';
   title 'Product Sales Report';
   footnote 'Confidential';
run;
```

6. **Output Delivery System**

```sas
ods listing;
ods html file = 'steel.html';
ods tagsets.excelxp file = 'steel.xml';
ods _all_ close;
ods listing;

proc sort data=sashelp.steel out=steel;
   by date;
run;

proc print data=steel;
run;

proc freq data=steel;
   tables date;
run;

BODY= can be used in place of FILE=.

a. How many destinations are open in the program? three (LISTING, HTML, and EXCELXP)
b. How many reports are sent to the open destinations? two (PROC PRINT and PROC FREQ)
Appendix B  Practice Exam

B.1  SAS Fundamental Concepts......................................................................................... B-3

B.2  Working with SAS Data Sets......................................................................................... B-6

B.3  Working with Raw Data and Microsoft Excel Files ................................................... B-12

B.4  Creating Variables ....................................................................................................... B-20

B.5  Manipulating Data......................................................................................................... B-26

B.6  Generating Reports ...................................................................................................... B-31

B.7  Scores........................................................................................................................... B-39

B.8  Answers......................................................................................................................... B-40
B.1 SAS Fundamental Concepts

Answer the 5 questions in this section in 9 minutes or less. Do not use your course notes.

After completing the questions, compare your answers with the correct answers in Section 8 of this appendix.

Complete the following table after you determine the number of questions that you answered correctly:

<table>
<thead>
<tr>
<th>Section</th>
<th>Number of Questions</th>
<th>Numbers of Questions Answered Correctly</th>
<th>Percent Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS Fundamentals Concepts</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Percent Correct

- 5 correct out of 5 = 100%
- 4 correct out of 5 = 80%
- 3 correct out of 5 = 60%
- 2 correct out of 5 = 40%
- 1 correct out of 5 = 20%
1. Which of the following LIBNAME statements has the correct syntax?
   a. `libname monthly 'c:\monthly';`
   b. `libname 'c:\monthly' monthly;`
   c. `libname monthly='c:\monthly';`
   d. `libname 'c:\monthly'=monthly;`

2. Which of the following statements is true?
   a. A libref can be 32 or fewer characters.
   b. A variable name can be 32 or fewer characters.
   c. Numeric variables are stored as 32 bytes by default.
   d. Character variables are stored as 32 bytes by default.

3. A program was submitted and the SAS log is shown below.

   ```sas
   169  data work.sales;
   170    set sashelp.orsales;
   171    drop quarter year;
   172  run;
   NOTE: There were 912 observations read from the data set SASHELP.ORSALES.
   NOTE: The data set WORK.SALES has 912 observations and 6 variables.
   173  */
   174  proc contents data=work.sales;
   175  run;
   176  */
   177  proc print data=work.sales;
   178  run;
   NOTE: There were 912 observations read from the data set WORK.SALES.
   ```

Which of the following is true regarding the CONTENTS procedure?
   a. The PROC CONTENTS step failed execution.
   b. The PROC CONTENTS step did not execute.
   c. The second note applies to the PROC CONTENTS step.
   d. The last note applies to the PROC CONTENTS step and the PROC PRINT step.
4. The following program is submitted:

```sas
proc contents data=temp.sales;
run;
```

Which is the result?

a. a report showing the data portion of the temporary data set `temp.sales`
b. a report showing the data portion of the permanent data set `temp.sales`
c. a report showing the descriptor portion of the temporary data set `temp.sales`
d. a report showing the descriptor portion of the permanent data set `temp.sales`

5. Which of the following statements is true concerning SAS date values?

a. The SAS date value for 05JAN1960 is 4.
b. The SAS date value for 03/24/1952 is a positive number.
c. A SAS date value represents the number of days between January 1, 1950, and a specified date.
d. The SAS date value for September 16, 1999, can be written as the SAS date constant '09/16/1999'd.
B.2 Working with SAS Data Sets

Answer the 9 questions in this section in 15 minutes or less. Do not use your course notes.

After completing the questions, compare your answers with the correct answers in Section 8 of this appendix.

Complete the following table after you determine the number of questions that you answered correctly:

<table>
<thead>
<tr>
<th>Section</th>
<th>Number of Questions</th>
<th>Numbers of Questions Answered Correctly</th>
<th>Percent Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working with SAS Data Sets</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Percent Correct

9 correct out of 9 = 100%
8 correct out of 9 = 89%
7 correct out of 9 = 78%
6 correct out of 9 = 67%
5 correct out of 9 = 56%
4 correct out of 9 = 44%
3 correct out of 9 = 33%
2 correct out of 9 = 22%
1 correct out of 9 = 11%
1. The following program is submitted:

```sas
data work.newsales;
  set work.sales;
  sales=price*quantity;
  <insert statement here>
run;
```

Which SAS statement will output observations with **product** equal to the character value Shorts and **sales** less than one million?

a. `if product eq Shorts and sales<1000000;`

b. `if product='Shorts' and sales lt 1000000;`

c. `where product=Shorts and sales lt 1000000;`

d. `where product eq 'Shorts' and sales<1000000;`

2. Which of the following is true regarding the SORT procedure?

a. The SORT procedure requires the BY statement.

b. The SORT procedure has the ability to create a report or a new data set.

c. The SORT procedure can only sort values based on character variables.

d. The SORT procedure sorts values by descending order unless the ASCENDING option is specified.

