Manual de Programación en SAS

Javier Portela García-Miguel

Facultad de Estudios Estadísticos
Universidad Complutense de Madrid

ISBN **978-84-96866-12-6**

1ª Edición: 2007

Índice

Introducción

El paquete estadístico SAS es utilizado habitualmente por gran cantidad de profesionales de sectores variados como pueden ser el análisis econométrico y financiero, la medicina, estudios de mercados, biología y en general todas las disciplinas donde el tratamiento estadístico de datos sea de cierta importancia.

Este manual de programación en el lenguaje utilizado por el paquete estadístico SAS pretende ser una primera guía de aprendizaje para aquellos usuarios que quieran iniciarse en los conceptos básicos del lenguaje SAS, con la intención de utilizar este paquete con cierta frecuencia. El libro está pues destinado a todos aquellos profesionales que deseen in en este paquete estadístico, pero también a estudiantes de diversas carreras como Ciencias Matemáticas, Estadística, Ciencias Económicas o cualquier carrera en la que se utilice el programa.

Las técnicas explicadas en el libro están planteadas siempre en compañía de ejemplos prácticos, donde se ponen de manifiesto aplicaciones directas de interés del concepto estudiado. Los ejemplos suelen ser cortos y sencillos, aunque a medida que se avanza en los conocimientos entrañan más complejidad.

Tras un primer capítulo de introducción a las posibilidades que ofrece el SAS, se presenta en el Capítulo 2 el esquema básico de los dos bloques que forman un programa básico SAS. En el Capítulo 3 se estudia la lectura de datos y creación de archivos SAS, a partir de diferentes estructuras. El Capítulo 4 estudia el funcionamiento básico del paso DATA, necesario para la programación general en SAS. En el siguiente capítulo se verán todas las formas de combinación y lectura de archivos SAS. Las principales funciones básicas SAS, entre las que se cuentan funciones matemáticas como de tratamiento de caracteres, son presentadas en el Capítulo 6. El Capítulo 7 presenta las sentencias condicionales en el entorno del paso DATA, con sus aplicaciones. En el Capítulo 8 y 9 se estudia, respectivamente, la utilización de bucles y de arrays en el entorno del paso DATA.

Los procedimientos SAS tienen una sintaxis general que es estudiada en el Capítulo 10. El Capítulo 11 presenta formatos de escritura de variables de ceirta aplicación en los procedimientos SAS. El Capítulo 12 ilustra algunos de los procedimientos del módulo SAS/BASE, insistiendo en el aspecto de la utilización de archivos de salida de los procedimientos para su utilización posterior.

Finalmente, se presentan en el Capítulo 13 las macros, herramienta de gran utilidad puesto que permite utilizar el paquete estadístico SAS en toda su potencia, creando aplicaciones que combinen las librerías estadísticas (procedimientos) con bloques más o menos complejos de programación.

Capítulo 1. Características técnicas del SAS

El SAS (Statistical Analysis System) es un complejo sistema informático cuyas características más importantes son la combinación de un lenguaje de programación con herramientas y aplicaciones estadísticas, gráficas y de bases de datos.

El SAS se adquiere por módulos. Algunos son:

BASE

El modulo BASE es el núcleo del sistema SAS. Contiene el interprete del lenguaje, así como todo el sistema de manipulación de datos y algunos procedimientos de estadística básica y de generación de informes, entre los que cabe citar:

CHART

Histogramas en dos y tres dimensiones, diagramas circulares, en estrella y otros (modo carácter).

CORR

Calcula distintos coeficientes de correlación: Pearson, Spearman, Kendall, Hoeffding y

otros. Pruebas de hipotesis.

FORMAT

Define formatos de variables. Categorización.

FREQ

Tablas de frecuencia. Tablas de contingencia. Pruebas y medidas de asociación.

MEANS

Estadísticos univariantes básicos para variables cuantitativas: media, varianza, error tipico, etc. Pruebas de hipotesis para la media.

PLOT

Gráficos de coordenadas rectangulares X-Y (modo carácter).

PRINT

Impresión de conjuntos de datos SAS.

RANK

Devuelve el número de orden de las observaciones. Estos ordinales o rangos pueden usarse para realizar pruebas no paramétricas.

REPORT

Generación de listados e informes.

SORT

Ordena las observaciones de un conjunto de datos.

SQL

Manejo de datos mediante lenguaje SQL.

STANDARD

Tipificación de variables, para una media y desviación típica dadas.

SUMMARY

Estadística descriptiva (variables cuantitativas). Como MEANS pero sin presentar resultados.

TABULATE

Tabulación de datos, pudiendo elaborar estadísticos descriptivos de MEANS, FREQ y SUMMARY para cada celdilla.

UNIVARIATE

Elabora estadísticos descriptivos de la distribución, incluyendo cuartiles.

STAT

El modulo STAT implementa las técnicas estadísticas mas usuales, incluyendo entre otros los siguientes procedimientos:

ACECLUS

Estima las covarianzas, autovalores y autovectores dentro de grupos, para el análisis cluster y para el análisis canónico.

ANOVA

Análisis de la varianza para cualquier modelo de diseño equilibrado. Análisis multivariante de la varianza. Contrastes múltiples de medias o pruebas a posteriori.

CALIS

Estima constantes y parámetros usando mínimos cuadrados, mínimos cuadrados generalizados o máxima verosimilitud.

CANCORR

Correlación canónica, correlación parcial canónica y análisis de redundancia canónica.

CANDISC

Análisis discriminante canónico, distancias de Mahalanobis y análisis de la varianza univariante y multivariante.

CATMOD

Modelos lineales para datos categorizados.

CLUSTER

Once métodos de análisis de grupos (cluster) para la ordenación jerárquica de conjuntos de datos. La salida puede ser tratada con el procedimiento TREE para la presentación de dendrogramas.

CORRESP

Análisis de correspondencias, simple y múltiple.

DISCRIM

Análisis de varias funciones discriminantes para la clasificación de observaciones en dos o mas grupos, utilizando una o mas variables. Utiliza tanto métodos paramétricos (variables cuantitativas normales), como no paramétricos.

FACTOR

Análisis factorial y de componentes principales. Realiza tanto rotaciones ortogonales como oblicuas.

FASTCLUS

Análisis cluster rápido, para gran cantidad de datos.

FREQ

Formación y análisis de tablas de contingencia. Análisis no paramétricos para uno o varios criterios de clasificación. Realiza múltiples pruebas chi-cuadrado así como correlaciones no paramétricas.

GLM

Modelo lineal general. Realiza múltiples pruebas de hipotesis: regresión, regresión múltiple, análisis de la varianza para diseños desequilibrados y con factores de efectos fijos y/o aleatorios, análisis de la covarianza, contrastes ortogonales o modelos de superficie respuesta, correlación parcial, análisis multivariante de la varianza.

LATTICE

Análisis de la varianza y covarianza para dichos diseños.

LIFEREG

Análisis de supervivencia.

LIFETEST

Análisis de supervivencia.

LOGISTIC

Regresión logística para datos de respuesta binaria u ordinal, por máxima verosimilitud.

NESTED

Análisis de la varianza y covarianza para modelos jerárquicos puros (factores de efectos aleatorios).

NLIN

Regresión no lineal. Ajustes mínimo-cuadraticos a modelos no lineales.

NPAR1WAY

Pruebas no paramétricas para modelos de análisis de la varianza de rangos.

ORTHOREG

Regresión (método Gentleman-Givens).

PHREG

Regresión para datos de supervivencia.

PLAN

Diseño de experimentos. Aleatoriza y construye diseños de experimentos tanto factoriales como jerárquicos.

PRINCOMP

Análisis en componentes principales.

PRINQUAL

Análisis en componentes principales de datos cualitativos (coordenadas principales).

PROBIT

Análisis Probit.

REG

Regresión lineal.

RSREG

Superficie respuesta cuadrática, para determinar la combinación óptima de niveles de los factores o respuesta óptima.

STEPDISC

Análisis discriminante paso a paso.

TRANSREG

Obtiene transformaciones lineales y no no lineales de las variables para su ajuste a la regresión, correlación canónica y modelos de análisis de la varianza.

TREE

Realiza el dendrograma con las salidas de los procedimientos CLUSTER y VARCLUS.

TTEST

Test t de Student para contraste de dos medias.

VARCLUS

Análisis cluster jerarquizado.

VARCOMP

Componentes de la varianza.

GRAPH

El módulo GRAPH ofrece un conjunto de procedimientos diseñados para la obtención de todo tipo de gráficos, de presentación y científicos. Incluye un editor de gráficos interactivo, una colección de mapas de diversas regiones del mundo y una librería con drivers para un gran número de dispositivos: X Window, PostScript, CGM, HP, Tektronix, etc. Algunos procedimientos gráficos son:

GCHART

Histogramas y diagramas circulares.

GCONTOUR

Gráficos de contorno (isolíneas).

GMAP

Representación de datos sobre mapas geográficos.

GPLOT

Gráficos XY bidimensionales.

GPRINT

Presentación gráfica de ficheros texto.

GPROYECT

Proyección plana de mapas definidos por coordenadas.

GREDUCE

Reducción del número de puntos para dibujar un mapa.

GREMOVE

Eliminación de subdivisiones en mapas.

GSLIDE

Preparación de transparencias.

G3D

Gráficos XYZ tridimensionales. Superficies.

Otros módulos son:

OR-. Investigación Operativa.

ETS-. Análisis de Series Temporales.

QC-. Control de Calidad.

AF-. Construcción de Aplicaciones SAS.

FSP-. Utilidades para presentar o manejar Archivos de datos SAS.

IML-. Módulo de lenguaje matricial.

Capítulo 2. Estructura General de un Programa SAS

Básicamente un programa SAS se compone de Bloques DATA y Bloques PROC, en principio independientes.

Bloque o Paso DATA-. Son bloques de programación cuya finalidad son operaciones de lectura y creación de archivos SAS, en las que se pueden utilizar sentencias de programación habituales como bucles y sentencias condicionales. El objetivo de un **bloque o paso data** es crear el archivo nombrado en la sentencia *data* inicial. Comienzan con la orden *data* y su fin está determinado si existe otra sentencia *data* posterior, una sentencia de procedimiento **proc**, o bien la sentencia **run;** o **cards;**.

En principio, en este manual, si no existe una sentencia **cards;**, finalizaremos el bloque data con la sentencia **run;** por claridad en la presentación visual de los bloques. Desde el punto de vista de la programación, no sería necesario si hay procedimientos o pasos data posteriores que finalizan con run; y se ejecutan todos a la vez.

Ejemplo de bloque data

```
data uno dos;
input a b @@;
if a>b then output uno;
    else output dos;
cards;
4 5 7 6 8 9
;
```

Bloque PROC-. Son subrutinas o procedimientos de sintaxis rígida, que incluyen utilidades de tratamiento de datos, subrutinas estadísticas o de investigación operativa complejas (PROC FREQ, PROC REG, PROC FACTOR...), programas gráficos (PROC GPLOT, PROC GCHART), etc. El fin de un bloque o paso proc está determinado por la sentencia **run;**.

Ejemplo de bloque proc

```
proc univariate data=uno plot normal;by y;var x;output out=dos
max=maxim1 mode=moda1 skewness=asim1;
run;
```

En general un programa básico SAS puede constar de varios bloques data y bloques procs combinados para obtener el resultado deseado.

Sintaxis general -.Cada sentencia con sus opciones debe finalizar por un punto y coma: ";". Las sentencias pueden escribirse seguidas en una línea, y pueden comenzar en una línea y terminar en la siguiente. El SAS no tiene en cuenta los espacios en blanco entre palabras para la compilación.

Salidas de los programas-. En general, los errores y la información sobre el proceso sale en la ventana LOG y las salidas de los PROC en la ventana OUTPUT, salvo que se cambien las opciones del sistema por defecto.

Capítulo 3. Lectura y escritura de datos

Tipos de Archivos Sas

Los archivos de datos SAS sólo pueden ser leídos por el propio sistema SAS y contienen los datos (valores de las observaciones para las variables del archivo), así como información sobre las variables (etiquetas de variables y valores, tipo…), fecha, etc.

Archivos SAS permanentes

Si se desea trabajar con un archivo SAS ya existente y que está guardado en una unidad de disco (disquetera o disco duro), o si se desea crear un archivo SAS nuevo y guardarlo en una unidad de disco, es necesario asignar un nombre, que llamaremos *libref* , al directorio en el que se va a trabajar. Este *libref* va a permanecer activo hasta que cerremos el sistema.

Para definir el *libref* se ha de ejecutar la sentencia libname en el editor de programas. Por ejemplo, para asignar el *libref* PEPE a la unidad de disco a:, se ejecuta:

```
libname pepe 'a:';
```

Ahora, si se tiene un Archivo SAS con el nombre *matriz* en un disco en a: y se quiere trabajar con él, realizando por ejemplo un análisis estadístico univariante de las variables que contiene, el archivo en cuestión debe ser nombrado como *pepe.matriz*, donde *pepe* indica el nombre virtual del directorio y *matriz* es el nombre del archivo SAS:

```
proc univariate data=pepe.matriz;run;
```

Los archivos permanentes tienen dos nombres: un nombre virtual para ser utilizado solamente dentro del sistema SAS, que tendrá la forma *libref.archivo,* y un nombre de archivo dentro del sistema operativo, que es *archivo.sas7bdat.*

Ejemplo: nombres de archivos permanentes dentro y fuera del sistema SAS

```
libname pepe 'c:\datos\';
data pepe.matriz;
input x y;
cards;
1 2
4 6
;
```

El archivo creado es el archivo permanente pepe.matriz con dos variables y dos observaciones:

```
X Y

1 2
4 6
```

Su nombre dentro y fuera del sistema es:

Dentro del sistema SAS	**Fuera del sistema SAS (sistema operativo)**
pepe.matriz	c:\datos\matriz.sas7bdat

Archivos SAS temporales

Se utilizan sólo dentro de la sesión abierta del sistema SAS. Al salir de ella se pierden de la memoria. Se nombran con una única palabra: *archivo* . Se utiliza memoria de disco duro tanto para ellos como para los archivos permanentes, con lo que la única ventaja que tienen los temporales es el tener nombres más cortos y no ocupar lugar en el disco duro una vez que salimos del entorno SAS. Si por ejemplo se ha creado el archivo temporal *matriz2* y se desea realizar un análisis estadístico univariante de las variables presentes en el archivo, el programa SAS sería:

```
proc univariate data= matriz2;
run;
```

Creación de un Archivo SAS

Se presentan los siguientes esquemas de creación de archivos, que responden a diferentes modos de lectura. El objetivo en cada uno de estos programas es crear un *archivo SAS.*

Esquemas de creación de un archivo SAS

Los datos están en un archivo en una ubicación física (disco duro, Cd, etc.) en modo texto:

```
DATA archivo SAS;
  INFILE archivo de datos;
  INPUT...;
RUN;
```

Los datos se introducirán en el editor de texto:

```
DATA archivo SAS;
  INPUT...;
CARDS;
líneas de datos...
;
```

Los datos van a ser leídos del archivo SAS 'c:\matriz.sas7bdat':

```
LIBNAME pepe 'c:\';
DATA archivo SAS;
  SET pepe.matriz;
RUN;
```

Los datos van a ser leídos de un archivo con formato EXCEL, DBASE, etc.:

```
PROC IMPORT datafile="archivo" out=archivo SAS;
opciones;
RUN;
```

Una vez "leídos" los datos estos pasan a estar en la memoria temporal o permanente. En la ventana LOG es mostrado un mensaje sobre la operación, archivo creado y variables y casos leídos, pero el archivo no es "mostrado". Para visionarlo, se puede utilizar el procedimiento proc print, que presenta un listado en la ventana OUTPUT de las observaciones y variables del archivo SAS:

```
proc print data=archivo SAS;
run;
```

Para estudiar la sintaxis con detalle de los esquemas de creación de archivos es necesario comentar el funcionamiento de cada sentencia SAS por separado.

Sentencia DATA;

Se estudiará a continuación la sintaxis de la sentencia data. La sentencia data define el nombre del (los) archivos SAS que se quieren crear. Además, sirve como cabeza de un bloque data, señalando el final de pasos data o proc anteriores. Su sintaxis es:

data *archivo*;

y su utilización para creación de archivos permanentes se puede ver con el siguiente esquema de ejemplo:

```
libname disco 'c:\datos';
data disco.nume;
otras sentencias SAS;
```

Con este programa se ha creado un archivo SAS en el directorio c:\datos\, que se llamará nume.sas7bdat cuando salga del SAS.

Para archivos temporales, la utilización de la sentencia data es:

```
data uno;
otras sentencias SAS;
```

De este modo se creará un archivo SAS temporal que se llamará *uno*. En el siguiente ejemplo se crean dos archivos SAS:

```
libname disco 'c:/datos';
data uno disco.nume;
otras sentencias SAS;
```

Son creados dos archivos, uno temporal, llamado *uno* , y uno permanente, llamado *disco.nume*.

Como se ha dicho, si no existe una sentencia **cards;**, se finalizará el bloque data con la sentencia **run;** por claridad en la presentación visual de los bloques.

Sentencia INFILE;

Esta sentencia define el archivo de donde se van a leer los datos, si éstos están en un archivo en formato ASCII (modo texto).

```
INFILE 'archivo' opciones;
```

La sentencia *infile* tiene varias opciones de interés:

dlm='caracter de separación'-. Se usa cuando los datos están separados por comas u otro carácter de separación en vez de espacios en blanco.

firstobs=número de líneaa-. Primer nº de línea que se quiere leer.

obs=nº de línea-. Ultima línea que se quiere leer si la lectura de datos con *input* es secuencial.

missover-. Con *input* en lectura en formato libre, evita el salto del puntero a la línea siguiente para buscar el valor no encontrado de la última variable.

eof=variable-. Crea una variable que indica el final de archivo, cuando el puntero de lectura no encuentra más datos que leer.

Sentencia CARDS;

Anuncia que los datos van a estar ubicados en el mismo programa SAS, en el editor de texto (introducidos por teclado o pegados). Los datos deben escribirse a partir de la primera columna en los formatos fijos. (En formato libre no importa).

```
cards;
  líneas de datos...;
;
```

Sentencia INPUT;

Describe la lectura de las variables de los datos de la matriz descrita en INFILE o escrita con CARDS. La orden INPUT es secuencial: lee los datos uno por uno del archivo de texto o del editor.

Lectura por columnas

```
INPUT var [$] colinicial-colfinal [.decimales];
```

$ Expresa que la variable es alfanumérica. El signo se utilizará con el mismo sentido en los restantes tipos de formato de lectura . La longitud en caracteres del nombre de las variables por defecto es 8.

En lectura por columnas, el **puntero de lectura** se sitúa en la columna posterior a la columna final indicada.

Ejemplo: Lectura por columnas

```
data uno;
  input edad 1-2 sexo $ 3 peso 4-6 .1;
cards;
24H804
12M337
15M384
;
```

Se ha creado el archivo SAS temporal uno, con las variables y valores siguientes:

```
EDAD  SEXO  PESO

24 H  80.4
12 M  33.7
15 M  38.4
```

Para ver en la ventana OUTPUT el contenido del archivo se utilizará el procedimiento print:

```
proc print data=uno;
run;
```

Si las variables alfanuméricas son largas es necesario utilizar en el paso data la sentencia **length** *var. $ n*, que define la longitud máxima *n* de la variable, que por defecto es 8.

Ejemplo: lectura por columnas de variables alfanuméricas largas

```
data uno;
  length provincia $ 30;
  input provincia $ 1-27 codigo 28-30;
cards;
La Coruña                      345
Las Palmas de Gran Canaria     260
Orense                         113
;
```

Se ha creado el archivo SAS temporal *uno* con las variables provincia y codigo:

```
PROVINCIA                  CODIGO
La Coruña                  345
Las Palmas de Gran Canaria 260
Orense                     113
```

Para ver en la ventana OUTPUT el contenido del archivo se utilizará el procedimiento print:

```
proc print data=uno;
run;
```

Lectura en formato libre

Se lee cada variable hasta encontrarse con el siguiente espacio en blanco, desde la posición actual del puntero de lectura.