3. Which items are potentially created at compile time of a DATA step?

a. input buffer, data values, and report

b. raw data file, program data vector, and report

c. raw data file, data values, and descriptor information

d. input buffer, program data vector, and descriptor information
4. Given the input SAS data set salary1:

<table>
<thead>
<tr>
<th>IDNUM</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>12649</td>
<td>52000</td>
</tr>
<tr>
<td>49255</td>
<td>75000</td>
</tr>
</tbody>
</table>

Given the input SAS data set salary2:

<table>
<thead>
<tr>
<th>ID</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>56391</td>
<td>89000</td>
</tr>
<tr>
<td>88376</td>
<td>66000</td>
</tr>
</tbody>
</table>

The following program is submitted:

```sas
data salaryall;
    <insert statement here>
run;
```

Given the desired output SAS data set salaryall:

<table>
<thead>
<tr>
<th>ID</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>12649</td>
<td>52000</td>
</tr>
<tr>
<td>49255</td>
<td>75000</td>
</tr>
<tr>
<td>56391</td>
<td>89000</td>
</tr>
<tr>
<td>88376</td>
<td>66000</td>
</tr>
</tbody>
</table>

Which statement will produce the desired output data set?

a. `set salary2 salary1(rename id=idnum);`
b. `set salary1 rename=(idnum=id) salary2;`
c. `set salary1(rename=(idnum=id)) salary2;`
d. `set salary1 salary2(rename=(id=idnum));`
5. The following program is submitted:

```sas
data work.firsthalf work.thirdqtr work.misic;
  set sashelp.retail;
  if 1<=month<=6 then output work.firsthalf;
  else if 7<=month<=9 then output work.thirdqtr;
run;
```

Which of the following statements is true regarding the previous program with an observation having `month` equal to 12?

a. The observation will be output to the `work.firsthalf` data set.
b. The observation will be output to the `work.thirdqtr` data set.
c. The observation will be output to the `work.misic` data set.
d. The observation will not be output to any data set.

6. The following SAS program is submitted:

```sas
proc sort data=sashelp.class new=sortdata;
  by name descending age;
run;
```

What is the result?

a. The program fails execution due to a syntax error with the NEW= option.
b. The program fails execution due to a syntax error with the DESCENDING option.
c. The program runs without errors and creates a new data set `work.sortdata` with the sorted observations.
d. The program runs with warnings and overwrites the original data set `sashelp.class` with the sorted observations.
7. Given the input data set `products`:

<table>
<thead>
<tr>
<th>CODE</th>
<th>PRODUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A123</td>
<td>Sandal</td>
</tr>
<tr>
<td>A234</td>
<td>Slipper</td>
</tr>
<tr>
<td>B345</td>
<td>Boot</td>
</tr>
<tr>
<td>B456</td>
<td>Sneaker</td>
</tr>
</tbody>
</table>

Given the input data set `costs`:

<table>
<thead>
<tr>
<th>CODE</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>A123</td>
<td>19.99</td>
</tr>
<tr>
<td>A234</td>
<td>9.99</td>
</tr>
<tr>
<td>B456</td>
<td>25.99</td>
</tr>
</tbody>
</table>

The following program is submitted:

```sas
data prodcost;
merge products(in=p) costs(in=c);
by code;
if p and c;
run;
```

Which is the result?

a. The program fails execution because of invalid IN= syntax.
b. The program fails execution because the subsetting IF statement is incomplete.
c. The program runs without errors or warnings and produces a data set with three observations and three variables.
d. The program runs without errors or warnings and produces a data set with four observations and three variables.

8. The following program is submitted:

```sas
data work.orsales;
set sashelp.orsales(firstobs=500 obs=700);
run;
```

How many observations are in the output data set `work.orsales`?

Enter your numeric answer.
9. The following program is submitted:

```sas
data work.sales;
  set sashelp.orsales;
  drop quarter year;
run;
```

Which of the following statements is true regarding the previous program?

a. The variables quarter and year will not be in work.sales.
b. The output data set will contain the variables _N_ and _ERROR_.
c. The data set work.sales is the input data set and sashelp.orsales is the output data set.
d. The output data set will have no observations because there is no OUTPUT statement.
B.3 Working with Raw Data and Microsoft Excel Files

Answer the 9 questions in this section in 15 minutes or less. Do not use your course notes.

After completing the questions, compare your answers with the correct answers in section 8 of this Appendix.