Tras la lectura de una variable, el puntero de lectura se sitúa en la columna posterior al espacio en blanco (segunda columna tras la última cifra de la variable).

```
INPUT var [$];
```

Ejemplo: Lectura de datos separados por espacios

Si los datos están en el archivo 'c:\paco.txt' en modo texto, y los datos están separados por espacios:

```
24 H 80.4
12 M 33.7
15   M 38.4
```

Se leerían así:

```
data uno;
  infile 'c:\paco.txt';
  input edad sexo $ peso ;
run;
```

Así se ha creado el archivo temporal *uno:*

```
EDAD  SEXO  PESO

24    H     80.4
12    M     33.7
15    M     38.4
```

Para ver en la ventana OUTPUT el contenido del archivo se utilizará el procedimiento print:

```
proc print data=uno;
run;
```

Si los datos en lugar de estar separados por espacios están separados por comas u otro símbolo, es necesario utilizar la sentencia infile con la opción dlm=",". Si los datos se introducen por teclado, se utiliza la opción *infile cards*; si por el contrario están en un archivo de texto, se utiliza el formato *infile archivo;*.

En esta opción es necesario considerar que los datos no pueden estar separados por puntos y coma, pues éste es un símbolo muy utilizado en las sentencias SAS. Si los datos están separados en el archivo original por el símbolo punto y coma ";", se puede utilizar, antes de su lectura, cualquier editor de texto para reemplazarlo automáticamente por otro símbolo como la coma.

Ejemplo: Lectura de datos separados por comas

Si los datos están en el archivo 'c:\paco.txt' en modo texto, y los datos están separados por comas:

```
24,H,80.4
12,M,33.7
15,M,38.4
```

Se leerían así:

```
data uno;
   infile 'c:\paco.txt' dlm=',';
   input edad sexo $ peso ;
run;
```

Así se crea correctamente el archivo *uno.*

Es importante conocer el funcionamiento de la sentencia input por defecto **en formato libre**:

(1) Los . son leídos como valores missing.

(2) Cuando el puntero de lectura acaba de leer todas las variables de una Observación, salta a la siguiente línea de texto para comenzar a leer la siguiente Observación. Esta manera de funcionar puede anularse utilizando la opción @@, que indica al puntero de lectura que debe

seguir en la misma línea hasta que no encuentre texto diferente de espacios en blanco, leyendo las variables por orden de lista y recomenzando cada vez la lectura de la lista si es necesario.

(3) Si el puntero de lectura no encuentra el valor de una variable (es decir, no encuentra texto donde debería estar el valor de una variable), lo busca en la línea siguiente (salvo cuando se utiliza la opción missover en la sentencia INFILE).

En los ejemplos siguientes se puede observar el proceso de lectura por defecto.

Ejemplo: Lectura por defecto en formato libre

```
data uno;
  input a b c;
cards;
12 2
4
1 3
3 5 6
5 . 8 8
;
```

Grabaría en el archivo temporal uno:

```
A    B    C

12   2    4
1    3    3
5    .    8
```

Ejemplo: Lectura por defecto en formato libre

```
data tres;
  input edad sexo $ peso ;
cards;
24
H 80.4
12
M 33.7
;
```

Grabaría en el archivo temporal tres:

```
EDAD  SEXO  PESO

24 H  80.4
12 M  33.7
```

Ejemplo: Uso de la opción @@ cuando una línea contiene varios casos

```
data dos;
  input edad sexo $ peso @@;
cards;
24 H 80.4 12 M 33.7
15 M 38.4
;
```

Grabaría en el archivo temporal dos:

```
EDAD  SEXO  PESO

24   H    80.4
12   M    33.7
15   M    38.4
```

Lectura de variables alfanuméricas en formato libre

(1) Si el archivo de texto está en formato libre, y la variable de texto ocupa más de 8 caracteres, se debe añadir al paso data la sentencia **length** *var. $ n*, donde n es el número máximo de caracteres de la variable.

(2) El símbolo **&** en lectura en formato libre indica que hacen falta dos blancos para cambiar de variable. Se utiliza para leer variables alfanuméricas que contienen un espacio interno entre cada palabra.

Ejemplo: lectura de variables alfanuméricas largas

```
data uno;
  length nombre $ 15;
  input nombre $ @@;
cards;
Cantalapiedra Gonzalez Shostakovich
;
```

leería correctamente los datos:

```
NOMBRE
Cantalapiedra
Gonzalez
Shostakovich
```

Ejemplo: Lectura de variables alfanuméricas que contienen espacios

Si queremos leer la variable nombre (que incluye el apellido) en los datos:

```
data uno;
  length nombre $ 30;
  input nombre $ &;
cards;
Paco Pérez 18
Maria Méndez 22
;
```

Es creado el archivo SAS *uno*, con la única variable nombre y dos observaciones:

```
Nombre

Paco Pérez
Maria Méndez
```

En este ejemplo hay que señalar que los datos numéricos 18 y 22 de la edad no son leídos, debido a que en el funcionamiento por defecto de input, al terminar de leer la única variable nombrada (la variable nombre) , el puntero de lectura salta de línea.

Si las variables alfanuméricas contienen en ocasiones más de un espacio, o bien por ejemplo en algunas observaciones contienen un espacio pero en otras no, y hay más variables presentes en los datos, las soluciones de lectura alternativas son:

1) Lectura en formato por columna, si los datos están organizados en ese modo.
2) Lectura con formato (se verá a continuación), equivalente a menudo a la lectura por columnas.
3) Variables separadas por comas u otro símbolo.

En todo caso con variables alfanuméricas siempre se tendrá cuidado de utilizar la opción **length** para cubrir toda la longitud (si la longitud asociada es mayor que la necesaria no ocurre nada, simplemente se utiliza un poco más de memoria).

Notación abreviada para la lectura de varias variables en formato libre

Si se desean asignar los nombres x1 a x15 a las 15 variables que se quieren leer en formato libre, se utilizará la notación abreviada

```
input x1-x15;
```

y las 15 variables serán leídas con formato libre, asignándoles los nombres x1, x2, x3, ..., x15.

Lectura con formato

En este tipo de lectura el puntero "cuenta" los espacios desde el lugar donde está situado, en contraposición a la lectura por columnas en la cual las variables se supone que están siempre en las mismas columnas. Este tipo de lectura especialmente útil cuando cada observación

ocupa varias líneas de texto.

`INPUT` *`variable`* `formato;`

El `formato` refleja las características de formato de lectura de la variable. Tras la lectura de ésta, el **puntero de lectura** se sitúa en la columna **posterior** a la última columna de la variable leída.

Algunos formatos de lectura habituales son los siguientes, donde w indica la longitud de la variable (width) y d el número de decimales, si procede:

Formato	**Ejemplo**	**Resultado**
w.d	`input a 3.2;`	lee el valor 345 como 3.45 (w indica las cifras y d los decimales)
bzw.d	`input a bz4.2;`	Los espacios en blanco se leen como ceros: Lee 2 45 como 20.45
Commaw.d	`input a comma10.2;`	No considera los caracteres extraños: Lee 2,45%&$456 como 2454.56
$w.	`input a $6.;`	Lee variables alfanuméricas: Lee kiosco

Ejemplo: lectura con formato

El programa

```
data uno;
  length nombre $ 20;
  input nombre $10. altura 3.2 ingresos comma7. peso bz3.1;
cards;
Pio Baroja165200,0007 5
;
```

Se crea el archivo SAS temporal *uno*:

```
NOMBRE          ALTURA     INGRESOS   PESO
Pio Baroja      1.65       200000     70.5
```

Normalmente la lectura con formato se utiliza combinada con instrucciones de control de cursor o puntero de lectura.

Controles de Cursor

A veces se quiere dirigir con exactitud el cursor=puntero de lectura, indicándole a qué línea y columna debe ir en cada observación para leer una determinada variable. Este modo de actuar es frecuente sobre todo en lectura con formato Los siguientes símbolos se pueden utilizar en la sentencia input:

@var	Mueve el cursor a la columna *var* (*var* puede ser una variable o un número).
+var	Avanza *var* columnas, contando la columna actual del cursor.
#var	Mueve el cursor a la línea *var*.
/	Salta a la línea siguiente.
@ (puesto al final de la sentencia)	Mantiene el cursor en la línea actual, para que se puedan utilizar varias sentencias INPUT sobre el mismo conjunto de datos.
@@ (puesto al final de la sentencia)	Mantiene el cursor en la misma línea para leer varios casos en la misma línea.

La principal utilidad de la lectura con formato y utilización de puntero de lectura es cuando cada observación ocupa varias líneas de texto. También es importante saber que se puede combinar la lectura con formato con la lectura por columnas, siendo esta combinación bastante habitual.

Ejemplo: observaciones que ocupan más de una línea

```
data uno;
   length nombre direccion $ 60;
   input
   #1 nombre $58. edad 59-60 sexo $ 61
   #2 direccion $60. codigo;
cards;
Pedro Pérez García                                          25V
Avenida de la Ilustración,29                                 28021
María López Maesó                                           23M
Calle Antonio López,42                                       28012
;
```

crearía el archivo SAS *uno*:

```
NOMBRE                EDAD SEXO DIRECCION                      CODIGO
Pedro Pérez García    25   V    Avenida de la Ilustración,29  28021
María López Maesó     23   M    Calle Antonio López,42        28012
```

En el anterior ejemplo se han utilizado las sentencias de control de cursor #1 y #2 para controlar cuál es la línea de lectura **de cada observación** a la que se dirige, es decir, #25 no significa “ir a la línea 25 de entre todas las líneas de texto”, sino “ir a la línea #25 de cada

observación presente en las líneas de texto". El número de líneas total de cada caso está determinado por la última línea nombrada en la sentencia **input**. En el ejemplo, ésta es la línea 2, con lo cual se está determinando que en el archivo cada caso (observación) ocupa 2 líneas de texto.

En caso de que cada observación ocupe varias líneas hay que indicar obligatoriamente en la sentencia INPUT, que sitúe el puntero de lectura en la **última** línea de cada observación, para que automáticamente comience a leer en la **primera** línea de texto de la siguiente observación. Esto se realiza con la sentencia de puntero #n, al final de la sentencia INPUT.

Ejemplo: observaciones que ocupan varias líneas

```
data uno;
  input edad 1-2 @3 sexo $ /  peso 1-3 .1 altura #3;
cards;
24H
804 1.75
435643643643
12M
337 1.60
234525252552
15M
384 1.70
457456465465
;
```

Se crearía correctamente el archivo uno:

```
EDAD  SEXO  PESO
24    H     80.4
12    M     33.7
15    M     38.4
```

En caso de que no todas las observaciones ocupen el mismo número de líneas de texto, y exista una variable de control entre las variables que se van leyendo, que indica cuántas líneas ocupa la observación en curso, es necesario utilizar sentencias condicionales. El siguiente ejemplo, donde la variable control toma el valor 1 si la observación ocupa una línea y valor 2 si tiene 2 líneas, ilustra ésta posibilidad.

Ejemplo: observaciones con número variable de líneas o número variable de variables a leer

```
data uno;
   input x1 x2 control @;
   if control=2 then input #2 x3 x4;
cards;
5 23 1
6 34 2
6 7
4 23 1
3 45 2
5 6
;
```

Genera el archivo uno:

X1	X2	CONTROL	X3	X4
5	23	1	.	.
6	34	2	6	7
4	23	1	.	.
3	45	2	5	6

En el ejemplo mostrado, si la variable CONTROL toma valor 1 el puntero de lectura pasa a leer la siguiente observación en la línea posterior. Si CONTROL es 2, el puntero de lectura lee las variables X3 y X4 en la línea #2 de la observación. El símbolo @ es necesario para que la primera sentencia input conserve el puntero de lectura en la misma línea, a la espera de lo que se determine en la sentencia condicional (el símbolo @ sólo tiene efecto cuando hay más sentencias input posteriores en el programa, si no el funcionamiento de input es el mismo que por defecto, saltando de línea para leer la siguiente observación).

Cuando la observación solamente ocupa una línea las variables X3 y X4 no son leídas, por lo cual su valor es ausente (missing), representado por un punto en el archivo SAS *uno*.

La utilización de las opciones de la sentencia infile permite más flexibilidad en la lectura de datos. Si los datos están presentes en el editor de texto, basta utilizar la opción infile cards;. El siguiente ejemplo ilustra cómo utilizar la sentencia infile para leer solamente algunas de las observaciones del texto, y no todas.

Ejemplo: lectura de un subconjunto de observaciones

```
data uno;
  infile cards firstobs=2 obs=2;
  input a b;
cards;
1 6
9 6
45 4
3 1
;
```

crea el archivo SAS temporal uno, con los datos:

```
A  B

9  6
45 4
```

La opción firstobs= indica la primera observación a leer, y la opción obs=, el número de observaciones a leer a partir de la primera leída. El número de líneas de texto que ocupa cada observación es determinado por el número de líneas mencionado en la sentencia input.

Notación abreviada para varias variables

Si por ejemplo hay 7 variables con el mismo formato, es útil la notación

input (var1-var7) (formato).

Esto es práctico si se desean leer muchas variables de la misma longitud.

Ejemplo: notación abreviada en lectura con formato

```
data uno;
  input (x1-x3) (2.);
cards;
455678
324512
;
```

Se obtiene el archivo SAS uno:

```
X1   X2   X3
45   56   78
32   45   12
```

Finalmente se abordará la manera de leer datos en los cuales hay varias observaciones por cada valor clave de una variable. Se utilizará la opción input… @; que deja el puntero de lectura preparado en la misma línea después de leer la variable clave.

Ejemplo: lectura de datos con varias observaciones por cada valor de una o varias variables clave

Supongamos que para cada valor de la variable grupo hay que leer 3 observaciones, y el valor de la variable grupo viene seguido de los valores de estas observaciones en los datos en modo texto.

```
data uno;
  input grupo @;
  do i=1 to 3;
     input a b @@;
     output;
  end;
cards;
1 78 43 34 21 2 1
2 33 11 9 7 8 5
;
```

Se obtiene el archivo SAS uno:

```
grupo     a     b

  1      78    43
  1      34    21
  1       2     1
  2      33    11
  2       9     7
  2       8     5
```

Del mismo modo se haría si hubiera varias variables clave, ,simplemente añadiéndolas al primer input.

La lectura con formato permite ahorrar tiempo en cuanto a seleccionar exclusivamente las variables que se desean leer, como se verá en el siguiente ejemplo. Esta posibilidad no existe en formato libre.

Programación eficiente

Ejemplo: leer solamente las variables necesitadas

No siempre es necesario leer todas las variables si no se van a utilizar posteriornente, pues aparte de consumir memoria al leerlas ocupan espacio en el archivo innecesariamente.

En el siguiente programa se lee exclusivamente la variable edad, a pesar de haber más variables en el archivo.

Es importante darse cuenta de la opción #3, que envía el cursor de lectura a la última línea de cada caso, a pesar de que en esa línea no se lean observaciones, pero es necesario para la correcta lectura de la variable edad en la línea 1 de cada caso.

```
data uno;
  input edad 1-2 #3;
cards;
24H
804 1.75
435643643643
12M
337 1.60
234525252552
15M
384 1.70
457456465465
;
```

Programación eficiente

Ejemplo: guardar los datos en archivos SAS

Si se va a trabajar más de una vez (por ejemplo, se volverá otro día a acceder a los datos) con datos que están inicialmente en modo texto, es mejor guardarlos en un archivo SAS permanente y leerlo posteriormente con la sentencia set, que volver a ejecutar el programa SAS de lectura de datos cada vez.

```
PRIMER DÍA

libname discoc ‘c:\’;
data discoc.uno;
  input edad 1-2 @3 sexo $ /  peso 1-3 .1 altura #3;
cards;
24H
804 1.75
435643643643
12M
337 1.60
234525252552
15M
384 1.70
457456465465
;

...
EN SESIONES SAS POSTERIORES…
…

libname discoc ‘c:\’;
data dos;
  set discoc.uno;
  líneas de programa…
run;
```

Por último se estudiará el modo de crear archivos SAS a partir de archivos de otros formatos distintos del modo texto.

Procedimiento PROC IMPORT

El procedimiento Proc Import está destinado a convertir archivos de formatos externos como EXCEL, DBASE o LOTUS, en archivos SAS. Su sintaxis básica es:

```
PROC IMPORT datafile=”archivo”
out=archivo SAS DBMS=Tipo de archivo;
[getnames=YES|NO];
[sheet=nombre hoja];
[range=”subselección de la hoja”];
run;
```

El archivo a leer puede estar en formato DBASE, LOTUS, EXCEL, etc. El archivo debe ser nombrado entre comillas. La opción DBMS= puede tomar los siguientes valores:

DBMS=DBF, para archivos en DBASE.

DBMS=WK1, WK3 o WK4 para archivos LOTUS según la versión.

DBMS=EXCEL, EXCEL4, EXCEL5, EXCEL97, EXCEL2000 para archivos EXCEL según la versión.

En principio, en el momento de ejecutar el proc import el archivo fuente a leer no puede estar siendo leído a la vez por otros programas por una cuestión de incompatibilidad.

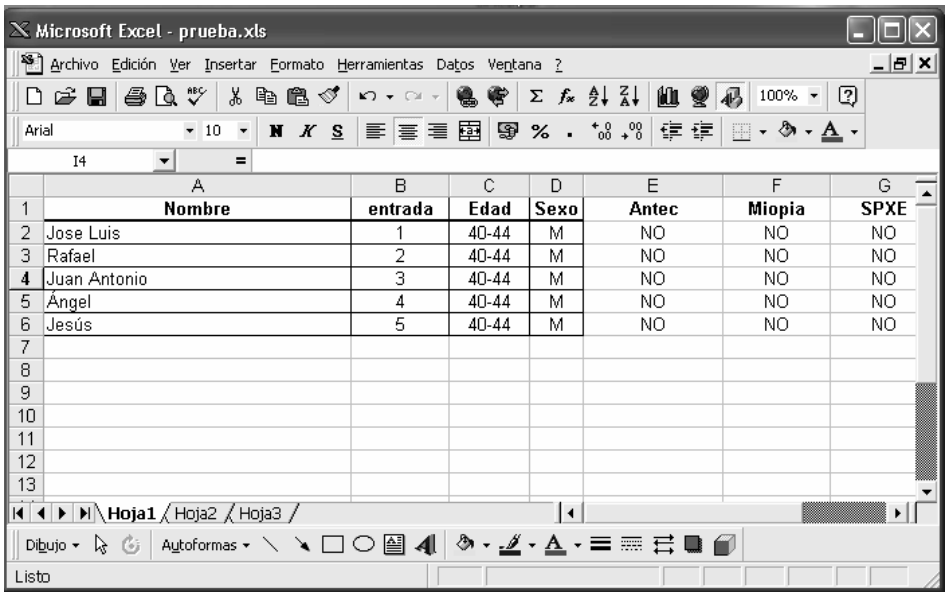
Por defecto, la opción getnames toma el valor YES. La opción getnames=NO indica que los nombres de las variables no están en la primera línea de la hoja de cálculo. En este caso las variables creadas en el archivo SAS se llamarán VAR1, VAR2, ...VARn.

La opción sheet=nombre hoja está destinada a la selección de la hoja de cálculo, cuando se trata de archivos LOTUS o EXCEL.

La opción range=*subselección* permite seleccionar un cuadro de celdas de la hoja. Por defecto el procedimiento lee toda la hoja de cálculo. El modo de nombrar las celdas es utilizando la notación *”celda-inicial:celda final”*. El procedimiento importa las variables y observaciones comprendidas en el rectángulo delimitado entre la celda inicial (situada arriba a la izquierda) y la celda final (situada abajo a la derecha).

Ejemplo: Lectura de un archivo EXCEL97

Supongamos que el archivo Excel97 a leer toma esta forma:



	A	B	C	D	E	F	G
1	**Nombre**	**entrada**	**Edad**	**Sexo**	**Antec**	**Miopia**	**SPXE**
2	Jose Luis	1	40-44	M	NO	NO	NO
3	Rafael	2	40-44	M	NO	NO	NO
4	Juan Antonio	3	40-44	M	NO	NO	NO
5	Angel	4	40-44	M	NO	NO	NO
6	Jesús	5	40-44	M	NO	NO	NO
7							
8							
9							
10							
11							
12							
13							

El programa para crear un archivo SAS temporal con las 7 variables y sólo las observaciones de 4 a 6 será:

```
PROC IMPORT datafile=”prueba.xls”
out=uno DBMS=EXCEL97;
getnames=YES;
sheet=Hoja1;
range=A4:G6;
run;
```

El archivo SAS uno tiene 7 variables y 3 observaciones:

```
NOMBRE          ENTRADA    EDAD       SEXO  ANTEC      MIOPIA    SPXE
Juan Antonio    3          40-44      M     NO         NO          NO
Angel           4          40-44      M     NO         NO          NO
Jesus           5          40-44      M     NO         NO          NO
```

Capítulo 4. Caracter iterativo del bloque DATA

El modo de funcionamiento del paso data cuando se leen observaciones, es la lectura secuencial de datos, a modo de bucle de programación. Conocer este funcionamiento iterativo es necesario para comprender gran parte de los problemas que se dan en la programación básica y avanzada con el SAS.

Las sentencias del bloque data INPUT, SET y MERGE leen los casos uno por uno, realizando con cada uno de ellos las operaciones intermedias que existan en el bloque data.

Es importante apuntar que durante el proceso del paso data, en cada momento lógico solamente existe en memoria una observación, es decir, los valores de todas las variables no missing para esa observación. Estas observaciones se van añadiendo iterativamente, una por una, al archivo creado en el paso data.

El proceso es el siguiente:

1) Inmediatamente tras la sentencia data...; se produce la actualización de la variable de sistema _n_. Esta variable toma valor 1 en la primera observación, y aumenta de valor en una unidad cada vez que el proceso iterativo pasa por la sentencia data...;.

2) A continuación se aplican las posibles sentencias de tipo ejecutable anteriores a las posibles sentencias de lectura (no es obligatorio que un paso data contenga sentencias de lectura de datos como input, set o merge; el paso data puede simplemente estar destinado a la programación per se, sin lectura de datos).

3) En la sentencia de lectura (input, set o merge) se produce la lectura y paso a memoria temporal de la observación en curso (es decir, el paso data mantiene en memoria una sola observación a la vez). El puntero de lectura, sea en archivos de texto o archivos SAS, pasa a ocupar la posición de lectura de la siguiente observación.

4) Si el puntero de lectura ha llegado al final del archivo, es decir, si se encuentra en este momento con que no hay más observaciones a leer, se detiene el flujo del programa y finaliza el paso data.

5) Si existen otras sentencias de programación tras las sentencias de lectura, se aplican.

6) La grabación de la observación en el archivo nombrado en la sentencia data se produce:

- o bien al llegar a alguna sentencia "**output** *archivo;* " , si existe esta sentencia en el bloque data,
- o bien al llegar a una sentencia que marque el fin del paso data. Estas sentencias de fin de paso data pueden ser:

  cards;

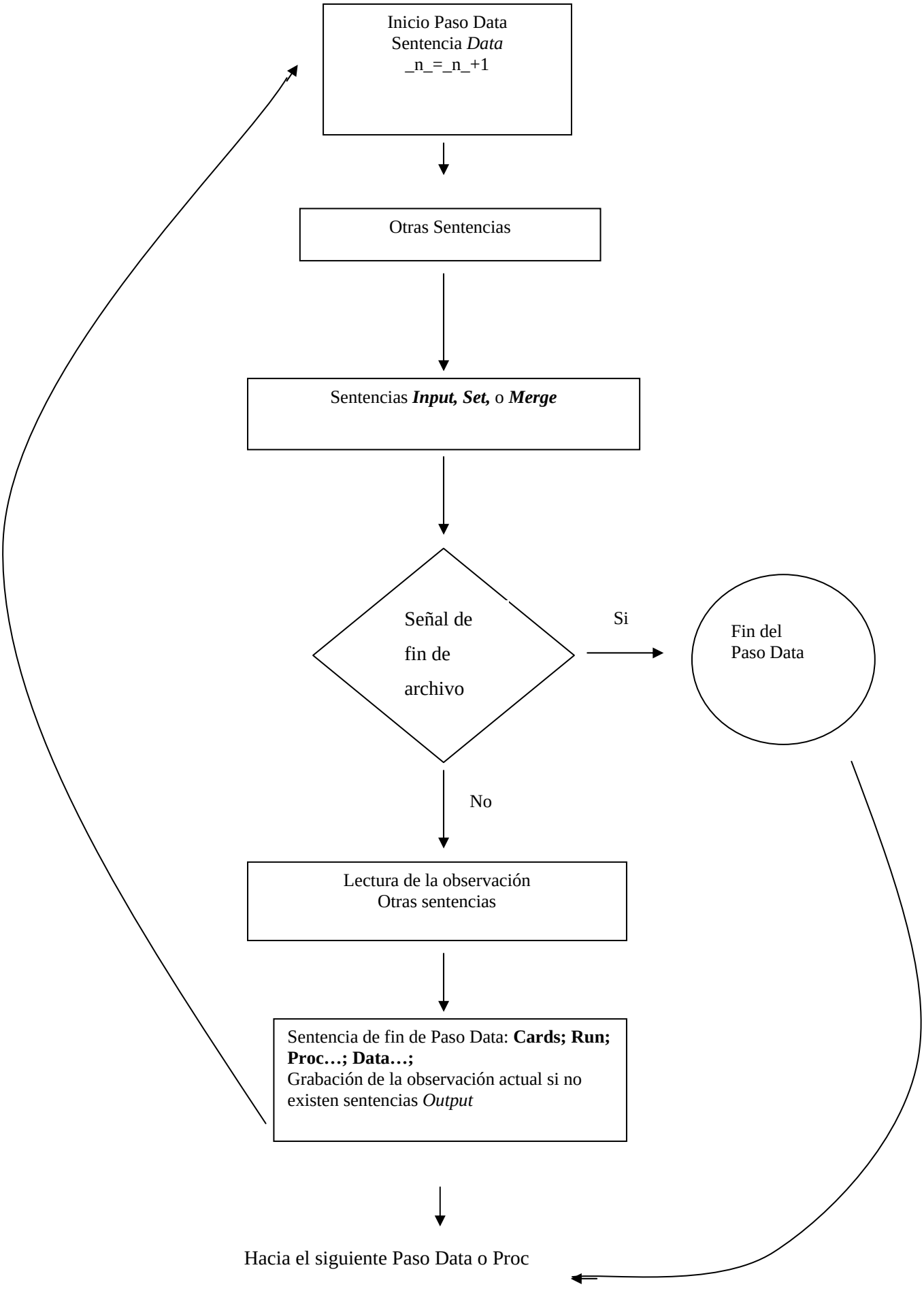
  run;

  proc...;

  data...;

7) En caso de que se haya llegado a alguna sentencia de fin de paso data, el control del programa vuelve a la línea posterior a la sentencia data, actualizando la variable de sistema _n_=_n_+1 , y se repiten los pasos **2)** a **6)**.

Esta secuencia de funcionamiento por defecto se puede ilustrar con un diagrama.


Inicio Paso Data
Sentencia *Data*
_n_=_n_+1
Otras Sentencias
Sentencias ***Input, Set,*** o ***Merge***
Señal de fin de archivo
Si
Fin del Paso Data
No
Lectura de la observación
Otras sentencias
Sentencia de fin de Paso Data: **Cards; Run; Proc…; Data…;**
Grabación de la observación actual si no existen sentencias *Output*
Hacia el siguiente Paso Data o Proc

Ejemplo: funcionamiento iterativo del paso data

```
data uno;
   put 'linea siguiente a data';
   put 'Observación nº' _N_ a= b=;
   input a b @@;
   put 'linea siguiente a input';
   put 'Observación nº' _N_ a= b=;
cards;
4 5 6 7 8 9
;
```

Pondría en la ventana LOG:

```
linea siguiente a data
Observación nº1 a=. b=.
linea siguiente a input
Observación nº1 a=4 b=5
linea siguiente a data
Observación nº2 a=. b=.
linea siguiente a input
Observación nº2 a=6 b=7
linea siguiente a data
Observación nº3 a=. b=.
linea siguiente a input
Observación nº3 a=8 b=9
linea siguiente a data
Observación nº4 a=. b=.
```

Y se grabaría el archivo SAS *uno* con las siguientes observaciones:

```
a b

4 5
6 7
8 9
```

En el ejemplo expuesto se ha utilizado la sentencia **put,** que escribe texto en la ventana LOG, para controlar el flujo del programa.

Se observa que el proceso de lectua del paso data “olvida” o pone a missing los valores de las variables leídas con input cuando el flujo de control pasa a la línea posterior a data, por ello aparecen los valores a=. y b=. En programación de pasos data en SAS, es fundamental controlar cuáles son los valores de las variables en cada momento del proceso, pues se suelen cometer muchos errores por no controlar estos valores.

Otro detalle que se puede observar en el programa es que el paso data finaliza exactamente cuando el puntero de lectura de la sentencia input llega al final del fichero que se está leyendo (o del texto que se está leyendo, pues en este caso se leen los datos del texto posterior a la sentencia cards).

Sentencias que pueden alterar el proceso iterativo por defecto del paso data

A continuación se estudiarán ciertas sentencias de importancia en el control del paso data, pues a menudo interesa alterar el proceso por defecto, guardando observaciones en los momentos apropiados o conservando los valores de las variables leídas con input de una iteración a otra.

Sentencia OUTPUT;

A veces nos puede interesar, en un mismo paso data, crear más de un archivo SAS, o controlar el momento exacto en que una observación va a ser grabada en el archivo de salida. Para ello se utiliza la sentencia

`OUTPUT` *`archivo`*`;`

en el momento en que se desee que la observación actual sea grabada en el archivo nombrado.

La sentencia OUTPUT significa “guarda la observación que está actualmente en la memoria del paso data, en el archivo nombrado”.

Ejemplo: creación de dos archivos SAS diferentes en un mismo paso data

```
data uno dos;
  input a b @@;
  if a>b then output uno;else output dos;
cards;
4 5 7 6 8 9
;
```

Crearía los archivos

```
uno

a b
7 6

dos

a b
4 5
8 9
```

Funcionamiento por defecto de la sentencia OUTPUT

- La sentencia OUTPUT solamente se puede referir a archivos nombrados previamente en la

sentencia data.

- Si no hay ninguna sentencia OUTPUT en el paso data, la grabación de la actual observación en el **primer** archivo nombrado en la sentencia DATA se produce al llegar al fin del paso data, determinado por otra sentencia DATA, CARDS, PROC , RUN, o el resto del editor vacío.

- Si hay alguna sentencia OUTPUT en el paso data ya no se producen grabaciones por defecto en el final del paso data.

- Si en la sentencia data hay dos (o más) archivos nombrados, y hay una sentencia OUTPUT nombrada en el paso data, no se produce grabación en los otros archivos.

- Si en la sentencia data hay dos (o más) archivos nombrados, y no hay ninguna sentencia OUTPUT en el paso data, se produce la grabación de las observaciones en su lugar por defecto (en la sentencia de final de paso data) en el primer archivo nombrado en la sentencia data, y no se guarda ninguna observación en el segundo archivo o posteriores.

Sentencia RETAIN;

A menudo se desea conservar los valores de las variables leídas con input de una iteración a otra dentro del paso data, evitando que sean puestas a missing debido al funcionamiento por defecto del paso data.

```
RETAIN variable valor;
```

Hace que la var. tome el *valor* en la primera iteración del paso DATA, y luego retenga el valor que la variable vaya tomando de una iteración a otra. Otras opciones de xintaxis de esta sentencia son:

```
RETAIN var;
```

Hace que la var. tome el valor missing sólo en la primera iteración y después vaya guardando los valores que tome.

```
RETAIN lista variables;
```

Como en el caso anterior, pero para todas las variables con que se trabaje.

```
RETAIN lista variables valor;
```

Todas las variables toman todas el mismo valor inicial valor y después guardan los valores de una iteración a otra.

```
RETAIN var1 valor1 var2 valor2 ...;
```

Las variables toman respectivamente los valores iniciales y después guardan los valores de iteración a otra.

Ejemplo: las variables son missing inicialmente

Queremos hallar la suma de los valores de una variable. Sin embargo:

```
data;
  input b;
  suma=suma+b;
  put suma=;
cards;
3 4 5
;
```

da en la ventana LOG:

```
suma=.
suma=.
suma=.
```

pues la variable suma no está inicializada inicialmente y las operaciones con valores missing dan como resultado valor missing.

De igual modo, no sirve utilizar la sentencia suma=0; inicialmente pues el paso data pasa todas las veces por esa sentencia, volviendo suma=0 cada vez, y no acumulando los valores anteriores.

Ejemplo: cálculo de la suma de observaciones leídas con input

```
data;
  retain suma 0;
  input b;
  suma=suma+b;
  put suma=;
cards;
3 4 5
;
pondría en la ventana LOG:

suma=3
suma=7
suma=12
```

Ejemplo: cálculo del mínimo de observaciones leídas con input

Hallar el mínimo de los valores de la variable b .

```
data minimo;
  retain m ;              (m toma el valor inicial missing)
  input b @@;
  m=min(b,m);
  put m=;
cards;
2 5 7 8 -1
;
da:
M=2
M=2
M=2
M=2
M=-1
```

En el ejemplo anterior se ha utilizado el hecho de que si m es missing, cualquier valor x no missing cumple que la función min(x,m)=x.

Sentencia IF expresion ;

Esta sentencia continúa procesando la observación leída si cumple la condición expuesta en la expresión lógica.

IF *expresion lógica*;

donde expresion puede ser cualquier expresion lógica, pudiendo contener funciones SAS.

Ejemplo: Selección de observaciones con IF

```
data uno;
  input x;
  if x>6
cards;
2 5 7 8 -1
;
```

Se obtiene como resultado el archivo uno:

```
x
7
8
```

Sentencia WHERE expresión;

WHERE expresion lógica;

donde expresion puede ser cualquier expresion lógica, pudiendo contener funciones SAS.

Funciona como la sentencia IF expresion; , salvo que WHERE sólo deja leer las observaciones que cumplen su condición, mientras que IF deja leer la observación y para su proceso si no cumple la condición.

WHERE es la primera sentencia ejecutada después de SET ,MERGE. No funciona con INPUT(mientras que IF ; sí.).

Ejemplo: selección de observaciones con Where

Tenemos los datos de varios individuos en el archivo uno:

```
data uno;
input mes $ ingresos;
cards;
julio 10000
agosto 13000
septiembre 15000
mayo 20200
junio 8800
;
```

Y queremos conservar en otro archivo los ingresos cuyo logaritmo sea mayor de 5, en los meses julio, agosto y septiembre.

```
data dos;
  set uno;
  where mes in ('julio','agosto','septiembre');
  if log(ingresos)>5;
run;
```

Capítulo 5. Lectura y combinación de datos SAS.

En este tema se estudiará cómo trabajar con archivos SAS ya creados, transformándoloes, uniéndolos y combinándolos de diferentes maneras.

Sentencia SET

La sentencia SET se utiliza para leer, dentro de un paso DATA, datos que **ya** están en un **archivo SAS** . Estos archivos SAS contienen la definición de casos y variables, y por lo tanto no necesitan las especificaciones de línea y columna que se dan en la sentencia INPUT, sentencia utilizada exclusivamente para leer datos en modo texto.

Sintaxis básica

La sintaxis básica de la sentencia SET es:

```
SET archivo SAS;
```

La sentencia SET lee los datos de un archivo SAS uno por uno, siguiendo dentro del paso DATA el mismo proceso iterativo que la sentencia INPUT. La diferencia en el proceso es que la sentencia RETAIN no tiene efecto sobre las variables leídas con SET, pues la sentencia

SET conserva los valores de las variables leídas del archivo SAS, sin asignarles valor missing.

Sin embargo, si se crean variables nuevas en el paso data, éstas sí toman valor missing en la línea posterior a la sentencia data.

Ejemplo: duplicación de un archivo SAS

```
data uno;
  set matriz;
run;
```

Crearía el archivo SAS temporal *uno*, idéntico al archivo SAS *matriz*.

```
libname discoc 'c:\';
data discoc.uno;
  set matriz;
run;
```

Crearía el archivo SAS permanente *discoc.uno*, idéntico al archivo SAS *matriz*. En el sistema operativo el archivo se llama uno.sas7bdat.

Opción point=i

La opción POINT de la sentencia SET puede ser útil en ciertas ocasiones.

```
SET archivo POINT=i ;
```

trae a memoria sólo la Observación nº **i** del archivo en cuestión.

En algunas circunstancias interesa leer algunas observaciones selectivamente, utilizando como control su orden o situación en el archivo. Dándole valores a **i** iremos leyendo las diferentes observaciones que nos interesan. La sintaxis del bucle do;…;end; se estudiará más adelante.

Es importante saber que cuando se utiliza la sentencia set… point… es necesario añadir la sentencia *stop* de fin de paso data, fuera del bucle donde habitualmente se utiliza set point. La razón es la detención del proceso por defecto del paso data (el retorno a la línea posterior a data), que lleva a un bucle infinito cuando se utiliza set point.

Con la sentencia stop se obliga al paso data a detener su proceso, una vez que las observaciones han sido leídas con set point. La sentencia stop no es necesaria utilizando el modo usual de lectura con set (solamente con set…point…), puesto que el proceso del paso data se detiene automáticamente cuando se llega al final del archivo leído.

Ejemplo: creación de un archivo SAS con observaciones seleccionadas de otro

Para este ejemplo, se crea artificialmente el archivo datos:

```
data datos;
do i=1 to 100;
 x=ranuni(0);
output;
end;
run;
```

A continuación, el programa

```
data uno;
do i=1,3,6;
   set datos point=i;
   output;
end;
stop;
run;
```

guarda en el archivo *uno* exactamente las observaciones 1,3 y 6 del archivo *datos*.

Y el programa

```
data uno;
do i=1 to 25,40 to 100;
   set datos point=i;
   output;
end;
stop;
run;
```

Guarda en el archivo *uno* exactamente las observaciones de la 1 a la 25 y de la 40 a la 100 del archivo *datos*.

Opciones nobs=variable y end=variable

La opción nobs=*variable* de la sentencia SET es utilizada en muchos programas SAS.

```
SET archivo nobs=variable;
```

La variable nombrada toma como valor constante el número de observaciones del archivo leído. Sin embargo esta nueva variable especial no es guardada por defecto en el archivo de salida del paso data. Si se desea guardarla es necesario crear otra variable en el paso data e igualarla a la primera.

Otra opción de la sentencia SET es la creación de una variable que toma valor 1 si se ha

leído la última observación y 0 si no. La variable se crea en la opción end=variable:

```
SET archivo end=variable;
```

Ejemplo: controlar el número de observaciones del archivo leído

```
data dos;
  set uno nobs=nume;
  put nume=;
  nume2=nume;
run;
```

escribe en la ventana LOG:

```
nume=100
nume=100
nume=100
```
…

El archivo creado dos contiene las variables originales del archivo uno y además la variable nume2, con valor constante 100 para todas las observaciones.

En el ejemplo siguiente se utiliza la función MOD para seleccionar cada 5 observaciones de un archivo SAS (es decir, se seleccionan las observaciones 5, 10, 15, 20… del archivo datos y se guardan en datos2).

Ejemplo: selección de cada *k* observaciones de un archivo SAS

```
data datos2;
   set datos;
   if MOD(_n_+1,5)=1 then output;
run;
```

Guarda en el archivo *datos2* las observaciones 5,10,…del archivo *datos*.

Sintaxis para la unión secuencial de archivos

```
SET archivo1 archivo2…archivon;
```

Une en serie los datos de varios archivos SAS. El proceso que sigue es el siguiente:

Va leyendo las observaciones del *archivo1* iterativamente, hasta llegar a la última. A continuación, lee del mismo modo las observaciones del *archivo2*. El archivo SAS de la sentencia DATA que se va creando contiene todas las observaciones del *archivo1* y todas las del *archivo 2*, por ese orden.