Complete the following table after you determine the number of questions that you answered correctly:

<table>
<thead>
<tr>
<th>Section</th>
<th>Number of Questions</th>
<th>Numbers of Questions Answered Correctly</th>
<th>Percent Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working with Raw Data and Excel Files</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Percent Correct

9 correct out of 9 = 100%
8 correct out of 9 = 89%
7 correct out of 9 = 78%
6 correct out of 9 = 67%
5 correct out of 9 = 56%
4 correct out of 9 = 44%
3 correct out of 9 = 33%
2 correct out of 9 = 22%
1 correct out of 9 = 11%
1. Given the raw data file `address.dat`:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue Smith</td>
<td>123 Main Street</td>
</tr>
<tr>
<td>San Diego</td>
<td>CA 92625</td>
</tr>
<tr>
<td>Julie Brown</td>
<td>456 Monroe Road</td>
</tr>
<tr>
<td>Birmingham</td>
<td>AL 35235</td>
</tr>
</tbody>
</table>

The following SAS program is submitted:

```sas
data work.address;
  infile 'address.dat';
  input first $ 1-9 last $ 10-14 / street $ 1-15 /;
run;
```

How many observations are in the output data set `work.address`?

Enter your numeric answer.
2. Given the raw data file 2005pop.dat:

|----|----10---|----20---|----30|
| 8,143,197 1 New York NY |
| 3,844,829 2 Los Angeles CA |
| 2,842,518 3 Chicago IL |

The following SAS program is submitted:

```sas
data work.population;
infile '2005pop.dat';
input @2 POPULATION comma9. @12 RANK 1.
@14 CITY $12.
@27 STATE $2;
run;
```

Which is the output data set work.population?

a.

<table>
<thead>
<tr>
<th>POPULATION</th>
<th>RANK</th>
<th>CITY</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>8,143,197</td>
<td>1</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>3,844,829</td>
<td>2</td>
<td>Los Angeles</td>
<td>CA</td>
</tr>
<tr>
<td>2,842,518</td>
<td>3</td>
<td>Chicago</td>
<td>IL</td>
</tr>
</tbody>
</table>

b.

<table>
<thead>
<tr>
<th>POPULATION</th>
<th>RANK</th>
<th>CITY</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>8,143,197</td>
<td>1</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>3,844,829</td>
<td>2</td>
<td>Los Angeles</td>
<td>CA</td>
</tr>
<tr>
<td>2,842,518</td>
<td>3</td>
<td>Chicago</td>
<td>IL</td>
</tr>
</tbody>
</table>

c.

<table>
<thead>
<tr>
<th>POPULATION</th>
<th>RANK</th>
<th>CITY</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>8143197</td>
<td>1</td>
<td>New York</td>
<td>8</td>
</tr>
<tr>
<td>3844829</td>
<td>2</td>
<td>Los Angeles</td>
<td>3</td>
</tr>
<tr>
<td>2842518</td>
<td>3</td>
<td>Chicago</td>
<td>2</td>
</tr>
</tbody>
</table>

d.

<table>
<thead>
<tr>
<th>POPULATION</th>
<th>RANK</th>
<th>CITY</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>8143197</td>
<td>1</td>
<td>New York</td>
<td>NY</td>
</tr>
<tr>
<td>3844829</td>
<td>2</td>
<td>Los Angeles</td>
<td>CA</td>
</tr>
<tr>
<td>2842518</td>
<td>3</td>
<td>Chicago</td>
<td>IL</td>
</tr>
</tbody>
</table>
3. The Excel workbook *customers.xls* contains a worksheet named **Females** and a worksheet named **Males**.

Which program will read the **Males** worksheet to create a SAS data set?

a.

```sas
libname customer excel 'customers.xls';

data work.males;
  set customer.males;
run;

libname customer clear;
```

b.

```sas
libname customer excel 'customers.xls';

data work.males;
  set customer.males.worksheet;
run;

libname customer clear;
```

c.

```sas
libname customer excel 'customers.xls';

data work.males;
  set customer.males$;
run;

libname customer clear;
```

d.

```sas
libname customer excel 'customers.xls';

data work.males;
  set customer.'males$'n;
run;

libname customer clear;
```
4. Which of the following is true regarding the DSD option when reading raw data files?
   a. The DSD option sets the delimiter to a blank.
   b. The DSD option treats two consecutive delimiters as a missing value.
   c. The DSD option belongs in the INPUT statement after a forward slash.
   d. The DSD option removes any delimiters located inside a set of quotation marks.

5. Given the raw data file **info.dat**:

<table>
<thead>
<tr>
<th>John Louisville KY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lorna Columbia MO 65203</td>
</tr>
</tbody>
</table>

   The following SAS program is submitted:

   ```sas
   data info;
   infile 'info.dat';
   input name $ city $ state $ zipcode $;
   run;
   ```

   Which is the result?
   a. The program runs without errors or warnings and produces a data set with two observations and four variables.
   b. The program produces a warning of invalid data for `zipcode` in line 1 and produces a data set with two observations and four variables.
   c. The program produces a note that SAS went to a new line when the INPUT statement reached past the end of a line and produces a data set with one observation and four variables.
   d. The program produces an error that SAS went to a new line when the INPUT statement reached past the end of a line and produces an error that SAS stopped processing the step because of errors.
6. Given the raw data file `revenue.dat`:

<table>
<thead>
<tr>
<th>Month</th>
<th>Revenue</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>$13,000</td>
<td>above</td>
</tr>
<tr>
<td>Feb</td>
<td>$900</td>
<td>below</td>
</tr>
<tr>
<td>Mar</td>
<td>$27,000</td>
<td>above</td>
</tr>
</tbody>
</table>

The following SAS program is submitted:

```sas
data work.target;
  infile 'revenue.dat';
  <insert statement here>
run;
```

Given the desired output data set `work.target`:

<table>
<thead>
<tr>
<th>Month</th>
<th>Revenue</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jan</td>
<td>13000</td>
<td>above</td>
</tr>
<tr>
<td>Feb</td>
<td>900</td>
<td>below</td>
</tr>
<tr>
<td>Mar</td>
<td>27000</td>
<td>above</td>
</tr>
</tbody>
</table>

Which statement will produce the desired output data set?

a. `input MONTH $ REVENUE TARGET $;`
b. `input MONTH $ REVENUE $ TARGET $;`
c. `input MONTH $ REVENUE: dollar8. TARGET $;`
d. `input MONTH $ REVENUE dollar7. TARGET $;`
7. A DATA step was submitted and a portion of the SAS log is shown below.

```sas
601  data work.population;
602    infile '2000pop.dat';
603    input population 1-7
604          rank 9
605          state 24-25
606          city $ 11-21;
607  run;

NOTE: Invalid data for state in line 1 24-25.
RULE:       ----+----1----+----2----+----3----+----4----+----5----
1         8008278 1 New York     NZ 25
population=8008278 rank=1 state=. city=New York _ERROR_=1 _N_=1
NOTE: Invalid data for state in line 2 24-25.
2         3694820 2 Los Angeles  CA 25
population=3694820 rank=2 state=. city=Los Angeles _ERROR_=1 _N_=2
NOTE: Invalid data for state in line 3 24-25.
3         2896016 3 Chicago      IL 25
population=2896016 rank=3 state=. city=Chicago _ERROR_=1 _N_=3
```

What is the cause of the notes about invalid data?

a. NZ is not a valid state value.

b. The variable state is not numeric data.

c. The variable state is not located before CITY.

d. The variable state is not located in positions 24 and 25.

8. Which LIBNAME statement will access the Excel workbook `products.xls`, which contains the worksheet `Children`?

a. `libname myexcel 'products.xls';`

b. `libname myexcel children 'products.xls';`

c. `libname myexcel 'products.xls' sheet='children';`

d. `libname myexcel workbook='products.xls' worksheet='children';`
9. Given the raw data file **1990pop.dat**:

<table>
<thead>
<tr>
<th>CITY</th>
<th>POPULATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>7,322,564</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>3,485,398</td>
</tr>
<tr>
<td>Chicago</td>
<td>2,783,726</td>
</tr>
<tr>
<td>Houston</td>
<td>1,630,553</td>
</tr>
</tbody>
</table>

The following SAS program is submitted:

```sas
data work.population;
  length CITY $ 12;
  <insert statements here>
run;
```

Given the desired output data set **work.population**:

<table>
<thead>
<tr>
<th>CITY</th>
<th>POPULATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>7322564</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>3485398</td>
</tr>
<tr>
<td>Chicago</td>
<td>2783726</td>
</tr>
<tr>
<td>Houston</td>
<td>1630553</td>
</tr>
</tbody>
</table>

Which statements will produce the desired output data set?

a. `infile '1990pop.dat';
   input CITY POPULATION:comma9. dlm='!' @;`

b. `infile '1990pop.dat';
   input CITY $ POPULATION:comma9. dlm='!' @@;`

c. `infile '1990pop.dat' dlm='!';
   input CITY $ POPULATION:comma9. @;`

d. `infile '1990pop.dat' dlm='!';
   input CITY POPULATION:comma9. @@;`
B.4 Creating Variables

Answer the 9 questions in this section in 15 minutes or less. Do not use your course notes.

After completing the questions, compare your answers with the correct answers in Section 8 of this appendix.

Complete the following table after you determine the number of questions that you answered correctly:

<table>
<thead>
<tr>
<th>Section</th>
<th>Number of Questions</th>
<th>Numbers of Questions Answered Correctly</th>
<th>Percent Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating Variables</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Percent Correct
9 correct out of 9 = 100%
8 correct out of 9 = 89%
7 correct out of 9 = 78%
6 correct out of 9 = 67%
5 correct out of 9 = 56%
4 correct out of 9 = 44%
3 correct out of 9 = 33%
2 correct out of 9 = 22%
1 correct out of 9 = 11%
1. The following SAS program is submitted:

```sas
data work.class;
    set sashelp.class(keep=name age);
    if age>=13 then group='Teen';
    if 11<=age<=13 then group='Pre-Teen';
run;
```

What is the value of `group` in the data set `work.class` if an observation has a value of `age` equal to 13?

a. missing  
b. Teen  
c. Pre-  
d. Pre-Teen

2. A DATA step was submitted and a portion of the SAS log is shown below.

```
data new;
    newvar = THIS IS A TEST;
    --
    388
    76
ERROR 388-185: Expecting an arithmetic operator.
ERROR 76-322: Syntax error, statement will be ignored.
run;
```

Which of the following actions resolves the error message?

a. Put quotation marks around THIS IS A TEST.  
b. Put parentheses around THIS IS A TEST.  
c. Add commas between the words THIS IS A TEST.  
d. Add a FORMAT statement declaring `newvar` as character.
3. Which statement must be added to the DATA step in order for SAS to create the temporary \texttt{FIRST.} and \texttt{LAST.} variables?