Ejemplo: unión secuencial de dos archivos SAS

Si tenemos los archivos SAS temporales *uno* y *dos*:

```
uno             dos

a  b            b    c

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

El programa

```
data tres;
set uno dos;
run;
```

crea el archivo *tres:*

```
a  b  c

2  1  .
3  4  .
.  6  9
.  5  1
.  2  7
```

En el ejemplo anterior se observa que si en algún archivo no existen la variables del otro, simplemente los valores de estas variables son puestos a missing para las observaciones del primer archivo.

Sintaxis para la unión en paralelo de archivos

```
SET archivo1;
SET archivo2;
```

…

Une en paralelo los datos de varios archivos SAS. El proceso que sigue consiste en:

1) Lee del *archivo1* la primera observación; a continuación lee del *archivo2* la primera observación. Continúa con el paso data.

2) Continúa leyendo iterativamente todas las observaciones hasta llegar al **final del archivo con menos observaciones.** Esto es debido al funcionamento por defecto del paso data, que se detiene en cuanto algún puntero de lectura (en este caso están los punteros de lectura de cada una de las sentencias set referidas a diferentes archivos) llega a la marca de final de archivo.

Ejemplo: unión en paralelo de archivos SAS con set

Si tenemos los archivos SAS temporales *uno* y *dos*:

```
uno     dos

a  b    b  c

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

El programa

```
data tres;
  set uno;
  set dos;
run;
```

crea el archivo *tres:*

```
a b  c
2 6  9
3 5  1
```

Se observa que el archivo resultante sólo tiene dos observaciones. Esto es debido a que el archivo con menos observaciones es el *uno*.

El orden con que se nombren los archivos tiene importancia si hay variables coincidentes: La variable b va tomando los valores del último archivo nombrado en la secuencia **set** uno; **set** dos. Esto es debido al funcionamiento por defecto del paso data, que sólo mantiene en memoria los valores de una observación a la vez.

Es decir, si el valor de b en una observación leída con "set uno;" es 3, y a continuación es leído el valor de b con la sentencia "set dos;", que es b=5, el valor b=3 anterior es olvidado y reemplazado por b=5 en la memoria instantánea del paso data.

Otra característica de la sentencia set es que las variables leídas con set son conservadas, es decir no son puestas a missing al pasar el control del paso data a la sentencia posterior a data...; como sí ocurre con la sentencia input. Esto se utiliza en la aplicación que se verá a continuación.

La mayor utilidad del formato de unión *set archivo1;set archivo2;* consiste en la siguiente aplicación:

Supongamos que tenemos dos archivos de los cuales el archivo1 contiene una sola observación, cuyos valores queremos conservar a lo largo de todo el proceso del paso data, y el archivo2 contiene varias observaciones que se irán leyendo. La mejor solución (la más sencilla) para unir estos dos archivos es el siguiente programa:

Ejemplo: unión de una observación que está en un archivo con todas las observaciones de otro archivo

El archivo uno contiene la variable a, con valor 1 y una única observación:

```
a
1
```

El archivo dos es:

```
b  c
3  2
4  5
7  8
```

Supongamos que se desea ir creando la variable z=b+c-a en el archivo tres, donde el valor de a es el que aparece en la primera observación del archivo uno (en este caso el valor a=1). Para ello es necesario tener la información de la variable a y conservarla desde la lectura de la primera observación. Esto se hace con la sentencia

```
if _n_=1 then set uno;
```

El programa sería el siguiente:

```
data tres;
  if _n_ =1 then set uno;
  set dos;
  z=b+c-a;
run;

El archivo tres creado es:

a b  c  z
1 3  2  4
1 4  5  8
1 7  8  14
```

Como regla general, el formato *set uno dos;* se utilizará para unir archivos con las mismas variables y distintos casos. El formato *set uno; set dos;* se utilizará para unir archivos con los mismos casos, y distintas variables, si se desea que se conserve el mínimo número de observaciones entre los dos archivos.

La utilidad principal de este último formato radica en la unión de dos archivos en los que uno contiene una única observación (es decir, el ejemplo anterior) , pues para uniones en paralelo se utilizará más habitualmente la sentencia MERGE.

Sentencia MERGE

```
MERGE archivo1 archivo2;
```

se comporta de manera similar a

```
SET archivo1;
SET archivo2;
```

salvo que no se detiene en la observación final del archivo con menos observaciones, sino que pone valores missing en las variables del archivo con menos observaciones no presentes en el otro archivo.

Ejemplo: unión en paralelo de archivos Sas con MERGE

```
data uno;
  input a;
cards;
3
;
data dos;
  input b;
cards;
4
5
;

data tres;
  merge uno dos;
run;
```

da lugar a:

```
tres

a b
3 4
. 5
```

mientras que

```
data tres;
  set uno;
  set dos;
run;
```

da lugar a:

```
tres

a b
3 4
```

A veces, el orden observación a observación de los dos archivos no coincide. En estos casos, si existe una variable común en los dos archivos, para que MERGE sea eficaz es

necesario:

a) Reordenar todos los archivos de origen por las variables comunes con PROC SORT DATA=*archivo1* BY *var1 var2...*

b) Utilizar la notación MERGE *archivo1 archivo2…*;BY *var1 var2...*

Este tipo de unión es usual a la hora de trabajar con paquetes estadísticos. Si en el archivo *uno* hay una observación por cada categoría de la variable *clave*, aparte de variables que representan identificaciones o cualidades de cada categoría, y en el archivo *dos* hay una población de elementos con características individuales entre las cuales está la variable *clave*, la unión de ambos archivos debe hacerse de modo que se conserve la información del primer archivo para cada categoría.

Ejemplo: unión paralela de archivos por variable clave

```
data persona;
  input nombre $ sexo $;
cards;
maria f
ana f
tomas m
;
data lugar;
  input nombre $ ciudad $ region;
cards;
jose alava 5
maria malaga 2
maria orense 7
ana orense 6
;
```

La utilización de MERGE:

```
proc sort data=persona;by nombre;
proc sort data=lugar;by nombre;
data datos;
  merge persona lugar;
  by nombre;
run;
```

da lugar al archivo datos, correctamente creado:

```
OBS     NOMBRE   SEXO  CIUDAD    REGION
1       ana         f  orense     6
2       jose        .  alava      5
3       maria       f  malaga     2
4       maria       f  orense     6
5       tomas       m  .          .
```

Opciones de lectura de archivos sas

Las siguientes opciones se utilizan en las sentencias DATA, SET y MERGE, entre paréntesis tras el nombre del archivo SAS. Son opciones fundamentales para reducir el tiempo de proceso y gestionar adecuadamente los programas:

`DROP=variables`

No graba o lee del archivo nombrado las variables mencionadas. De esta manera se ahorra memoria.

`KEEP=variables`

Sólo graba o lee del archivo nombrado las variables mencionadas

`FIRSTOBS=n`

Primera Observación a leer (opción solo disponible en SET y MERGE).

`OBS=n`

Última Observación a leer (opción solo disponible en SET y MERGE).

`WHERE=(condición lógica)`

Sólo lee o guarda las observaciones que cumplen la condición. Esta opción también está disponible en los procedimientos SAS.

`IN=var`

Crea una variable cuyo valor es 1 si la Observación pertenece al archivo nombrado y 0 si no (no se utiliza en la sentencia data, solo en set o merge).

Ejemplo: selección básica de observaciones y variables en archivos SAS

El archivo SAS *matriz* contiene 15 casos de las var. X1,X2,Y1,Y2.

```
data uno(keep=X1 X2 firstobs=4) dos (drop=X2 obs=10);
  set matriz (drop=Y2);
run;
```

El archivo *uno* contiene las observaciones de la 4 a la 15 de las variables X1 y X2 del archivo *matriz*. El archivo *dos* contiene las observaciones de la 1 a la 10 de las variables X1 y Y1 del archivo *matriz*.

En general, si se puede elegir entre poner opciones en la sentencia data y la sentencia set, será más eficiente poner las opciones firstobs, obs, drop y keep en la sentencia set, para ahorrar en proceso de lectura.

La opción where permite de una manera sencilla seleccionar observaciones que cumplan ciertas condiciones.

Ejemplo: selección condicional de observaciones

```
data matriz;
   input x z nombre $;
cards;
3 6 pepe
8 3 maria
9 5 JUAN
;
data uno;
   set matriz (where=(x>7 and (0<z<6 or nombre='JUAN')));
run;
proc print;run;

x    z    nombre

8    3    maria
9    5    JUAN
```

La opción IN=*variable* también es de cierta utilidad para controlar observaciones leídas. A menudo, cuando se unen dos archivos SAS con merge, solo se desea unir aquellas observaciones que existen en los dos archivos, es decir, que tienen el mismo valor en la variable clave en los dos archivos. Aquellas observaciones que están en un archivo y no en el otro, en el sentido de la variable clave de identificación, no serían guardadas. El siguiente ejemplo ilustra como realizar una unión de archivos SAS con esta condición.

Ejemplo: unión de archivos por palabra clave, con selección de observaciones

```
data persona;
  input nombre $ sexo $;
cards;
maria f
ana f
tomas m
;
data lugar;
  input nombre $ ciudad $ region;
cards;
jose alava 5
maria malaga 2
maria orense 7
ana orense 6;
```

La utilización de MERGE:

```
proc sort data=persona;by nombre;
proc sort data=lugar;by nombre;
```

```
data datos;
  merge persona (in=per) lugar (in=lug);
  by nombre;
run;
```

da lugar al archivo datos:

```
NOMBRE  SEXO     CIUDAD    REGION
ana     f        orense     6
maria   f        malaga     2
maria   f        orense     6
```

Opción BY. Lectura y proceso por categorías

Si utilizamos la sentencia SET con la opción BY *var*, donde *var* es una lista de variables en principio categóricas, las observaciones son leidas por orden de esas categorías.

Además, son creadas dos importantes variables de sistema: FIRST.*var* y LAST.*var* que toman valores respectivos 1 según estemos en la primera (para FIRST.*var*) o última (para LAST.*var*) observación de cada grupo de la variable categórica. En el resto de observaciones estas variables toman valor 0.

Estas variables no son guardadas como variables SAS, sino que sólo existen en el proceso de lectura del archivo. **Como siempre, si vamos a utilizar la opción BY deberemos ordenar previamente el archivo por las variables del BY.**

Ejemplo: variables que indican comienzo y fin de categoría

El archivo uno ya ordenado es

```
cat edad

1 21
1 20
2 23
2 25
3 24

data dos;
   set uno;
   by cat;
run;
```

En el transcurso de la lectura las variables presentes en memoria son:

```
cat  edad first.cat   last.cat

1    21   1           0
1    20   0           1
2    23   1           0
2    25   0           1
3    24   1           1
```

El archivo dos resultante es exactamente igual al archivo uno:

```
cat edad

1    21
1    20
2    23
2    25
3    24
```

La opción BY también crea estas variables en la sentencia MERGE. Estas variables de sistema son de gran utilidad en aplicaciones, combinadas adecuadamente con sentencias condicionales. A continuación se presentan aplicaciones variadas en uniones de archivos que se dan con cierta frecuencia.

Ejemplo : Extraer todos los valores diferentes de una variable categórica

Si tenemos el siguiente archivo *uno*, donde hay varias observaciones para cada categoría:

```
ciudad    habit1      habit2

Valencia  900000      .
Valencia  600000      700000
Sevilla   600000      ...
...
```

y queremos saber qué ciudades diferentes hay en el archivo, se realiza el programa siguiente:

```
proc sort data=uno;by ciudad;

data dos (keep=ciudad);
  set uno;by ciudad;
  if first.ciudad=1 then output;
run;

proc print data=dos;
run;
```

Ejemplo: Localizar categorías con un solo elemento.

Si en el archivo *uno* anterior necesitamos localizar qué ciudades constan de una sola observación, basta poner una condición simple:

```
proc sort data=uno;by ciudad;
data dos (keep=ciudad);
  set uno;by ciudad;
  if (first.ciudad=1 and last.ciudad=1) then output;
run;
proc print data=dos;
run;
```

El programa anterior presenta en la ventana OUTPUT un listado de las categorías con un solo elemento.

Ejemplo: Localizar observaciones duplicadas con set…;by…;

Si en el archivo *uno* la ciudad es como una variable de identificación y queremos localizar qué ciudades están de alguna manera duplicadas en el archivo (en el sentido de tener el mismo valor en la variable CIUDAD), y guardar las duplicadas en un archivo, se puede utilizar el siguiente programa.

El archivo *uno* es

```
CIUDAD    CODIGO      OBS

Valencia  4           1
Orense    6           2
Madrid    7           3
Madrid    7           4
Barcelona 3           5
```

```
proc sort data=uno;by ciudad;
data unicas duplicas;
  set uno;by ciudad;
  if (first.ciudad=1 and last.ciudad=1) then output unicas;
  else output duplicas;
proc print data=unicas;
proc print data=duplicas;
run;
```

da lugar en la ventana OUTPUT a un listado de las observaciones no duplicadas:

```
CIUDAD    CODIGO      OBS

Valencia  4           1
Orense    6           2
Barcelona 3           5
```

y a un listado de las duplicadas:

```
CIUDAD    CODIGO    OBS

Madrid       7       3
Madrid       7       4
```

Ejemplo: Borrar observaciones duplicadas con proc sort;

Si simplemente se desea eliminar observaciones duplicadas, se puede realizar simplemente con la opción NODUPKEY del procedimiento SORT :

```
proc sort data=uno out=nodupli nodupkey;
by ciudad;
run;
```

Crea el archivo nodupli donde las observaciones con valor repetido en la variable ciudad no están (solo aparece la primera observación con ciudad=MADRID):

```
CIUDAD    CODIGO      OBS

Valencia  4           1
Orense    6           2
Madrid    7           3
Barcelona 3           5
```

Dentro del paso data se puede controlar el proceso de computación y programación utilizando la sentencia put, que escribe información en la ventana LOG o en un archivo de texto, para verificar los valores de las variables en cada momento del paso data. Además esta sentencia permite la creación de informes concretos muy controlados por el programador.

Sentencia PUT

Escribe por defecto los valores de las variables en la ventana LOG. Utiliza la misma sintaxis que INPUT. Frecuentemente se utiliza con símbolos de control de cursor.

Ejemplo: sintaxis de la sentencia put

```
data uno;
  input a b c @@;
  put @3 a / b +4 c;
cards;
3 4 5 1 2 3
;
```

pondría en el LOG:

```
  3
4    5
  1
2    3
```

Por otra parte, la sentencia

```
put a= b= c=;
```

pondría en LOG también el signo igual =:

```
a=3 b=4 c=5
a=1 b=2 c=3
```

```
put 'este es el valor de a:' a;
```

pondría:

```
este es el valor de a: 3
este es el valor de a: 1
```

Para escribir en un archivo en modo texto, hay que utilizar la sentencia FILE:

```
FILE ‘archivo de texto’;
```

escribe la salida de la instrucción PUT en el archivo de texto nombrado.

Ejemplo: escritura de datos en un archivo de texto

```
data uno;
  input a b c @@;
  file ‘c:\datos.txt’;
  put @3 a / b +4 c;
cards;
3 4 5 1 2 3
;
```

crearía el archivo de texto ‘c:\datos.txt’ con el texto:

```
  3
4    5
  1
2    3
```

La sentencia PUT siempre escribe en el fichero nombrado en la sentencia FILE anterior más cercana. Por ejemplo, si ponemos:

```
file 'b:datos1.txt';
```

```
put a= b= c=;
file 'b:datos2.txt';
put d= e= f=;
```

los valores de las variables a,b,c son escritos en el archivo 'b:\datos1.txt' y los de las variables d,e,f en 'b:\datos2.txt'. Cada sentencia file lleva asociado un puntero de escritura a cada archivo, con lo cual en cada uno de los dos archivos saltará líneas correctamente en cada observación.

Una utilidad directa de la sentencia put es la creación de un archivo de datos en modo texto a partir de un archivo SAS.

Ejemplo: creación de un archivo de datos en modo texto a partir de un archivo sas

Si se dispone del archivo sas temporal *uno*:

```
CIUDAD    CODIGO      OBS

Valencia  4           1
Orense    6           2
Madrid    7           3
Barcelona 3           5
```

se puede crear el archivo de texto c:\matriz.txt con el siguiente paso data:

```
data;
   set uno;
   file ‘c:\matriz.txt’;
   put ciudad $ 1-12 codigo 13 obs 15;
run;
```

Crearía el archivo de texto c:\matriz.txt con el texto

```
Valencia    4 1
Orense      6 2
Madrid      7 3
Barcelona   3 5
```

La opción FILE PRINT añade a las salidas de los procedimientos de la ventana OUTPUT las salidas de la orden PUT.

Ejemplo: combinar la salida de la ventana OUTPUT y el texto de la sentencia PUT

```
data matriz;
   input b @@;
   file print;
   put b=;
cards;
3 5 7 8
;
```

```
proc means data=matriz;
run;
```

Aparecen en la ventana OUTPUT:

1) Los valores de b en formato b=.

```
b=3
b=5
b=7
b=8
```

2) La tabla de estadísticos descriptivos de la variable b:

```
               Analysis Variable : b

    Número de                        Desviación
observaciones           Media          estándar           Mínimo
-----------------------------------------------------------------
            4       5.7500000       2.2173558       3.0000000
-----------------------------------------------------------------

      Máximo
------------
   8.0000000
------------
```

Opción file 'archivo' mod;

Por defecto la sentencia file sobreescribe el archivo si es un archivbo ya existente. Es importante la opción *mod* de la sentencia FILE para añadir datos al archivo de texto sin sobreescribirlo:

Ejemplo: añadir información a un archivo de texto ya existente

Si se dispone del archivo de texto c:\dat1.txt con los valores de la variable x:

```
4
5
8
```

y se desea añadir información nueva leída del archivo sas *uno*, que contiene nuevos valores de la variable x:

```
data;
   set uno;
   file 'c:\dat1.txt' mod;
   put x;
run;
```

Capítulo 6. Funciones SAS

Las funciones Sas permitirán transformar valores de variables existentes y crear nuevas variables en el paso data. Además son necesarias para cualquier cálculo o programación numérica compleja al margen de las ofrecidas por los procedimientos SAS.

La sintaxis para crear o modificar variables es sencilla. Si se desea cambiar el valor de una variable o crear una nueva, basta poner el signo igual =:

```
a=log(b)+3.4*sin(c);
```

Si se quiere duplicar una variable, creando por ejemplo la variable nueva a idéntica a la variable b:

```
a=b;
```

Para asignar un valor constante a una variable en todas las observaciones del archivo:

```
a=sqrt(8);
```

Es necesario recordar aquí el proceso iterativo del paso data: si por ejemplo se están leyendo observaciones con la sentencia set de un archivo sas existente, cada vez que se lee una observación el puntero de control del paso data llega a la sentencia de asignación

```
a=sqrt(8);
```

y en ese momento se actualiza el valor de a para la observación en memoria. Cuando el puntero de control del paso data llega al momento de guardar la observación , la variable a es guardada con ese valor. Este proceso se repite para todas las observaciones leídas del archivo sas con la sentencia set, de modo que si en ese archivo sas había 100 observaciones, en el archivo sas creado en el paso data habrá las mismas, solo que también estará la variable a, que tomará el valor constante sqrt(8) (raíz cuadrada de 8) para todas las observaciones.

En la Ayuda del programa está la definición y ejemplos de todas las funciones SAS.

Funciones Matemáticas SAS

Las funciones matemáticas SAS operan sobre variables o expresiones que dan lugar a un número. Se aplican a través de la sentencia de asignación, asignando su valor a una variable:

var=f(expresión matemática);

Algunas funciones matemáticas conocidas son ABS(x), COS(x), EXP(x), INT(x), LOG(x), SQRT(x), SIGN(x), SIN(x).

Otras pueden ser, por ejemplo:

GAMMA(x)-. Devuelve la integral (0,inf) de t**(x-1)*exp(-t)dt.

Si x es un entero positivo, GAMMA(x) devuelve (x-1)!:

En general, las sentencias matemáticas operan no solamente sobre una variable, sino sobre cualquier expresión matemática SAS. Por ejemplo, la función LOG(variable) se puede aplicar tanto a una variable concreta como a una expresión:

z=log (ingresos);

z=log(ingresos*123+log(5)-x);

Ejemplo: factorial de un número

```
data facto7;
  y=gamma(8);
  put '7!=' y;
run;
```

tiene como resultado en la ventana LOG:

7!=5040

Funciones estadísticas

Las funciones que se presentan a continuación son funciones que operan sobre varias variables en cada observación.

MIN(x1,x2,...) devuelve el mínimo de los valores de las diferentes **variables** en cada **Observación**:

Ejemplo: hallar el mínimo valor entre variables para cada observación

```
data mini;
  input a b;
  c=min(a,b);
  put 'minimo(a,b)=' c;
cards;
3 4
2 1
;
```

pondría en la ventana LOG:

```
minimo(a,b)= 3
minimo(a,b)= 1
```

y el archivo SAS creado *mini* sería:

```
a b  c
3 4  3
2 1  1
```

MAX(x1,x2,...) devuelve el máximo de los valores de las variables para cada Observación.

MEAN(x1,x2,...) devuelve la media de los valores de las variables para cada Observación.

STD(x1,x2...) devuelve la d.típica de los valores de las variables para cada Observación.

Para estas funciones, cuando las variables tienen nombres del tipo *raiz*1,…,*raiz*n, por ejemplo las variables z1, z2, z3 o ingresos1,ingresos2, etc., se puede utilizar la notación abreviada **función(of *raiz*1-*raiz*n).**

Aunque las variables no tengan estos nombres con subíndices, en el archivo está ordenadas según como se creó inicialmente el archivo SAS. En estos casos se puede utilizar la notación abreviada separando la primera variable de interés y la última con el operador '--'.

Ejemplo: notación abreviada en las funciones estadísticas

```
data mini;
  input x1 x2 x3;
  c=min(of x1-x3);
  put 'minimo=' c;
cards;
3 4 6
2 1 -2
;
```

pondría en la ventana LOG:

```
minimo= 3
minimo= -2

data mini;
```

```
  input a b c;
  c=min(of a--c);
  put 'minimo=' d;
cards;
3 4 6
2 1 -2
;
```

pondría en la ventana LOG:

```
minimo= 3
minimo= -2
```

Funciones de generación de números aleatorios

En estas funciones tiene particular interés la semilla de generación. Esta puede ser:

- Aportada por el usuario. Este modo tiene la ventaja del control, pues cada vez que se ejecuta la misma sentencia de generación con la misma semilla se da lugar al mismo número.
- El valor 0, que toma como semilla el valor del reloj interno del ordenador.

RANBIN(semilla,n,p)-.Genera una Observación de una B(n,p).

RANCAU(semilla)-.Genera una Observación de una Cauchy (0,1).

RANEXP(semilla)-.Genera una Observación de una EXP(1).

RANGAM(semilla,a)-.Genera una Observación de una Gamma(a).

RANNOR(semilla)-.Genera una Observación de una N(0,1).

RANPOI(semilla,lambda)-.Genera una Observación de una P(lambda).

RANUNI(seed)-.Genera una Observación de una U(0,1).

RANTBL(seed,p1,p2,...,pn)-.Genera una Observación de una variable discreta tabulada, que toma valores 1,2,...,n con probabilidades p1,p2,...,pn.

Ejemplo:generación de números aleatorios

```
data uno;
  x=ranuni(0);
  y=rannor(0)*3+5;
  z=
run;
```

Genera el archivo *uno* con una sola observación y las variables x=U(0,1) e y=N(5,3).

Funciones de Probabilidad

POISSON(lambda,x)-.devuelve P(X<=x), donde X=Poisson(lambda).

PROBBETA(x,a,b)-. devuelve P(X=x), donde X=Beta(a,b).

PROBBNML(p,n,x)-. devuelve P(X<=x), donde X=B(n,p).

PROBCHI(x,gl)-. devuelve P(X<=x), donde X= χ^2(gl).

PROBF(x,gln,gld)-. devuelve P(X<=x), donde X=F(gln,gld).

PROBGAM(x,a)-. devuelve P(X<=x), donde X=GAMMA(a).

PROBNORM(x)-. devuelve P(X<=x), donde X=NORMAL(0,1).

PROBIT(p)-. devuelve el percentil p en una Normal(0,1).

PROBT(x,gl)-. devuelve P(X<=x), donde X=t-STUDENT(gl).

Ejemplo: cálculo de probabilidades

Calcular la probabilidad P(X=5) en una Poisson(3) y la probabilidad del intervalo (3,4) en una Normal(2,2).

```
data;
  poi=poisson(3,5)-poisson(3,4);
  nor1=probnorm(sqrt(2));
  nor2=probnorm(1/sqrt(2));
  norm=nor1-nor2;
  put poi= norm=;
run;

Sale en la ventana LOG:

poi=0.10081...  norm=0.1611...
```

Funciones de redondeo numérico

ROUND (variable, unidad de redondeo**)-**.Redondea la variable a la unidad de redondeo.

INT (variable**)-**. Toma la parte entera de la variable.

Ejemplo: redondeo de variables con ROUND e INT

```
data uno;
  input a;
  c1=round (a,10);
  c2=round (a,1);
  c3=round (a,0.1);
  c4=round (a,0.2);
  c5=int(a);
cards;
```

```
23.145
1234.42
;
proc print;run;
```

Se obtiene en la ventana OUTPUT:

```
Obs       a         c1      c2       c3        c4       c5

 1       23.15      20       23      23.1      23.2      23
 2     1234.42    1230    1234    1234.4    1234.4    1234
```

A menudo se necesita que la variable quede truncada por ejemplo, al segundo decimal, respetando el valor que este toma. La función ROUND utiliza el decimal más cercano en cuanto a la unidad de redondeo, y por tanto no tiene por qué respetar el valor del decimal.

El siguiente ejemplo muestra una manera de truncar o redondear los valores de una variable respetando las cifras decimales originales. Se utiliza la función $INT(x*10^n)/ 10^n$, donde n es el número de decimales a los que se quiere redondear la variable.

Ejemplo: otro tipo de redondeo de variables

```
data uno;
  input a;
/* c1=redondeo al primer decimal */
/* c2=redondeo al segundo decimal */
  c1=int(a*10)/10;
  c2=int(a*100)/100;
 cards;
23.145
1234.42
;
proc print;run;
```

Se obtiene en la ventana OUTPUT:

```
Obs       a         c1         c2

 1       23.15      23.1      23.14
 2     1234.42    1234.4    1234.42
```

Funciones de Cadena

COMPRESS(var. de cadena,'caracteres'**)**-. Elimina del valor de la variable los caracteres propuestos.

TRIM(var. de cadena**)**-. Elimina los espacios en blanco del comienzo del valor de la variable.

SCAN(var. de cadena,n,delimitador)-. Devuelve como valor la enésima palabra de la variable de cadena, si esta consta de varias palabras separadas por el caracter delimitador.

Tanto en COMPRESS como en SCAN, los caracteres y delimitadores por defecto son los espacios en blanco.

Ejemplo: manipulación de variables alfanuméricas

```
data uno;
length a $30.;
a='  En un lugar de la Mancha';
x=compress(a);put x=;
y=trim(a);put y=;
z=scan(a,3);put z=;
run;
```

Pondría en el LOG:

```
x=EnunlugardelaMancha
y=En un lugar de la Mancha
z=lugar
```

Funciones de Control de Observaciones Pasadas

LAGn(variable**)**-. Devuelve el valor leído n iteraciones antes en el paso data.

DIFn(variable**)-.**Devuelve el valor var-lagn(var) .

La función LAG puede utilizarse en muchas aplicaciones en pasos data en los que es necesario conservar varios valores anteriores, para evitar utilizaciones complicadas de la sentencia RETAIN.

Ejemplo: utilización de la función LAG

```
data uno;
  input x @@;
  y=lag1(x);
  z=lag2(x);
cards;
1 2 3 4 5 ;
;
```

Crearía el archivo uno siguiente:

```
x y  z
1 .  .
2 1  .
3 2  1
4 3  2
5 4  3
```

Funciones de Conversión

Una operación que hace falta a menudo en paquetes estadísticos es la conversión de variables de cadena a numéricas o viceversa.

PUT(variable,informat**)-**. Pasa la variable de numérica a de cadena . Si la variable es numérica, se lee como de cadena utilizando un formato de lectura de cadena.

INPUT(variable,format**)-**. Pasa la variable de cadena a numérica . Si la variable es de cadena, puede devolver su valor numérico utilizando un formato de escritura numérico.

Ejemplo: conversión de variables de cadena a numéricas y viceversa

```
data uno;
  x='22222';
  y=66666;
  z=put(y,$10.);
  w=input(x,20.);
  put z= w=;
run;

proc contents data=uno;run;

 Lista alfabética de variables y atributos

Núm    Variable    Tipo            Longitud

  4    w           Numérica               8
  1    x           Alfanumérica           5
  2    y           Numérica               8
  3    z           Alfanumérica          10
```

Con el proc contents, que da información sobre los atributos de las variables, se puede observar cómo las nuevas variables z y w son, respectivamente, de cadena y numérica.

Capítulo 7. Sentencias condicionales

Sentencia IF-THEN-ELSE

Este conjunto de sentencias permite ejecutar sentencias bloques de programación bajo condiciones lógicas. La sintaxis es:

IF (expresion lógica) THEN sentencias ejecutables; ELSE sentencias ejecutables;

Si la condición de la expresión es cierta, se ejecutarán las sentencias posteriores a THEN, continuando después el curso del programa, salvo las sentencias del ELSE.

Si la condición es falsa, se ejecutarán las sentencias del ELSE, continuando después el curso del programa. Si no existe una sentencia ELSE, se continuará en la sentencia siguiente.

Operadores lógicos válidos en SAS son (< (menor que), > (mayor que), = (igual), NE (distinto de), <= y >=).

Ejemplo: selección condicional de observaciones

Borrar de la memoria los casos en los que la variable edad toma valores menores que 15:

```
data uno;
  set dos;
```

```
  if edad<15 then delete;
  output;
run;
```

Programación eficiente

Ejemplo: selección condicional utilizando else if

Utilizar la sentencia else if ahorra tiempo de memoria, como se verá en el siguiente ejemplo.

```
data uno dos tres;
  set archivo;
  if nombre='Maria' then output uno;
     else if nombre='Juan' then output dos;
        else if nombre='Pedro' then output tres;
run;
```

Es más eficiente que

```
data uno dos tres;
  set archivo;
  if nombre='Maria' then output uno;
  if nombre='Juan' then output dos;
  if nombre='Pedro' then output tres;
run;
```

Programación eficiente

Ejemplo: Utilizar las condiciones en orden descendiente de probabilidad

En una serie de condiciones mutuamente exclusivas el sistema SAS deja de evaluar condiciones en cuanto encuentra la primera cierta. Por ello situando las condiciones más probables en primer lugar se ahorra tiempo de proceso.

Supongamos que en el archivo datos las edades tempranas son más probables.

```
data uno dos tres;
  set datos;
  if 0<edad<10 then output uno;
     else if 10=<edad<30 then output dos;
        else if 30=<edad<60 then output tres;
run;
```

Es más eficiente que

```
data uno dos tres;
  set datos;
  if 30<edad<60 then output tres;
     else if 10=<edad<30 then output dos;
        else if 0<edad<10 then output uno;
run;
```

Observaciones

1) Se puede utilizar la condición:

if var **in** (val1, val2,...,valn)

donde *var* es una variable y *val1, val2,...,valn* son posibles valores que pueda tomar. La condición es cierta si la variable toma alguno de esos valores, y falsa si no toma ninguno de ellos. Si los valores son alfanuméricos, se ponen entre comillas.

2) La opción

if not (*condición lógica*) **then...**

es de gran utilidad en muchos casos. Ejecuta las sentencias ejecutables subsiguientes si la condición lógica es falsa.

Ejemplo: selección de observaciones con la opción if var in (…)

Conservar los datos de individuos que no sean ingleses ni franceses ni italianos.

```
data datos2;
  set datos;
  if pais in (‘inglat’, ‘francia’, ’italia’) then delete;
run;
```

es más eficiente que :

```
data datos2;
  set datos;
  if pais =‘inglat’
     or pais =‘francia’
     or pais =’italia’
     then delete;
run;
```

Bloques DO; ...;END

Si el conjunto de sentencias ejecutables relativo a la sentencia THEN es grande se puede emplear la sentencia:

DO;

sentencias ejecutables;

END;

Ejemplo: bloque de programación bajo sentencia condicional

```
data dos;
  set uno;
  if edad<15 then
  do;
   edad=edad+7;
    tasa=altura/edad;
  end;
  else
  do;
   edad=edad-5;
   tasa=altura/edad -4;
  end;
run;
```

Sentencia SELECT.

Esta sentencia se puede utilizar cuando se desea utilizar la sentencia if… then… else con muchas variantes “else if” . A menudo ahorra memoria y es un bloque de programación identificable y elegante.

En la siguiente sintaxis, la ‘expresion’ no es una condición, sino una fórmula que da lugar a un valor (numérico o alfanumérico).

SELECT (expresion);

WHEN (expresion) sentencias;

...

OTHERWISE *sentencias;*

END;

Modo de proceder:

La sentencia select es equivalente a utiliar una sentencia if…; con varios else then…; . Se

evalúa la expresión del primer WHEN. Si da el mismo resultado que la expresión del SELECT, se ejecutan las sentencias correspondientes (incluyendo posibles bloques DO;END). Si no, se pasa al siguiente WHEN.

Si en ninguno de los WHEN coincide el valor de su expresion con el de la expresion de SELECT se ejecutan las sentencias del OTHERWISE.

Si en este último caso no se ha puesto OTHERWISE se obtiene un error. Es conveniente, pues, poner siempre OTHERWISE;

Ejemplo: utilización de la sentencia condicional select

Tenemos un archivo con el año, ingresos y gastos de una institución.

Queremos repartir en distintos archivos las observaciones que provienen de años diferentes. El año de las obs. viene en la variable tiempo, en el archivo uno.

```
data anterio period1 period2 period3 period4 period5;
  set uno;
  select (tiempo);
   when (1989) output period1;
   when (1990) output period2;
   when (1991) output period3;
   when (1992) output period4;
   when (1993) output period5;
   otherwise output anterio;
  end;
run;
```

Ejemplo: sentencias select anidadas

Se pueden anidar los SELECT. Si en el ejemplo anterior queremos distribuir también según los meses:

```
data ant per1mes1 per1mes2...;
  set uno;
  select (tiempo);
   when (1989) do;
      select (mes);
       when ('enero') output per1mes1;
       when...
       ...
       otherwise;
      end;
   when...
   ...
   otherwise output ant;
  end;
run;
```

Capítulo 8. Bucles

Sentencia DO;…;END;

Es el bucle de programación habitual, realizando las operaciones ejecutables en el interior del bucle para cada valor diferente de la variable índice, que puede variar según el incremento o de 1 en 1 por defecto.

DO *indice=ini* TO *fin* BY *incremento*;

Sentencias ejecutables.…

END;

indice es una variable SAS (en principio no existe antes, sino que es creada por el propio bucle), mientras que *ini* y *fin* son números o expresiones que dan lugar a números.

Ejemplo: sintaxis básica de la sentencia do;…;end; (1)

```
data;
do i=1 TO 5;
  put 'HOLA';
end;
run;
```

Aparece en la ventana LOG:

```
HOLA
HOLA
HOLA
HOLA
HOLA
```

En el siguiente ejemplo se utiliza la opción del incremento para sumar solamente los números impares.

Ejemplo: sintaxis básica de la sentencia do;…;end; (2)

Si se quieren sumar los números impares del 1 al 13:

```
data;
suma=0;
do i=1 to 13 by 2;
  suma=suma+i;
end;
put suma=;
run;
```

Pone en la ventana LOG:

```
suma=49
```

También se puede utilizar el bucle do;end; para valores puntuales de la variable índice, como se verá en el siguiente ejemplo.

Ejemplo: sintaxis básica de la sentencia do;…;end; (3)

Si se quieren sumar los números 2,5,9,18, se separan los valores por comas:

```
data;
suma=0;
do i=2,5,9,18;
  suma=suma+i;
end;
put suma=;
run;
```

Los valores de la variable índice pueden ser alfanuméricos, como se ve en el siguiente ejemplo.

Ejemplo: sintaxis básica de la sentencia do;…;end; (4)

Si se quieren ejecutar ciertas sentencias para varios valores de la variable alfanumérica mes:

```
data uno;
length mes $ 15;
conta=0;
do mes='enero', 'febrero','abril';
  conta=conta+1;
  put conta mes;
end;
run;
```

Aparece en la ventana LOG:

```
1 enero
2 febrero
3 abril
```

Se pueden combinar formatos para los valores de la variable índice, como se verá en el siguiente ejemplo.

Ejemplo: sintaxis básica de la sentencia do;…;end; (5)

```
data uno;
do cuenta=3 to 5, 20 to 26 by 2;
  output;
end;
run;
```

El archivo uno creado es:

```
Obs    cuenta

 1        3
 2        4
 3        5
 4       20
 5       22
 6       24
 7       26
```

Se pueden cambiar los valores de la var. índice dentro del DO para finalizarlo a la fuerza según ciertas condiciones. El siguiente ejemplo ilustra esta posibilidad.

Hay que notar que es importante en programación SAS, cuando se utilizan bucles, utilizar la opción de la sentencia data drop *variable*, donde *variable* es la variable índice. La razón, aparte de conservación de espacio, memoria y tiempo, es que a menudo se utilizan los mismos nombres para la variable índice (i, j, k,…) y al combinar o leer otros archivos SAS con la misma variable se puede dar lugar a errores difíciles de identificar. En el ejemplo, en la sentencia data se añade esta orden drop:

```
data uno (drop=i);
```

Ejemplo: sintaxis básica de la sentencia do;…;end; (6)

```
data uno (drop=i);
  input x @@;
  put 'leo Observación nº'_N_x=;
  stop=10;
  suma=0;
do i=1 to stop;
   suma=suma+i*x;
   put i= suma=;
   if suma>20 then i=stop;
end;
cards;
3 4 7
;
```

Pone en la ventana LOG:

```
leo obs nº1 x=3
I=1 SUMA=3
I=2 SUMA=9
I=3 SUMA=18
I=4 SUMA=30
leo obs nº2 x=4
I=1 SUMA=4
I=2 SUMA=12
I=3 SUMA=24
leo obs nº3 x=7
I=1 SUMA=7
I=2 SUMA=21
```

Se pueden anidar los bucles, como se verá en el ejemplo siguiente.

Ejemplo: sintaxis básica de la sentencia do;…;end; (7)

```
data uno (drop=i j);
  do i=1 to 3;
   do j=1 to 4;
     output;
   end;
  end;
run;
```

Crearía 12 observaciones en el archivo uno:

```
i=1 j=1
i=1 j=2
...
```

Sentencia DO WHILE (expresion);…;END;

Esta sentencia condicional ejecuta las sentencias interiores al bucle si se cumple la expresión lógica. La sintaxis es:

DO WHILE (*expresión lógica*);

Sentencias ejecutables

END;

donde *expresion lógica* es una condición, y puede contener expresiones matemáticas, los símbolos <,>,<>,<=,>=,AND,OR,etc.

La sentencia realiza las instrucciones anteriores a END; mientras se cumpla la expresion del paréntesis.

Ejemplo: sentencia do while(expresion);…;end;

```
data;
  n=0;
  do while(n < 5);
  put n=;
  n=n+1;
end;
run;
```

Pondría en la ventana LOG:

```
n=0
n=1
n=2
n=3
n=4
```

Sentencia DO UNTIL

DO UNTIL (*expresión lógica*);

Sentencias ejecutables

END;

Ejecuta las instrucciones anteriores a END hasta que se cumpla la expresión.

El ejemplo anterior es equivalente al siguiente:

Ejemplo: sentencia do until(expresion);…;end;

```
data;
  n=0;
  do until(n >= 5);
  put n=;
  n=n+1;
end;
```

Pondría en la ventana LOG:

```
n=0
n=1
n=2
n=3
n=4
```

Se puede combinar el formato iterativo de la sentencia DO con los formatos DO WHILE y DO UNTIL.

Ejemplo: combinación de formatos DO, DO UNTIL, DO UNTIL y DO WHILE

```
data;
  suma=0;
  do i=1 to 10 by .5 while(suma < 8.5);
   suma=suma+i;
   put suma=;
  end;
run;
```

Pone en la ventana LOG:

```
suma=1
suma=2.5
suma=7
suma=10
```

El siguiente ejemplo es un programa sencillo que utiliza los bucles y la sentencia de escritura put para construir una tabla de probabilidades.