Enter the keyword of the statement. Do not add leading or trailing spaces to your answer.

4. The following program is submitted:

```
data work.total;
  n1 = 4;
  n2 = .;
  n3 = 10;
  n4 = n1 + n2 + n3;
run;
```

What is the resulting value of \texttt{n4} in the data set \texttt{work.total}?

a. .
b. 14
c. \(4 + . + 10\)
d. \(n1 + n2 + n3\)
5. Given the SAS data set **Work.Employees**:

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jeff</td>
</tr>
<tr>
<td>Dawn</td>
</tr>
<tr>
<td>Mary</td>
</tr>
<tr>
<td>Gene</td>
</tr>
</tbody>
</table>

The following SAS program is submitted:

```sas
data Work.EmployeeCount;
  set Work.Employees;
  Count=Count+1;
run;
proc print data=Work.EmployeeCount noobs;
run;
```

What is the result of the PRINT procedure?

a. |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Jeff</td>
</tr>
<tr>
<td>Dawn</td>
</tr>
<tr>
<td>Mary</td>
</tr>
<tr>
<td>Gene</td>
</tr>
</tbody>
</table>

b. |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Jeff</td>
</tr>
<tr>
<td>Dawn</td>
</tr>
<tr>
<td>Mary</td>
</tr>
<tr>
<td>Gene</td>
</tr>
</tbody>
</table>

c. |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Jeff</td>
</tr>
<tr>
<td>Dawn</td>
</tr>
<tr>
<td>Mary</td>
</tr>
<tr>
<td>Gene</td>
</tr>
</tbody>
</table>

d. |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Jeff</td>
</tr>
<tr>
<td>Dawn</td>
</tr>
<tr>
<td>Mary</td>
</tr>
<tr>
<td>Gene</td>
</tr>
</tbody>
</table>
6. The following program is submitted:

```sas
data personnel;
  hired='01MAR2003'd;
  name='William Smith';
run;
```

Which of the following is true regarding the variables created with the assignment statements?

a. The variables `hired` and `name` are 8 bytes.

b. The variables `hired` and `name` are character.

c. The variable `hired` is 9 bytes and `name` is 13 bytes.

d. The variable `hired` is numeric and `name` is character.

7. Given the SAS data set `birth`:

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tim</td>
<td>CA</td>
</tr>
<tr>
<td>Sue</td>
<td>IN</td>
</tr>
<tr>
<td>Bill</td>
<td>NY</td>
</tr>
</tbody>
</table>

The following SAS program is submitted:

```sas
data birthregion;
  set birth;
  if state='CA' then do;
    region='West';
  end;
  else if state='NY' then do;
    region='East';
  end;
run;
```

What is the result?

a. The program fails execution because of invalid DO block syntax.

b. The program fails execution because there is not a DO block for the `state` value of IN.

c. The program runs without errors or warnings and produces a data set with two observations and three variables.

d. The program runs without errors or warnings and produces a data set with three observations and three variables.
8. Which of the following is true regarding the sum statement?
   a. The sum statement can be used only for variables being read in from a SET statement.
   b. The sum statement initializes the variable to missing before the first iteration of the DATA step.
   c. The sum statement automatically retains the variable value without using a RETAIN statement.
   d. The sum statement produces an error if a missing value is added to the accumulator variable.

9. Which of the following is valid syntax for SELECT and WHEN statements?
   a. 
   ```plaintext
   select(salary);
   when <100000 status='Non-Exec';
   when >=100000 status='Exec';
   end;
   ```
   b. 
   ```plaintext
   select(salary);
   when(<100000) status='Non-Exec';
   when(>=100000) status='Exec';
   end;
   ```
   c. 
   ```plaintext
   select;
   when salary<100000 status='Non-Exec';
   when salary>=100000 status='Exec';
   end;
   ```
   d. 
   ```plaintext
   select;
   when(salary<100000) status='Non-Exec';
   when(salary>=100000) status='Exec';
   end;
   ```
B.5 Manipulating Data

Answer the 9 questions in this section in 15 minutes or less. Do not use your course notes.

After completing the questions, compare your answers with the correct answers in Section 8 of this appendix.

Complete the following table after you determine the number of questions that you answered correctly:

<table>
<thead>
<tr>
<th>Section</th>
<th>Number of Questions</th>
<th>Numbers of Questions Answered Correctly</th>
<th>Percent Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manipulating Data</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Percent Correct

9 correct out of 9 = 100%
8 correct out of 9 = 89%
7 correct out of 9 = 78%
6 correct out of 9 = 67%
5 correct out of 9 = 56%
4 correct out of 9 = 44%
3 correct out of 9 = 33%
2 correct out of 9 = 22%
1 correct out of 9 = 11%
1. Which statement has correct syntax for a DO statement?
   a. `do year = 2000 to 2005 while(amount < 1000000);`
   b. `do year = 2000 to 2005, while(amount < 1000000);`
   c. `do year = 2000 to 2005 or while(amount < 1000000);`
   d. `do year = 2000 to 2005 and while(amount < 1000000);`

2. A DATA step was submitted and a portion of the SAS log is shown below.

   ```sas
   345  data numdates;
   346    set chardates;
   347    newhired=input(hired, date9);
   -----
   85
   76
   ERROR 85-322: Expecting a format name.
   ERROR 76-322: Syntax error, statement will be ignored.
   348  run;
   NOTE: The SAS System stopped processing this step because of errors.
   ```

   Which of the following actions resolves the error message?
   a. Put quotation marks around `date9`.
   b. Put `date9` before `hired`.
   c. Put a period at the end of `date9`.
   d. Delete the comma before `date9`.

3. The following SAS program is submitted:

   ```sas
   data orders;
   set product.orders;
   <insert statement here>
   run;
   ```

   Which SAS statement returns observations containing the text GOLF, regardless of the case, in the variable `product_name`?
   a. `if find(product_name,'golf','t')>0;`
   b. `if find(product_name,'golf','t')=0;`
   c. `if find(product_name,'golf','i')=0;`
   d. `if find(product_name,'golf','i')>0;`
4. The following SAS program is submitted:

```sas
data salesgoals(drop=i);
  set sales(keep=jan feb mar);
  array sales{3} jan feb mar;
  array goals{3} _temporary_ (130,170,170);
  array diff{3};
  <insert DO loop here>
run;
proc print data=salesgoals noobs;
run;
```

The following report is generated:

<table>
<thead>
<tr>
<th>jan</th>
<th>feb</th>
<th>mar</th>
<th>diff1</th>
<th>diff2</th>
<th>diff3</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>185</td>
<td>160</td>
<td>-10</td>
<td>15</td>
<td>-10</td>
</tr>
<tr>
<td>115</td>
<td>160</td>
<td>180</td>
<td>-15</td>
<td>-10</td>
<td>10</td>
</tr>
</tbody>
</table>

Which DO loop created the report?

a. ```sas
do i=1 to 3;
  diff{i}=sales{i}-goals{i};
end;
```  

b. ```sas
do i=1 to 3;
  sales{i}=goals{i}-diff{i};
end;
```  

c. ```sas
do array=1 to 3;
  sales=goals-diff;
end;
```  

d. ```sas
do array=1 to 3;
  diff{array}=sales{array}-goals{array};
end;
```
5. Which of the following is true regarding the SCAN function?
   a. The SCAN function uses only two default delimiters (the blank and the comma) if a delimiter is not specified.
   b. The SCAN function has an optional fourth argument, which is the direction (forward or backward) to read the string.
   c. The SCAN function returns a missing value if the number of the word scanned is greater than the number of words in the character string.
   d. If the SCAN function returns a value to a variable that was not yet assigned a length, the variable length is determined by the length of the first argument.

6. The following SAS program is submitted:

```sas
data investment;
  do year=1 to 5;
    invest+1000;
    do month=1 to 12 by 3;
      invest+50;
      output;
    end;
  end;
run;
```

How many observations are in the data set `investment`?
Enter your numeric answer.

7. Which assignment statement will produce a value for `FULLNAME` with a comma between `LASTNAME` and `FIRSTNAME`?
   a. `FULLNAME = CATS(',', LASTNAME, FIRSTNAME);`
   b. `FULLNAME = CATX(',', LASTNAME, FIRSTNAME);`
   c. `FULLNAME = CATS(LASTNAME, FIRSTNAME, ',');`
   d. `FULLNAME = CATX(LASTNAME, FIRSTNAME, ',');`
8. A character array that contains three variables (name1, name2, and name3) with the values of Smith, Jones, and Westinghouse is requested.

Which ARRAY statement will create the desired array?

a. array name{3} $ ('Smith','Jones','Westinghouse');
b. array name(3) $ 12 ('Smith','Jones','Westinghouse');
c. array name1-name3 $ ('Smith' 'Jones' 'Westinghouse');
d. array name{3} name1-name3 ('Smith' 'Jones' 'Westinghouse');

9. The following SAS program is submitted:

```
data personnel;
  Phone = 6667778888;
  <insert statement here>
run;
proc print data=personnel;
run;
```

The following report is generated:

<table>
<thead>
<tr>
<th>Obs</th>
<th>Phone</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6667778888</td>
<td>666</td>
</tr>
</tbody>
</table>

Which SAS statement created the report?

a. AreaCode=substr(Phone,1,3);
b. AreaCode=substr(Phone,2,3);
c. AreaCode=substr(put(Phone,10.),1,3);
d. AreaCode=substr(input(Phone,10.),1,3);
B.6 Generating Reports

Answer the 9 questions in this section in 15 minutes or less. Do not use your course notes.