Ejemplo: Creación de una tabla probabilística (Tabla de la distribución de Poisson)

```
  data;
   /* CABECERA    */
    put / @7 'TABLA DE PROBABILIDADES ACUMULADAS PARA
POISSON(LAMBDA)';

   do i=1 to 62;
     put '-' @;
   end;
   put;
   put @25 'LAMBDA';
   do i=1 to 62;
     put '-' @;
   end;
   put;
   put 'k' @3 @;
   do i=0.1 to 1.2 by 0.1;
     put i 4.1 +1 @;
   end;
   put;
   do i=1 to 62;
     put '-' @;
   end;

/* CREACION DE LA TABLA */

  do k=0 to 7;
    put / k @;
    do lambda=0.1 to 1.2 by 0.1;
       pro=poisson(lambda,k);
       put pro 4.3 +1 @;
    end;
  end;
run;
```

La tabla resultante aparece en la ventana LOG:

```
      TABLA DE PROBABILIDADES ACUMULADAS PARA POISSON(LAMBDA)
------------------------------------------------------------
                          LAMBDA
------------------------------------------------------------
k  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.0  1.1  1.2
------------------------------------------------------------
0 .905 .819 .741 .670 .607 .549 .497 .449 .407 .368 .333 .301
1 .995 .982 .963 .938 .910 .878 .844 .809 .772 .736 .699 .663
2 1.00 .999 .996 .992 .986 .977 .966 .953 .937 .920 .900 .879
3 1.00 1.00 1.00 .999 .998 .997 .994 .991 .987 .981 .974 .966
4 1.00 1.00 1.00 1.00 1.00 1.00 .999 .999 .998 .996 .995 .992
5 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 .999 .999 .998
6 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
7 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00 1.00
```

Simulación de archivos

Para simular un archivo con valores de variables aleatorias se crean bucles de tamaño igual al número de observaciones del archivo que se quiere crear ; a medida que se generan los valores de las variables, se graban en el archivo de salida con la orden **output**.

Ejemplo: creación de un archivo SAS con varias variables artificiales

Crear un archivo SAS donde estén las variables X1=N(2,2) y X2=Gamma(a,p)=Gamma(2,5), con 20 observaciones:

```
data uno;
do i=1 to 20;
   X1=sqrt(2)*rannor(i)+2;
   X2=5*rangam(i,2);
   output;
end;
run;
```

Nota: Si se hubiera puesto erróneamente el output fuera del bucle, sólo se hubiese grabado en el archivo una observación.

Extracción de muestras aleatorias de archivos SAS

En los siguientes ejemplos se utiliza la sentencia DO para obtener muestras aleatorias de archivos SAS.

Ejemplo: obtención de muestras con reemplazamiento

Para extraer una muestra aleatoria de 10 observaciones del archivo SAS *uno* que tiene 30 observaciones, generamos 10 veces (mediante una uniforme discreta) valores enteros entre 1 y 30. Cada número generado será el número de la observación a leer con set point del archivo en cuestión.

La sentencia de detención del paso data 'stop;' es necesaria para detener el proceso del paso data, una vez obtenida la muestra.

```
data dos;
  do i=1 to 10;
     nume=int(30*ranuni(0));
     set uno point=nume;
     output;
  end;
  stop;
run;
```

Hemos extraído muestras **con reemplazamiento :** Una misma Observación puede ser leída dos veces.

Ejemplo: obtención de muestras sin reemplazamiento (método 1)

El siguiente método sencillo permite extraer muestras aleatorias **sin reemplazamiento** :

Se añade al archivo en cuestión una variable generada (uniforme, por ejemplo).

Se ordena el archivo mediante esta variable (PROC SORT). Esta ordenación es por lo tanto aleatoria.

Se toman las 10 primeras observaciones del archivo. Esta será una muestra aleatoria de tamaño 10 del archivo original.

```
data dos ;
  set uno ;
  nume=int(30*ranuni(0));
run;
proc sort data=dos ;by nume ;
run;
data tres ;
  set dos ;
  if n >10 then stop ;
run ;
```

Este método es sencillo pero requiere ordenar todo el archivo original por una variable continua. Si este archivo es muy grande el proceso puede llevar demasiado tiempo.

El siguiente método es más rápido y extrae n observaciones sin reemplazamiento del archivo original con una probabilidad mínima (controlada por el usuario) de extraer menos de n.

Ejemplo: obtención de muestras sin reemplazamiento (método 2)

El siguiente método sencillo permite extraer muestras aleatorias **sin reemplazamiento** :

Si se desean exactamente, pongamos 320 observaciones, se puede aplicar el método en dos fases, seleccionando aproximadamente 400 observaciones en una primera fase, y después desechar aleatoriamente las observaciones sobrantes. (Debido a que este método se emplea cuando el archivo original es grande (pongamos que en este caso el archivo contiene 10000 observaciones) la ley de los grandes números asegura cierta precisión en el objetivo de 400 observaciones.

```
data alea;
  set origin;
  x=ranuni(0);
  if ranuni(0)<0.04 then output alea;
run;
proc sort data=alea;by x;run;
data alea;
  set alea;
  if _n_>320 then stop;
run;
```

El muestreo sistemático con arranque aleatorio selecciona la primera unidad aleatoriamente y a partir de ésta se escogen de k en k unidades.

Si se desean tomar 200 observaciones del archivo *uno,* que tiene 2000 observaciones, habría que escogerlas de 10 en 10. La primera observación se selecciona aleatoriamente entre los 10 primeros números enteros, y las restantes se seleccionan automáticamente cada 10.

Ejemplo: muestreo sistemático

```
data dos;
  u=int(ranuni(0)*10+1);
  do i=u to 2000 by 10;
       set uno point=i;
   output;
  end;
  stop;
run;
```

Si se desea extraer una muestra de una proporción de las observaciones del archivo original, cuyo tamaño es desconocido, se utiliza la opción nobs=variable de la sentencia set, que guarda en una variable el valor del número de observaciones del archivo leído.

Ejemplo: extracción de una muestra según porcentaje del archivo original

Queremos extraer una muestra del tamaño 30% del archivo original con reemplazamiento.

```
data dos;
   z=1;
   if _n_=1 then set uno point=z nobs=total;
   do until (z>int(0.30*total));
      nume=int(total*ranuni(0)+1);
      set uno point=nume;
      z=z+1;
      output;
   end;
   stop;
run;
```

Sin reemplazamiento:

```
data dos ;
   set uno ;
   nume=int(30*ranuni(0));
run;
proc sort data=dos ;by nume;run;

data tres ;
   set dos nobs=total;
   if n > int(porcen*total+1) then stop ;
run ;
```

Las extracciones de muestras aleatorias presentadas anteriormente también se pueden realizar utilizando el procedimiento SURVEYSELECT del módulo STAT del SAS.

En el siguiente ejemplo se extrae una muestra eligiendo el archivo original aleatoriamente.

Ejemplo: extracción escogiendo aleatoriamente el archivo original

Supongamos que queremos decidir de manera aleatoria de qué archivo extraemos observaciones (aleatoriamente o no; en este caso obtenemos las 10 primeras de cada archivo). Utilizaremos la sentencia select:

```
data uno;
x=rantbl(0,1/3,1/3,1/3);
select(x);
  when(1) do i=1 to 10;
    set datos1 point=i;
    output;
   end;
  when(2) do i=1 to 10;
    set datos2 point=i;
    output;
     end;
  when(3) do i=1 to 10;
     set datos3 point=i;
     output;
    end;
stop;
run;
```

Capítulo 9. Arrays

En SAS, los arrays permitirán aplicar las mismas sentencias ejecutables a un gran número de variables. Este proceso consiste en crear grupos de variables (arrays) con elementos indexados, a los que se aplicarán las sentencias mediante un bucle.

Es importante señalar que un array es un grupo de **variables**. Por lo tanto, **para cada observación diferente**, los elementos del array, que son variables, tendrán **diferentes valores**.

Sentencia ARRAY

Agrupa **variables** en un vector.

Cada elemento del array es una variable (columna de datos). Por lo general la sentencia array se define en la línea posterior al data, para evitar problemas.

Las sentencias que se pueden utilizar para referirse a las variables de un ARRAY son: DO WHILE,DO UNTIL,DO/TO/BY,IF,PUT,SELECT,INPUT.

Sintaxis 1:

ARRAY nombre [$] elementos (valores iniciales);

Cada variable del ARRAY se llamará nombre{1},...nombre{i}, donde i es la posición que ocupa, en el transcurso del bloque data actual. Sin embargo estos elementos del array tomarán como nombres los nombres de los elementos a efectos de grabar en archivo de salida y de escribirse en el LOG.

Si las variables del ARRAY son alfanuméricas se pondrá el $ detrás del nombre del array.

Si se quiere dar valores iniciales a las variables se ponen entre paréntesis separados por espacios en blanco.

Ejemplo: transformación similar de varias variables usando arrays (1)

```
data uno;
array grupo a b c;
  input a b c;
  do i=1 to 3;
     if grupo{i}<4 then grupo{i}=4;
  end;
cards;
2 3 6
1 2 5
;
```

crea el archivo *uno*, con las siguientes variables y observaciones:

```
a    b    c

4    4    6
4    4    5
```

Obsérvese que las variables grupo1, grupo2 y grupo3 no son creadas, esos nombres solo se utilizan dentro del paso data, pero las variables creadas conservan los nombres anteriores a, b y c.

El bucle sobre el array grupo{i} es equivalente a las sentencias

```
if a<4 then a=4;
if b<4 then a=4;
if c<4 then a=4;
```

Sintaxis 2:

ARRAY nombre{n} variables (valores iniciales);

n indica la dimensión del vector.

Si se están creando las variables y no se les da nombre, adquieren nombre1...nombren. Los valores iniciales son opcionales, como en la sintaxis 1.

Ejemplo: transformación similar de varias variables usando arrays (2)

Supongamos que tenemos el archivo SAS *uno*, y nos interesa dar valor 0 a los valores negativos de las variables a, b y c, en todas las observaciones. Para ello :

Creamos un array con las variables a, b, c : a =x{1}, b=x{2}, c=x{3}.

Realizamos un bucle en el que i vaya de 1 a 3, y dentro del cual se realiza la operación mencionada con una variable cada vez.

Este proceso se repetirá tantas veces como observaciones haya en el archivo tratado, debido al funcionamiento de lectura iterativo de la sentencia SET.

El programa sería :

```
data dos ;
array x{3} a b c ;
   set uno ;
   do i=1 to 3 ;
     if x{i}<0 then x{i}=0 ;
   end ;
run ;
```

El archivo creado es idéntico al archivo uno, pero donde los valores de a, b y c han sido transformados según la expresión condicional, para todas las observaciones.

Se pueden crear variables nuevas con el nombre dado al array. El siguiente ejemplo crea un archivo artificial con 5 variables normales con media 0 y varianza 1 y 100 observaciones.

Ejemplo: creación de un archivo SAS artificial

```
data uno;
array x{5};
   do obse=1 to 100;
     do i=1 to 5 ;
       x{i}=rannor(0);
     end ;
     output;
   end;
run ;
```

El paso data anterior crea el archivo Sas uno, con 100 observaciones y 5 variables (las variables llamadas x1, x2, x3, x4 y x5). Obsérvese que la sentencia output está situada en el bucle exterior relativo a las observaciones.

Sería un error situarla en el bucle interior, pues se crearía un archivo con 500 observaciones en el cual, por ejemplo, la primera observación tomaría valor missing en las variables x2, x3, x4, x5, pues el valor inicial de éstas es el valor missing, representado por un punto '.' en el sistema SAS.

Los arrays son útiles también para leer masas de datos de texto sin tener que escribir los nombres de las variables. En el siguiente ejemplo se leen 10 variables en formato libre de un archivo en modo texto.

Ejemplo: lectura de datos utilizando arrays

```
data uno;
array x{10};
  input x1-x10;
cards;
2 4 8 2 1 5 9 0 4 5
5 8 6 9 5 34 5 5 5 6
5 4 3 2 1 7 8 9 0 10
;
```

En el siguiente ejemplo se ve como se puede hacer que los arrays permitan tratar los datos de una variable como se haría en un lenguaje de programación usual, es decir, considerando cada valor de una variable como un elemento del array.

Aunque esta manera de trabajar no corresponde al estilo de programación en SAS, a veces es más sencillo realizar ciertos cálculos así, sobre todo cuando se está "traduciendo" el programa de otro lenguaje de programación y se quiere incorporar al SAS.

El principal problema es que para cada variable creada en el array el sistema SAS reserva memoria para que ésta variable contenga un gran número de observaciones, y por lo tanto la creación de arrays con muchos elementos con el único propósito de que incluyan solo una observación por variable puede generar problemas de memoria.

Ejemplo: utilización de arrays en modo de programación "no SAS"

Se leerá la variable a presente en el archivo SAS uno introduciéndola en el array. Solamente en la última observación es cuando el array está completo y se puede trabajar con él. Es necesario utilizar la variable de sistema _n_ que irá indicando el número de observación (y por lo tanto el elemento del array), así como la variable fin que indica el fin de los datos .

```
data uno;
  input a @@;
cards;
2 4 8 2 1 5 9 0 4 5
;
data dos(drop=a);
array x{10};
  set uno end=fin;
  x{_n_}=a;
  if fin=1 then output;
run;
```

El archivo dos creado contiene una observación y 10 variables (las variables x1-x10 del array). Se puede entonces trabajar con esos datos dentro de un paso data, como si se tratara de un lenguaje clásico de programación. Como ejemplo se calculará la suma de los cuadrados de esa variable a través de un bucle.

```
data;
array x{10};
  set dos;
  suma=0;
  do i=1 to 10;
     suma=suma+x{i}**2;
  end;
  put ‘suma de cuadrados=’ suma;
run;
```

Una manera de realizar el anterior programa sin necesidad de arrays, programando de manera habitual en SAS, es la siguiente:

```
data uno;
  retain suma 0;
  input a @@;
  suma=suma+a**2;
  if _n_=10 then put ‘suma de cuadrados=’ suma;

cards;
2 4 8 2 1 5 9 0 4 5
;
```

Trabajar con las funciones SAS de tipo estadístico, si es posible, es más eficiente que utilizar bucles sobre los arrays. En el siguiente ejemplo se halla la media de varias variables para cada observación.

Programación eficiente

Ejemplo: utilizar funciones SAS en lugar de programar con bucles

Supongamos que en el archivo uno están las variables a b c d e y se desea calcular la media de ellas para cada observación.

```
data dos ;
 set uno;
 media=mean(of a--e);
run ;
```

El anterior programa es más eficiente y corto que el siguiente:

```
data dos (drop=i x1-x5);
array x{5} a b c d e;
 set uno;
 media=0;
 do i=1 to 5;
   media=media+x{i};
 end;
 media=media/5;
run ;
```

Los arrays en varias dimensiones tienen interés en ciertas facetas de programación.

ARRAY multidimensional

La sintaxis de la definición de array multidimensional es similar a la del array unidimensional. En la notación nombrada como sintaxis 2 anteriormente, sería :

ARRAY {n1,n2,...} variables (valores iniciales);

El orden de las variables creadas o existentes referentes a los elementos del array es de derecha a izquierda según esos elementos, y se verá a continuación.

Por ejemplo, si se utiliza

```
array vari{2,3} x1-x6;
```

el array contendría los siguientes elementos:

```
vari{1,1} vari{1,2} vari{1,3}       x1      x2      x3
vari{2,1} vari{2,2} vari{2,3}       x4      x5      x6
```

Ejemplo: array multidimensional

La utilidad del array multidimensional está en ordenar las variables según dimensiones que tengan sentido, como se verá en el siguiente ejemplo, donde hay dos ciudades (dimensión 1) con 5 medidas de temperatura cada una (dimensión 2).

Se desea hacer la transformación de las temperaturas de grados Farenheit a Celsius, para lo cual se organizan las variables en array bidimensional. Aunque se podría utilizar un array unidimensional también para esa operación, la estructuración del array bidimensional es más apropiada en cuanto a leer los datos y tenerlos guardados en un cierto orden, de manera a poder operar con esas variables en pasos data posteriores.

```
data tempes (drop=i j);
   array temprg{2,5} c1t1-c1t5 c2t1-c2t5;
   input c1t1-c1t5 /
         c2t1-c2t5;
   do i=1 to 2;
     do j=1 to 5;
       temprg{i,j}=(temprg{i,j}-32)/1.8;
     end;
   end;
cards;
89.5 65.4 75.3 77.7 89.3
73.7 87.3 89.9 98.2 35.6
75.8 82.1 98.2 93.5 67.7
101.3 86.5 59.2 35.6 75.7
;

proc print data=tempes;
   title 'Temperatura de dos ciudades';
run;
```

En SAS, en realidad siempre existe una dimensión más que la indicada en los arrays, pues cada variable se supone que contiene a su vez cierto número de observaciones. Aunque en el ejemplo anterior solamente había una observación por celda del array, usualmente el espíritu de la programación en SAS está en utilizar el hecho de que cada elemento del array es una variable conteniendo varias observaciones.

Se pueden utilizar los arrays multidimensionales en el sentido del ejemplo de programación “no SAS” planteado en arrays unidimensionales. En todo caso, la utilización de arrays multidimensionales no es tan frecuente, pues cuando se desea programar utilizando más de dos dimensiones en SAS se suele utilizar más habitualmente el módulo IML.

Capítulo 10. Introducción a los procedimientos SAS

Un programa SAS consta de Pasos DATA y Pasos PROC concatenados e independientes. Los Pasos PROC o procedimientos, son programas determinados que cumplen cada uno una función más o menos compleja. Un listado de procedimientos se puede encontrar en la introducción, pero a efectos de ejemplo se pueden citar algunas de estos :

Utilidades sobre archivos

PROC SORT (ordenar archivos), PROC TRANSPOSE (transponer archivos), PROC CONTENTS (información sobre archivos), PROC FORMAT (utilidades de formato de lectura y escritura).

Estadística descriptiva

PROC PRINT (listado de los valores de las variables de un archivo), PROC TABULATE (tablas), PROC MEANS, PROC UNIVARIATE (cálculo de estadísticos básicos)...

Estadística avanzada

PROC CLUSTER (análisis de conglomerados), PROC FACTOR (Análisis factorial), PROC REG (regresión multivariante), PROC ANOVA (análisis de la varianza),...

Gráficos

PROC GPLOT, PROC GCHART, PROC G3D,...

Dependiendo de los módulos de que se disponga, un gran número de procedimientos pueden cubrir otras funciones específicas como gráficos de control de calidad, estudio de series temporales o estudio de diagramas PERT.

Opciones Del Sistema

De gran importancia es saber establecer las opciones del sistema para obtener los resultados en un formato apropiado. Se puede acceder a la ventana options, pero lo más indicado es tener escrito un programa con la sentencia options que defina las opciones preferidas por el usuario. La sintaxis de la sentencia OPTIONS toma la forma

```
OPTIONS opciones;
```

Algunas de estas opciones son:

`center/nocenter`	Centra el texto en OUTPUT. A menudo es interesante no centrarlo.
`details/nodetails`	Detalles en el LOG.
`formdlim='caracter'`	Delimitador que separa páginas en OUTPUT. A menudo se debe definir como formdlim=‘’ .
`linesize=n`	Anchura de la línea en OUTPUT.
`pagesize=n`	Longitud de la página en OUTPUT.
`number/nonumber`	Números de línea en el programa en el LOG.
`source/nosource`	Se ve el programa escrito, una vez ejecutado, en el LOG.
`notes/nonotes`	Notas informativas en el LOG.
`date/nodate`	Fecha en las salidas en OUTPUT.

Ejemplo: Opciones personales por defecto

Es interesante utilizar una serie de opciones distintas de las establecidas por defecto en el sistema. Habitualmente y como primer programa de la sesión SAS yo suelo utilizar las siguientes opciones:

```
options

nocenter
nodate
nodetails
formdlim=''
linesize=90
nonumber
pagesize=30000
probsig=2
nosource
nonotes;
```

Ubicación de las salidas de los procedimientos

Los resultados de los procedimientos son por defecto destinados a la ventana OUTPUT. Los resultados gráficos son destinados a la ventana GRAPH. Estas opciones por defecto se pueden variar , si se desea por ejemplo crear un archivo con el contenido de la ventana OUTPUT.

La manera más sencilla de destinar el contenido de la ventana OUTPUT y/o de la ventana LOG a un archivo externo es utilizando el procedimiento PROC PRINTTO, que se verá a continuación.

PROC PRINTTO

`proc printto [LOG='archivo'][ PRINT='archivo'] [NEW];RUN;`

Destina los resultados dirigidos al LOG en los pasos data subsiguientes, al archivo indicado. Análogamente, destina los resultados de los procedimientos, que en principio van a la ventana OUTPUT, al archivo indicado. La opción NEW permite reemplazar el archivo indicado en lugar de añadir los resultados al final de éste. El procedimiento PRINTTO es muy útil cuando se realizan macros con repetición de muchos pasos data y proc, con la intención de obtener directamente los resultados en un fichero.

Cuando ya no se desee destinar la salida a ficheros, basta ejecutar:

```
proc printto;run;
```

Ejemplo: destinar por defecto a archivos de texto las salidas de la ventana OUTPUT y LOG

En el siguiente programa se destina al archivo datoslog.txt el contenido de la ventana LOG, y al archivo datoutput.txt el contenido de la ventana OUTPUT.

```
proc printto log='c:\datlog.txt' print='c:\datoutput.txt';
run;

data dos;
 set uno;
 if salario<1000 then put 'salario<1000' +5 'observación nº ' _n_;
run;

proc print data=uno noobs;run;
```

Aparecerá en el archivo c:\datlog.txt:

```
salario<1000     observación nº 1
salario<1000     observación nº 2
salario<1000     observación nº 7
salario<1000     observación nº 11
salario<1000     observación nº 23
salario<1000     observación nº 25
salario<1000     observación nº 27
salario<1000     observación nº 29
…
```

y en el archivo c:\datoutput.txt:

```
salario     sexo

 626.01    Mujer
 772.46    Hombre
1781.49    Hombre
1664.54    Mujer
1115.77    Hombre
1111.69    Hombre
 600.47    Mujer
1427.12    Mujer
1367.13    Mujer
1508.50    Mujer
 600.33    Mujer
1352.24    Mujer
1089.36    Mujer
```

Otro programa que destina los contenidos de las salidas de procedimientos a archivos es el ODS (Output Delivery System).

El programa ha de ser escrito antes de las sentencias de procedimientos, para que éstos destinen su salida al archivo nombrado.

Su sintaxis básica, aunque tiene muchas más opciones, es

ods [*opción de formato*] file=”archivo”;

Entre las opciones de formato del archivo de salida que contiene el contenido de la ventana OUTPUT, se pueden destacar:

listing	produce un archivo en modo texto
pdf	produce un archivo en modo pdf
html	produce un archivo en modo html

Cuando, en la sesión SAS en la que se ha ejecutado alguna vez el programa ODS ya no se quiera destinar al archivo el contenido de la ventana OUTPUT, se volverá a utilizar la opción por defecto que es *listing*, sin especificar archivo:

ods listing;

Algunas sentencias comunes a los Procedimientos

Por lo general la sintaxis de los procedimientos es rígida, de la forma

```
proc nombre; opciones...;
```

(las opciones suelen estar separadas por punto y coma ;).

Entre estas **opciones** la mayor parte son inherentes al procedimiento en cada caso,pero existen algunas que se pueden utilizar en casi todos:

VAR

```
VAR variables;
```

Indica las variables para las que se quiere ejecutar el procedimiento.

BY

```
BY [DESCENDING] variable [NOTSORTED];
```

Especifica que el procedimiento se ejecutará separadamente para cada uno de los grupos que forman los diferentes valores de la variable del BY.

El archivo con que se trabaje ha de ser previamente ordenado según la variable del BY, con PROC SORT. Si no es así, hay que utilizar la opción NOTSORTED.(Lo mejor es utilizar PROC SORT primero).

DESCENDING implica que la variable del BY ha sido ordenada en orden descendente.

Ejemplo: presentación de listado por valores de variable clave

```
proc sort data=uno;by sexo;
run;
proc print data=uno;var orto;
by sexo;
run;
```

realiza un listado de los valores de la variable ORTO en los casos del archivo SAS uno, indicando los grupos de sexo.

CLASS

```
CLASS variables;
```

En muchos procedimientos se puede utilizar, en lugar de BY, la opción CLASS, que significa lo mismo que la opción BY (realizar los análisis por separado por grupos de combinaciones de valores de las variables indicadas), pero la opción CLASS no requiere que el archivo esté previamente ordenado por las variables indicadas, con lo cual es más cómodo y rápido en cuanto a la programación que utilizar la opción BY. Sin embargo, si el archivo sí está previamente ordenado, es preferible utilizar la opción BY pues se ahorra tiempo de proceso.

ID

```
ID variables;
```

En algunos procedimientos se utiliza para designar variables de identificación de los individuos para cabeceras y etiquetas en las salidas.

FREQ

```
FREQ variable;
```

Nombra una variable cuyo valor es la frecuencia de ocurrencia de la Observación. El procedimiento tomará el archivo como si cada Observación apareciese n veces, siendo n el valor correspondiente de la observación en la variable del FREQ.

OUTPUT

```
OUTPUT OUT=archivo SAS [nombre clave=var.];
```

o, en algunos procedimientos,

```
OUT=archivo SAS [nombre clave=variables];
```

Graba las salidas específicas de cada procedimiento en un archivo SAS. A las variables grabadas se les puede asignar diferentes nombres que los dados por el procedimiento, o nombres claves. Si se trata de muchas variables, se puede utilizar la notación abreviada var1-vark. Por ejemplo, mean=media1-media15 se refiere a que la media de la primera variable del archivo estará en la variable media1 del archivo de salida, la media de la segunda en media2, etc.

Si el procedimiento se ejecuta con la opción BY, entonces el archivo de salida contiene una observación por cada combinación de las variables presentes en el BY.

Ejemplo: creación de archivos de salida en los procedimientos

En este ejemplo se crea un archivo SAS con información sobre estadísticos de las variables de otro archivo.

```
proc univariate data=uno;var x y;
output out=dos mean=mediax mediay min=minimox minimoy max=maximox
maximoy;
run;
```

El programa anterior graba en el archivo *dos* una observación con solamente los valores de las variables creadas:

```
mediax      mediay      minx
32          45          23
```

Ejemplo: creación de archivos de salida en los procedimientos con BY

En este ejemplo se crea un archivo SAS con información sobre estadísticos de las variables de otro archivo, pero realizando el análisis por grupos de dos variables, llamadas SEXO (dos categorías) y GRUPO (3 categorías). Como hay 2x3=6 combinaciones de valores de ambas variables, el archivo de salida contiene 6 observaciones, con los estadísticos calculados sobre las variables continuas para cada grupo de observaciones formado por cada combinación de valores de las variables expuestas en BY.

```
proc univariate data=uno;var x y;
output out=dos mean=mediax mediay min=minimox;
by sexo grupo;
run;
```

El programa anterior graba en el archivo *dos* una observación con solamente los valores de las variables creadas:

sexo	grupo	mediax	mediay	minimox
1	1	32	47	18
1	2	31	52	21
1	3	37	44	22
2	1	29	43	23
2	2	35	41	27
2	3	24	50	29

WHERE

Se trata de la única sentencia condicional del paso PROC.

```
WHERE expresión condicional;
```

donde la expresión condicional es similar a la del WHERE del paso DATA: puede contener expresiones lógicas y los operadores *,/,+,- pero no funciones SAS.

La sentencia sólo permite que se ejecute el procedimiento para las observaciones que cumplan la condición.

Ejemplo: utilización de la opción *where* en los procedimientos

```
proc print data=uno;var orto;where sexo=2;
run;
```

Sólo presenta el valor de orto para las observaciones con sexo femenino.

TITLE

Presenta un título para las salidas. Es un comando que se emplea antes de los procedimientos. La sintaxis es

```
TITLE 'texto';
```

De utilidad a veces es poner simplemente

```
TITLE ' ';
```

con lo que se evita el título de SAS que aparece por defecto o cualquier título declarado anteriormente.

La opción NOPRINT

Es una opción interesante para ahorrar memoria y tiempo de proceso cuando no se desea ver los resultados de los procedimientos en la ventana OUTPUT, y solamente interesa crear los archivos de salida. Se escribe en general directamente después de data=archivo. Por ejemplo, si en el ejemplo anterior del proc univariate sólo interesara el archivo de salida, y no el contenido de la ventana OUTPUT, se utilizaría el siguiente :

Ejemplo: utilizar la opción noprint para no obtener resultados en la ventana OUTPUT

```
proc univariate data=uno noprint;var x y;
output out=dos mean=mediax mediay min=minimox;
by sexo grupo;
run;
```

Capítulo 11. Formato de escritura de variables SAS

A menudo es útil asociar a ciertas variables características de formato de escritura, para que sus valores aparezcan en la salida de los procedimientos SAS tal como deseamos. Algunas posibilidades son especificar el número de decimales que se desean presentar para algunas variables numéricas, o si desean presentarse con sus valores con algún signo añadido, o especificar etiquetas de valores para algunas variables.

Estas opciones se pueden abordar con el procedimiento siguiente.

Proc Format

Formatos para representar valores puntuales de variables cualitativas

La utilización de fotmatos de escritura consiste en crear un formato de escritura con PROC FORMAT, asignándole un nombre, y posteriormente asignar el formato de escritura correspondiente a cada variable que nos interese, en los procedimientos SAS que se vayan a utilizar.

Si la variable es cualitativa o discreta los formatos suelen ser puntuales, uno por cualidad o valor diferente de la variable.

```
proc format opciones;value [$] nombre valor1='formato-1' …
valorn='formato-n';
```

Si la variable cualitativa es alfanumérica se indica la opción $.

El formato se indica en los procedimientos mediante $formato. si es alfanumérico o bien formato. si es numérico, como se verá en el procedimiento print del siguiente ejemplo.

Ejemplo: formato para valores de variables categóricas alfanuméricas

Supongamos que en el archivo uno está la variable sexo cuyos valores son 'v' y 'm'.

```
proc format;value $formsex 'v'='varon' 'm'='mujer';run;
proc print data=uno;format sexo $formsex.;run;
```

Mostará un listado de los valores de la variable sexo del archivo *uno* , pero los valores aparecerán bajo la forma VARON y MUJER en vez de V y M.

Ejemplo: formato en variables categóricas numéricas

```
proc format;value formedad low-12=’niño’ 13-19=’adolescente’ 20-
high=’adulto’;
run;
data uno;input edad;
cards;
15
23
7
13
;
proc print data=uno;
format edad formedad.;
run;
```

Salida en la ventana OUTPUT:

```
OBS    EDAD

 1     adolescente
 2     adulto
 3     niño
 4     adolescente
```

La función put(variable,formato) ;

Esta sentencia identifica la variable con los valores del formato indicado, que previamente ha sido definido en un procedimiento format anterior. La variable2 creada es siempre alfanumérica. La sintaxis es

```
variable2=put(variable1,formato.) ;
```

Es más eficiente utilizar los formatos de escritura y la sentencia put con el formato que crear nuevas variables alfanuméricas para representar valores. El siguiente ejemplo lo ilustra.

Programación eficiente

Ejemplo: utilizar la función put(variable, formato) para crear nuevas variables cualitativas

```
data uno;
 length pais $ 30;
 input pais $ 1-30;
cards;
España
Estados Unidos
Japón
;
proc format;
value $mone
'Estados Unidos'='Dólar'
'España'='Euro'
'Japón'='Yen';
run;
data dos;
 set uno;
 moneda2=put(pais,$mone.);
run;
```

```
proc print data=dos;run;
```

Se obtiene la siguiente salida en la ventana OUTPUT:

```
pais              moneda2

España             Euro
Estados Unidos     Dólar
Japón              Yen
```

El programa anterior es más eficiente que crear la variable moneda2 directamente, pues las sentencias condicionales requieren que el sistema busque si se cumple la condición:

```
data dos;
set uno;
 if pais='Estados Unidos' then moneda2='Dólar';
 else if pais='España' then moneda12='Euro';
 else if pais='Japón' then moneda2='Yen';
run;
```

Formatos para representar números

Se utiliza la opción

```
picture nombre valores1='formato-1' … valoresn='formato-n';
```

El formato consiste en signos 0 y 9, combinados con caracteres escogidos por el programador. Tanto el 0 como el 9 representan dígitos de la variable a la que se le aplica el formato, pero los 0 indican que no se escribirán ceros a la izquierda, mientras que sí se escribirán si utilizamos el código 9.

La opción *(noedit)* al final de la sentencia picture especifica que el formato presentado no representa valores, sino un mensaje (ver el siguiente ejemplo).

Ejemplo: representación de intervalos numéricos

```
proc format; picture ingresos 0-70000='00000'
                        70001-150000='000000'
                        other='mas de 150000' (noedit);
run;
data uno;input ingre;
cards;
50000
300000
130000
;
proc print data=uno;
format ingre ingresos.;
run;
```

Da lugar a la salida del procedimiento print:

```
OBS             INGRE

 1              50000
 2     mas de 150000
 3             130000
```

Opciones especiales de la sentencia picture

low, high

Representan el mínimo y máximo valor respectivamente:

```
picture ingresos low-70000='00000' 70000-high='000000';
```

other

Representan cualesquiera otros valores, como está indicado en el ejemplo.

Prefix=’caracteres’

Presenta un prefijo antes del valor numérico:

```
picture ingresos low-70000='00000' (prefix=’pts ’);
```

presenta el valor 35000 como

```
pts 35000
```

Se pueden utilizar de manera conjunta decimales y comas:

```
picture ingresos low-70000='00,000.00';
```

presenta el valor 35000 como

```
pts 35,000.00
```

Ejemplo: formatos numéricos

Algunos ejemplos de creación de formato.

Valor	Picture	Presentación en procedimientos
3456.66	000.00	456.66
565.78	000.0	565.7
54.21	000.00	54.21
54.21	999.99	054.21
5421	000.00	421.00

Capítulo 12. Algunos Procedimientos del módulo SAS/BASE

Proc Print

El procedimiento print

```
proc print data=archivo [round] [noobs];[VAR var; ID var;BY var;SUM var;]
run;
```

Presenta en la pantalla de OUTPUT un listado de los valores de los datos.

Con la opción

```
round
```

se redondean los números de la salida a dos cifras decimales, para las variables de las que se ha pedido la suma (ver SUM, abajo).

La opción

`noobs`

es para que no se presente el número de observación en el listado.

La opción

`SUM` *`var;`*

presenta la suma de los valores de la variable indicada.

La opción

`ID` *`variable`*`;`

presenta, en lugar del número de observación, el valor de la variable de identificación *variable.*

Ejemplo: utilización del procedimiento print

En primer lugar se creará artificialmente un archivo para ser presentado con el proc print.

```
data uno;
  do i=1 to 10;
     x=5+normal(i)*2;
     y=3*ranexp(i);
     output uno;
  end;
run;

proc print data=uno round;id i;sum x;
run;
```

presenta en OUTPUT los valores de las variables simuladas x (redondeada a 2 decimales) e y, y los valores respectivos de i, así como la suma de los valores de x:

```
 i       x         y

 1      8.61     2.75
 2      7.89     0.09
 3      2.83     9.00
 4      6.96     1.94
 5      6.03     0.13
 6      4.56     1.11
 7      5.06     1.12
 8      2.52     3.74
 9      6.37     1.36
10      3.22     2.93
       =====
       54.05
```

Proc Sort

El procedimiento sort ordena el archivo por las variables indicadas.

```
proc sort data=archivo [OUT=archivo];BY variables;
run;
```

Si se indica el archivo de salida, es conservado el orden original en el primer archivo. Si no, el archivo original es alterado en orden.

El procedimiento sort es necesario si se desea utilizar la opción BY *variables*; en algún procedimiento posterior. En el ejemplo siguiente se utiliza para presentar un listado con proc print por valores de la variable sexo.

Ejemplo: utilización del procedimiento sort y print (1)

```
proc sort data=uno;by sexo;
run;

proc print data=uno noobs; by sexo;
run;
```

Presenta el siguiente listado:

```
sexo=0

 i       x          y

 1    8.60965    4.04817
 2    5.79315    1.89507
 6    3.52440    3.74246
 7    6.37001    1.58104
 9    5.68142    3.63982
10    2.30031    5.34980


sexo=1

i       x          y

3    9.47659    1.93953
4    6.02732    3.64011
5    3.81164    4.45493
8    3.51144    2.04008
```

Ejemplo: utilización del procedimiento sort y print (2)

```
proc sort data=uno;by sexo;
run;

proc print data=uno noobs;sum x;by sexo;
run;
```

Presenta el siguiente listado:

```
sexo=0

   i       x          y

   1     8.6096    4.04817
   2     5.7932    1.89507
   6     3.5244    3.74246
   7     6.3700    1.58104
   9     5.6814    3.63982
  10     2.3003    5.34980
----    -------
sexo    32.2789


sexo=1

   i       x          y

   3     9.4766    1.93953
   4     6.0273    3.64011
   5     3.8116    4.45493
   8     3.5114    2.04008
----    -------
sexo    22.8270
        =======

        55.1059
```

La opción noequals

La opción *noequals* se utiliza, usualmente, cuando se está ordenando el archivo por una variable categórica, de modo que hay varias observaciones por categoría. En este caso, si no es necesario que las observaciones pertenecientes a la misma categoría conserven el mismo orden que en el archivo original, la opción noequals realiza la ordenación por categorías, ahorrando tiempo de proceso.

Programación eficiente

Ejemplo: utilización del procedimiento sort con la opción noequals

```
proc sort data=uno noequals;by sexo;
run;
```

Queda el archivo ordenado por sexo, pero dentro de cada categoría de sexo no se conserva forzosamente el orden original de las observaciones.

Proc Contents

El procedimiento contents presenta información sobre el archivo SAS indicado: sus variables, el número de observaciones, longitud de las variables y posición que ocupan.

```
proc contents data=archivo [OUT=archivo];
run;
```

Se presenta información sobre el archivo SAS indicado: sus variables, el número de observaciones, longitud de las variables y posición que ocupan. El archivo creado en la opción OUT contiene esta información en forma de variables.

Ejemplo: utilización del procedimiento Contents

Supongamos que el archivo uno contiene 100 observaciones de las variables salario y sexo:

```
salario     sexo

1000.35    Mujer
1528.62    Hombre
1580.16    Hombre
1026.80    Mujer
…
```

El programa

```
proc contents data=uno;
run;
```

obtiene la siguiente salida en la ventana OUTPUT:

```
Nombre conj. datos        WORK.UNO   Observaciones                100
Tipo miembro              DATA       Variables                    2
Engine                    V9         Índices                      0
Creado                    martes     Longitud de la observación   16
                          25 de
                          septiemb
                          re de
                          2006
                          16H11
Última modificación       martes     Observaciones borradas       0
                          25 de
                          septiemb
                          re de
                          2006
                          16H11
Protección                           Comprimido                   NO
Tipo de conj. datos                  Clasificado                  NO
Etiqueta
Representación de datos   WINDOWS_
                          32
Codificación              wlatin1
                          Western
                          (
                          Windows)


           Información referente a la máquina y al host

Tamaño de página del conj. datos        4096
Número de páginas del conj. datos       1
Primera página de datos                 1
Obs. máx por página                     252
Obs. en primera página de datos         100
Número de reparaciones del conj. datos  0
Nombre de fichero                       C:\DOCUME~1\ucm\CONFIG~1\
                                        Temp\SAS Temporary
                                        Files\_TD3876\uno.sas7bdat
Versión creada                          9.0101M3
Host creado                             XP_PRO


 Lista alfabética de variables y atributos

Núm    Variable    Tipo            Longitud

  1    salario     Numérica               8
  2    sexo        Alfanumérica           6
```

A veces se necestia obtener un listado de las variables presentes en un archivo, cuando estas son muy numerosas, en un archivo de modo texto o como texto en la ventana LOG. Esa lista de variables puede ser utilizada, utilizando “copiar y pegar” en programas SAS sencillos o macros. El procedimiento CONTENTS permite obtener ese listado, como se verá en el

ejemplo siguiente. Los nombres de las variables en el archivo de salida están en la variable NAME

Ejemplo: obtención de un listado de las variables presentes en un archivo

En este ejemplo se expondrá el listado en la ventana LOG y también en el archivo listvar.txt. En primer lugar se crea un archivo de prueba. En el proc contents se utiliza la opción noprint, pues solamente interesa el listado obtenido, y no la información completa del archivo en la ventana OUTPUT.

```
data uno;
input edad sexo altura salario hijos;
cards;
23 1 170 20000 1
28 2 190 15000 0
;
proc contents data=uno out=dos noprint;
run;
proc print data=dos;var type length varnum name ;run;
```

El archivo de salida tiene muchas variables (ver el manual), de las cuales solo presentamos el tipo de variable, longitud, orden que ocupa la variable en el archivo y su nombre (esta variables es alfanumérica):

```
Obs    TYPE    LENGTH    VARNUM    NAME

 1       1        8         3      altura
 2       1        8         1      edad
 3       1        8         5      hijos
 4       1        8         4      salario
 5       1        8         2      sexo
```

A continuación se obtiene el listado deseado mediante un paso data.

```
data;
 set dos;
 put name @@;
 file 'c:\listvar.txt';
 put name @@;
run;
```

En la ventana LOG y también en el archivo c:\listvar.txt se encuentra la lista:

```
altura edad hijos salario sexo
```

Proc Transpose

El procedimiento transpose transpone el archivo cambiando observaciones a variables y variables a observaciones. Su sintaxis es:

`proc transpose data=`*`archivo opciones`*`;[VAR `*`var`*`;ID `*`var`*`;BY `*`var`*`;]`

Algunas **opciones** (separadas por espacios en blanco) son:

`PREFIX=prefijo`

prefijo para las nuevas variables creadas: prefijo1,...,prefijon. Se recomienda por un mayor control en la programación.