After completing the questions, compare your answers with the correct answers in Section 8 of this appendix.

Complete the following table after you determine the number of questions that you answered correctly:

<table>
<thead>
<tr>
<th>Section</th>
<th>Number of Questions</th>
<th>Numbers of Questions Answered Correctly</th>
<th>Percent Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generating Reports</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Percent Correct

9 correct out of 9 = 100%
8 correct out of 9 = 89%
7 correct out of 9 = 78%
6 correct out of 9 = 67%
5 correct out of 9 = 56%
4 correct out of 9 = 44%
3 correct out of 9 = 33%
2 correct out of 9 = 22%
1 correct out of 9 = 11%
1. The following SAS program is submitted:

```sas
proc freq data=sashelp.shoes;
   where region contains 'East' and
   product in ('Sandal','Slipper');
   title 'Sandal and Slipper Report';
run;
```

The following report is generated:

<table>
<thead>
<tr>
<th>Region</th>
<th>Product</th>
<th>Frequency</th>
<th>Percent</th>
<th>Percent</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eastern Europe</td>
<td>Sandal</td>
<td>3</td>
<td>23.08</td>
<td>42.86</td>
<td>50.00</td>
</tr>
<tr>
<td></td>
<td>Slipper</td>
<td>4</td>
<td>30.77</td>
<td>57.14</td>
<td>57.14</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>7</td>
<td>53.85</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>Middle East</td>
<td>Sandal</td>
<td>3</td>
<td>23.08</td>
<td>50.00</td>
<td>50.00</td>
</tr>
<tr>
<td></td>
<td>Slipper</td>
<td>3</td>
<td>23.08</td>
<td>50.00</td>
<td>42.86</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>6</td>
<td>46.15</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>Sandal</td>
<td>6</td>
<td>46.15</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Slipper</td>
<td>7</td>
<td>53.85</td>
<td>100.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td>13</td>
<td>100.00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which SAS statement created the report?

a. `tables region product;`

b. `tables region product / list;`

c. `tables region*product / list;`

d. `tables region*product / crosslist;`
2. Given the following SAS program:

```
options nodate nonumber;
ods listing;

proc univariate data=sashelp.shoes;
  var Sales Inventory;
  label Sales='Sales Total'
    Inventory='Number of Stores';
  title 'Univariate Report';
run;
```

How many of the statements are global statements?

Enter your numeric answer.

3. Given the SAS data set `marriage`:

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thomas</td>
<td>15413</td>
</tr>
<tr>
<td>Susan</td>
<td>17000</td>
</tr>
<tr>
<td>Marsha</td>
<td>16529</td>
</tr>
</tbody>
</table>
```

The following SAS program is submitted:

```
proc print data=marriage;
  <insert statement here>
  format date mmddyy10.;
run;
```

The following report is generated:

```
<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Susan</td>
<td>07/18/2006</td>
</tr>
</tbody>
</table>
```

Which SAS statement created the report subsetting for `date` greater than or equal to January 1, 2006?

a. `where date >= 01/01/2006;`

b. `where date >= '01jan2006';`

c. `where date >= '01jan2006'd;`

d. `where date >= '01/01/2006't;`
4. Given the following program:

```sas
ods html file='report.html';
proc print data=retail;
run;
proc freq data=retail;
run;
<insert ODS statement here>
```

Which ODS statement must end the HTML file?

a. `ods end;`

b. `ods close;`

c. `ods html end;`

d. `ods html close;`
5. The following SAS program is submitted:

```sas
proc freq data=sashelp.cars;
    tables cylinders;
run;
```

The following report is generated:

The following report is desired:

Which OPTIONS statement will produce the desired report?

a. `options nodate page=1;`

b. `options nodate pageno=1;`

c. `options notime nodate number=1;`

d. `options notime nodate pagenumber=1;`
6. The following SAS program is submitted:

```sas
proc print data=account;
   <insert statement here>
run;
```

The following list report containing the character variable `name` and the numeric variables `date` and `amount` is generated:

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>date</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JACK</td>
<td>16MAY2007</td>
<td>$123,830.23</td>
</tr>
<tr>
<td>2</td>
<td>SUE</td>
<td>17JUL2006</td>
<td>$89,654.05</td>
</tr>
</tbody>
</table>

Which SAS statement created the report?

a. `format date date9. amount dollar11.2;`

b. `format date ddmmyy9. amount dollar10.2;`

c. `format date ddmmmyyyy9. amount comma11.2;`

d. `format date date7. amount dollarcomma10.2;`

7. The following SAS program is submitted:

```sas
proc means data=sashelp.orsales maxdec=0;
   class product_line;
   var total_retail_price / sum mean;
run;
```

What is the result?

a. The program runs without errors or warnings and produces a summary table.

b. The program fails execution because of invalid options in the `VAR` statement.

c. The program fails execution because of an invalid option in the `PROC` statement.

d. The program runs with warnings due to the order of the `CLASS` and `VAR` statements.
8. Given the SAS data set **dept** with two character variables:

<table>
<thead>
<tr>
<th>Department</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>0</td>
</tr>
<tr>
<td>IT</td>
<td>1</td>
</tr>
<tr>
<td>Marketing</td>
<td>0</td>
</tr>
<tr>
<td>Sales</td>
<td>5</td>
</tr>
</tbody>
</table>

The following SAS program is submitted:

```sas
proc format;
  value $yesno  '0'='No'
               '1'='Yes'
               'other'='Unknown';
run;

proc print data=dept noobs;
  format answer $yesno.;
run;
```

Which report is created?

a.  

<table>
<thead>
<tr>
<th>Department</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>0</td>
</tr>
<tr>
<td>IT</td>
<td>1</td>
</tr>
<tr>
<td>Marketing</td>
<td>0</td>
</tr>
<tr>
<td>Sales</td>
<td>5</td>
</tr>
</tbody>
</table>

b.  

<table>
<thead>
<tr>
<th>Department</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>No</td>
</tr>
<tr>
<td>IT</td>
<td>Yes</td>
</tr>
<tr>
<td>Marketing</td>
<td>No</td>
</tr>
<tr>
<td>Sales</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

c.  

<table>
<thead>
<tr>
<th>Department</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>No</td>
</tr>
<tr>
<td>IT</td>
<td>Yes</td>
</tr>
<tr>
<td>Marketing</td>
<td>No</td>
</tr>
<tr>
<td>Sales</td>
<td>5</td>
</tr>
</tbody>
</table>

d.  

<table>
<thead>
<tr>
<th>Department</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>Unknown</td>
</tr>
<tr>
<td>IT</td>
<td>Unknown</td>
</tr>
<tr>
<td>Marketing</td>
<td>Unknown</td>
</tr>
<tr>
<td>Sales</td>
<td>Unknown</td>
</tr>
</tbody>
</table>
9. Which program will create a file that can be opened in Microsoft Excel?

a. 
```sas
ods rtf file='report.rtf';
proc freq data=sashelp.shoes;
   tables region;
run;
ods rtf close;
```

b. 
```sas
ods excel file='report.xls';
proc freq data=sashelp.shoes;
   tables region;
run;
ods excel close;
```

c. 
```sas
ods excelxp file='report.xml';
proc freq data=sashelp.shoes;
   tables region;
run;
ods excelxp close;
```

d. 
```sas
ods csvall file='report.csv';
proc freq data=sashelp.shoes;
   tables region;
run;
ods csvall close;
```
# B.7 Scores

Complete the following table as you complete the sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Number of Questions</th>
<th>Numbers of Questions Answered Correctly</th>
<th>Percent Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS Fundamentals Concepts</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Working with SAS Data Sets</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Working with Raw Data and Excel Files</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creating Variables</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manipulating Data</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generating Reports</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Percent Correct**

- 45-50 correct out of 50 = 90-100%
- 40-44 correct out of 50 = 80-89%
- 35-39 correct out of 50 = 70-79%
- 33-34 correct out of 50 = 65-69%
- 30-32 correct out of 50 = 60-64%
- 25-29 correct out of 50 = 50-59%
- 20-24 correct out of 50 = 40-49%
- 15-19 correct out of 50 = 30-39%
## B.8 Answers

<table>
<thead>
<tr>
<th>Question Number</th>
<th>B.1</th>
<th>B.2</th>
<th>B.3</th>
<th>B.4</th>
<th>B.5</th>
<th>B.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A.</td>
<td>B.</td>
<td>2</td>
<td>C.</td>
<td>A.</td>
<td>D.</td>
</tr>
<tr>
<td>2</td>
<td>B.</td>
<td>A.</td>
<td>C.</td>
<td>A.</td>
<td>C.</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>B.</td>
<td>D.</td>
<td>D.</td>
<td>BY</td>
<td>D.</td>
<td>C.</td>
</tr>
<tr>
<td>4</td>
<td>D.</td>
<td>C.</td>
<td>B.</td>
<td>A.</td>
<td>A.</td>
<td>D.</td>
</tr>
<tr>
<td>5</td>
<td>A.</td>
<td>D.</td>
<td>C.</td>
<td>A.</td>
<td>C.</td>
<td>B.</td>
</tr>
<tr>
<td>6</td>
<td>A.</td>
<td>C.</td>
<td>D.</td>
<td>20</td>
<td>A.</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>C.</td>
<td>B.</td>
<td>A.</td>
<td>B.</td>
<td>B.</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>201</td>
<td>A.</td>
<td>C.</td>
<td>B</td>
<td>C.</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A.</td>
<td>D.</td>
<td>D.</td>
<td>C.</td>
<td>D.</td>
<td></td>
</tr>
</tbody>
</table>