`NAME=variable`

especifica la variable del archivo de salida que conserva los nombres de las var. originales. Recomendable también.

`OUT=archivo`

archivo donde se grabará el archivo SAS transpuesto. Necesario para no perder el archivo original.

Ejemplo: transposición de un archivo SAS

```
data orig;
 input x y;
cards;
1 2
4 5
6 7
4 5
;
proc transpose data=orig prefix=a name=nombres out=trans;
run;
proc print data=trans;
run;
```

da como resultado en la ventana OUTPUT:

OBS	NOMBRES	A1	A2	A3	A4
1	X	1	4	6	4
2	Y	2	5	7	5

Ejemplo: operaciones con una matriz utilizando arrays y proc transpose

Dada una matriz cuadrada A, calcular A+A', siendo A'=A transpuesta.

(Creamos A, con 4 variables y 4 observaciones).

```
data uno;
 input x1-x4;
cards;
1 3 5 1
3 5 8 7
3 2 1 1
3 2 2 2
;
proc transpose data=uno prefix=y out=tra;
run;
data sum;
  array x{4};
  array y{4};
  array z{4};
  set uno;
  set tra;
  do i=1 to 4;
   z{i}=x{i}+y{i};
  end;
run;
```

realiza la operación deseada.

Proc Univariate

El proc univariate es uno de los procedimientos estadísticos básicos de interés para variables continuas. Presenta en la ventana OUTPUT estadísticos univariantes para las variables indicadas, y permite crear un archivo de salida con estadísticos. Como sus opciones son prácticamente idénticas a las de otros procedimientos como Proc Means o Proc Summary, solamente se presentará el procedimiento Univariate.

```
proc univariate data=archivo opciones;[VAR var; BY var; FREQ var; ID var; OUTPUT OUT=archivo];
run;
```

Entre las opciones cabe destacar:

PLOT-. tallos y hojas, box-plot, plot de normalidad.

FREQ-. tabla de frecuencias.

NORMAL-. Test de normalidad de Shaphiro-Wilks.

ROUND-. unidades de redondeo para los datos que se presentan en la ventana OUTPUT.

Aparte de la información estadística que aparece en la ventana OUTPUT, si se crea un archivo de salida éste contiene una observación por cada variable del archivo inicial, con los valores de las variables que representan estadísticos como N, MEAN, SUM, STD, VAR, SKEWNESS, KURTOSIS, MAX, MIN, RANGE, Q1, Q3, MEDIAN, P1, P5, P10, P90, P95, P99, MODE.

En la sentencia OUTPUT OUT=archivo se da nombre a estas variables, como se verá en el siguiente ejemplo. La sintaxis está explicada en la sección de sentencias comunes a los procedimientos. En el procedimiento Univariate, si se utiliza OUTPUT OUT es necesario explicitar la lista de variables en la sentencia VAR variables.

Ejemplo: proc univariate

```
options nocenter linesize=66;
data uno (drop=i);
do i=1 to 100;
  x=4*normal(i)+10;
  output;
end;
run;
proc univariate data=uno plot normal;
var x;
output out=dos mean=media max=maximo skewness=asimetr;
run;

proc print data=dos;
run;
```

La salida del procedimiento aparece en la ventana OUTPUT:

```
Procedimiento UNIVARIATE
Variable:  x

                          Momentos

N                          100  Pesos de la suma           100
Media                 9.88045277  Observaciones        988.045277
                                  de la suma
Desviación típica     3.81313743  Varianza             14.5400171
Asimetría             0.11067712  Kurtosis             -0.3848217
Suma de cuadrados     11201.7964  Suma de cuadrados    1439.46169
no corregidos                     corregidos
Coeficiente           38.5927398  Media de error       0.38131374
de variación                      estándar
```

Medidas estadísticas básicas

Localización		Variabilidad	
Media	9.880453	Desviación típica	3.81314
Mediana	9.584535	Varianza	14.54002
Moda	.	Rango	18.89289
		Rango intercuantil	6.11071

Tests para posición: Mu0=0

Test	-Estadístico-		-----P-valor------	
T de Student	t	25.91161	Pr > |t|	<.0001
Signo	M	50	Pr >= |M|	<.0001
Puntuación con signo	S	2525	Pr >= |S|	<.0001

Tests para normalidad

Test	--Estadístico--		-----P-valor------	
Shapiro-Wilk	X	0.983894	Pr < W	0.2636
Kolmogorov-Smirnov	D	0.079381	Pr > D	0.1219
Cramer-von Mises	W-Sq	0.103977	Pr > W-Sq	0.0989
Anderson-Darling	A-Sq	0.622247	Pr > A-Sq	0.1028

Cuantiles (Definición 5)

Cuantil	Estimador
100% Máx	18.9531775
99%	18.3333552
95%	16.2110953
90%	15.2083901

El procedimiento print muestra el archivo de salida creado por el proc univariate, con una observación con las tres variables creadas:

Obs	media	asimetr	maximo
1	9.88045	0.11068	18.9532

En el siguiente ejemplo, más complejo, se muestra cómo utilizar el archivo de salida del proc univariate .

Ejemplo: utilización del archivo de salida del proc univariate

Cada mes obtenemos los datos de ganancias de 4 empresas, en variables en la que cada Observación es el resultado de una operación de la empresa en ese mes, en el archivo datos:

x1=resultados de operaciones en la empresa 1 en el mes 1.

x2=" " " " 1 " 2.

...

x48= 4 " 12.

1) Obtener la media del resultado de las operaciones en cada empresa en los meses Enero, Marzo,Diciembre.

Corresponderá a la media de X1,X3,X12,X13,X15,X24,X25,X27,X36,X36,X39,X48.

2) Obtener la suma total de las ganancias de las cuatro empresas en el mes de Abril.

Será la suma (sum(X4)+sum(X16)+sum(X28)+sum(X40)).

```
/* INTRODUCIMOS EN UNA OBSERVACION LAS MEDIAS Y SUMAS DE TODAS LAS
*/
/* VARIABLES, EN EL ARCHIVO DAT2 */

proc univariate data=datos; output out=dat2 mean=media1-media48
sum=sum1-sum48;
var X1-X48;
run;

/* REALIZAMOS UN PROGRAMA PARA VER EN LOG SOLO LO QUE INTERESA */

data ver;
array media{4,12};
set dat2;

sumtot=sum4+sum16+sum28+sum40;
put 'SUMA TOTAL DE GANANCIAS EN ABRIL';

do i=1 to 4;
   do j=1,3,12;
   put 'EMPRESA 'i 'MES ' j ':MEDIA GANANCIAS=' media{i,j};
  end;
end;
run;
```

Proc Freq

El procedimiento freq es el procedimiento básico para variables categóricas o numéricas discretas. Produce tablas de frecuencia con medidas de asociación y test Ji-cuadrado. La sintaxis básica es

```
proc freq data=archivo;tables tabla / opciones;[BY var];
run;
```

tabla indica la(s) variable(s) que forman la tabla de frecuencias.

Si se desea la lista de frecuencias de una sola variable x:

```
proc freq data=archivo; tables x;
```

Para la tabla cruzada de dos variables x e y:

```
proc freq data=archivo; tables x*y;
```

Para varias tablas cruzadas de x e y según los valores de la variable z:

```
proc freq data=archivo;tables x*y*z;
```

Las opciones de la tabla se indican a continuación de la barra separadora / . Algunas de estas opciones son:

CHISQ-. Test Ji-cuadrado y V de Cramer, Phi, Contingencia.

EXPECTED-. Frecuencias esperadas en la tabla.

CUMCOL-. Porcentajes acumulados para cada columna de la tabla.

Para el archivo de salida se utilizará

OUT=archivo.

En este archivo de salida hay una observación para cada combinación de valores de las variables de la tabla. Cada observación contiene los valores diferentes de estas variables, el valor de COUNT (frecuencia de esa combinación de valores) y PERCENT (porcentaje de ocurrencia de esa combinación de valores).

Si los valores de las variables tienen asociado algún tipo de formato, éste es el que se

utiliza para representar esos valores en el Procedimiento Freq.

Ejemplo: utilización del procedimiento freq

```
data uno;
  input x y;
cards;
2 3
1 4
1 2
1 3
2 2
2 3
1 4
2 3
;
proc freq data=uno;tables x*y / chisq expected out=dos;
run;
proc print data=dos;
run;
```

La salida en la ventana OUTPUT es:

```
Tabla de x por y

x          y

Frequencia|
Esperada  |
Porcentaje|
Pct fila  |
Pct col   |       2|       3|       4|  Total
----------^--------^--------^--------^
        1 |      1 |      1 |      2 |      4
          |      1 |      2 |      1 |
          |  12.50 |  12.50 |  25.00 |  50.00
          |  25.00 |  25.00 |  50.00 |
          |  50.00 |  25.00 | 100.00 |
----------^--------^--------^--------^
        2 |      1 |      3 |      0 |      4
          |      1 |      2 |      1 |
          |  12.50 |  37.50 |   0.00 |  50.00
          |  25.00 |  75.00 |   0.00 |
          |  50.00 |  75.00 |   0.00 |
----------^--------^--------^--------^
Total             2        4        2        8
                25.00    50.00    25.00   100.00


Estadísticos para Tabla de x por y
```

```
Estadístico                                    DF       Valor
-----------------------------------------------------------
Chi-cuadrado                                    2      3.0000
Ratio chi-cuadrado de la verosimilitud          2      3.8191
Chi-cuadrado Mantel-Haenszel                    1      0.8750
Coeficiente Phi                                        0.6124

 Aviso: 100% de las celdas esperaban cuentas menores
                 que 5. Puede que chi-cuadrado
                no sea un test válido.

Estadístico                                   Probabilidad
-----------------------------------------------------------
Chi-cuadrado                                        0.2231
Ratio chi-cuadrado de la verosimilitud              0.1481
Chi-cuadrado Mantel-Haenszel                        0.3496
Coeficiente Phi

    Aviso: 100% de las celdas esperaban cuentas menores
         que 5. Puede que chi-cuadrado no sea un test válido.
```

El archivo de salida queda:

```
        OBS    X    Y    COUNT    PERCENT

         1     1    2      1        12.5
         2     1    3      1        12.5
         3     1    4      2        25.0
         4     2    2      1        12.5
         5     2    3      3        37.5
```

Se observa que hay 5 observaciones, pues en el archivo original solamente hay 5 combinaciones de los valores de las variables X e Y. La variable COUNT indica el número de observaciones del archivo que toman los valores respectivos de X e Y presentados. Por ejemplo, hay COUNT=2 observaciones con valor X=1 e Y=4. A su vez, la variable PERCENT indica el porcentaje relativo de observaciones que toman los valores respectivos X e Y.

Proc Rank

```
proc rank data=archivo1 out=archivo2 [GROUPS=n] [FRACTION]
[TIES=HIGH/MEAN/LOW];
VAR variables;
RANKS variables;
run;
```

Crea un archivo nuevo con rangos creados según la opción escogida. Se especificará GROUPS si se desea formar grupos basados en percentiles de las variables indicadas en el apartado var, y FRACTION para rangos fraccionales (dividiendo el rango por el número de observaciones no missing). La opción TIES especifica el rango escogido para los empates. Las nuevas variables a creara que indican los rangos se indicarán en la opción RANKS. Por defecto los números de grupos creados son números enteros que comienzan por cero.

El procedimiento RANK no genera salida en la ventana OUTPUT. Su único cometido es crear el archivo de salida. Es útil, por ejemplo,

- para obtener grupos basados en percentiles de la población.

- para saber el orden que toma cada individuo en los datos basado en valores de una variable continua. Por ejemplo, el puesto en una carrera basado en el tiempo utilizado.

Ejemplo: creación de rangos basados en percentiles

Supongamos que en el archivo uno está la variable edad y queremos formar cinco grupos basados en los percentiles de esta variable, asignando el valor 0 al grupo de menor edad, y así sucesivamente.

```
proc rank data=uno out=dos groups=5;
var edad;
ranks edad2;
run;

proc print data=dos;
var edad edad2;
run;
```

Aparece en la ventana OUTPUT:

```
Obs    EDAD    EDAD2

  1      15       3
  2      16       4
  3      15       3
  4      15       3
  5      15       3
  6      16       4
  7      15       3
  8      14       1
  …
```

Ejemplo: creación una variable de ordenación

Supongamos que en el archivo uno está la variable peso y queremos ordenar los items de acuerdo a su peso. Como queremos obtener tantos grupos como valores de peso diferentes haya, se utiliza proc rank sin la opción groups=n.

```
proc rank data=uno out=dos;
var peso;
ranks peso2;
run;

proc print data=dos;
var peso peso2;
run;
```

Aparece en la ventana OUTPUT:

```
Obs    peso    peso2

 1      25       4
 2      34       5
 3      15       1
 4      16       2
 5      21       3
 …
```

Proc Corr

El procedimiento CORR Calcula la matriz de correlaciones de un archivo SAS. El archivo de salida es un archivo SAS que contiene esta matriz de correlaciones, además de estadísticos básicos para cada variable. En principio se calcula la matriz para todas las variables nombradas. Si se desean solamente las correlaciones concretas de unas variables con otras, se debe utilizar la opción WITH.

```
proc corr data=archivo1 out=archivo2 ;VAR var WITH var;
run;
```

El siguiente ejemplo ilustra la salida por defecto del procedimiento corr, aparte de presentar la posibilidad, mediante programación, de un listado ordenado de las correlaciones entre variables de más alta a más baja. El programa se puede generalizar para cualquier número y nombre de variables (véase el capítulo dedicado a las *macros*).

Ejemplo: Identificación de las variables más correladas

Supongamos que en el archivo *uno* están las variables de x1 a x4 y deseamos comprobar cuáles de ellas están más correladas, sin necesidad de buscar visualmente en la matriz de correlaciones (esta necesidad es muy frecuente en estudios con muchas variables). En primer lugar se ejecuta el proc corr:

```
proc corr data=uno outp=dos;run;
```

En la ventana OUTPUT aparece la matriz de correlaciones y estadísticos básicos (no los presentamos aquí), y además es creado el archivo de salida *dos.*

```
          Coeficientes de correlación Pearson, N = 200
              Prob > |r| suponiendo H0: Rho=0

                   x1            x2            x3            x4

x1        1.00000        0.63009       0.07802       0.07050
                          <.0001        0.2722        0.3212

x2        0.63009        1.00000      -0.01206      -0.03388
           <.0001                       0.8654        0.6339

x3        0.07802       -0.01206       1.00000       0.89493
           0.2722         0.8654                      <.0001

x4        0.07050       -0.03388       0.89493       1.00000
           0.3212         0.6339        <.0001
```

```
proc print data=dos;run;
```

Se observa con este *proc print* el contenido del archivo de salida *dos*, donde la variable _TYPE_ indica el estadístico y las correlaciones están representadas, en las líneas con _TYPE_=CORR, de modo que la correlación entre x1 y x2 está en la observación 4 del archivo de salida (que corresponde al valor 'x1' de la variable _NAME_), en el valor de la variable x2.

```
Obs   _TYPE_   _NAME_        x1          x2          x3          x4

 1     MEAN                0.523       1.522       0.509       3.521
 2     STD                 0.287       1.274       0.281       2.209
 3     N                 200.000     200.000     200.000     200.000
 4     CORR      x1        1.000       0.630       0.078       0.071
 5     CORR      x2        0.630       1.000      -0.012      -0.034
 6     CORR      x3        0.078      -0.012       1.000       0.895
 7     CORR      x4        0.071      -0.034       0.895       1.000
```

El programa presentado a continuación conserva en primer lugar las observaciones que contienen las correlaciones; a continuación construye un archivo con las correlaciones de modo que ocupan una por observación, con las variables i y j como indicadores de las variables originales.

Por último se ordena el archivo creado por orden descendiente de las correlaciones en valor absoluto y se muestran en la ventana LOG.

```
data tres;
 set dos;
 if _type_='CORR' then output;
run;

data cuatro (keep=i j corre correabs);
array x{4};
set tres;
j=_n_;
do i=1 to 4;
 if i<=j then do;
 correabs=abs(x{i});
 corre=x{i};
 output;end;
end;
run;

proc sort data=cuatro;by descending correabs;run;

data;
 set cuatro;
 if _n_=1 then put 'i' @5 'j' @10 'Abs(Correlación)' @40
'Correlación' //;
 if i ne j then put i @5 j @10 correabs @40 corre;
run;
```

Se obtiene en la ventana Log el siguiente listado, donde los índices i y j corresponden a pares de variables diferentes. Se observa, por ejemplo, que la correlación más alta en valor absoluto es de 0.89 y se da enter las variables x3 y x4:

```
i   j    Abs(Correlación)              Correlación


3   4    0.8949291446                 0.8949291446
1   2    0.630092044                  0.630092044
1   3    0.0780171956                 0.0780171956
1   4    0.0705016284                 0.0705016284
2   4    0.0338772042                 -0.033877204
2   3    0.0120571053                 -0.012057105
```

Proc Reg

El procedimiento REG estima los parámetros de regresión por mínimos cuadrados, aportando todas las opciones propias del análisis de regresión (tabla ANOVA, residuos, etc.). El archivo de salida contiene las variables originales, residuos y predicciones.

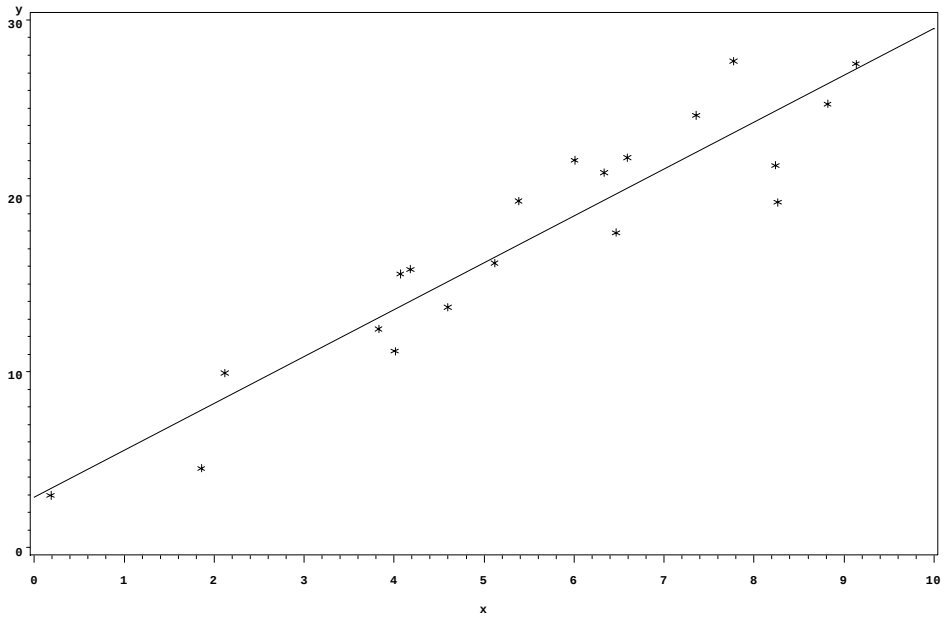
```
proc reg data=archivo1; model var.dependientes=var.regresoras;
OUTPUT OUT=archivo2 r=var p=var;
run;
```

Ejemplo: Proc Reg

En este ejemplo simulamos los datos previamente. Se utiliza el procedimiento gráfico gplot para representar la nube de puntos y la recta de regresión estimada.

```
data uno (drop=i epsi);
    do i=1 to 20;
      epsi=rannor(111)*sqrt(6);
      x=rannor(222)*3+4;
      y=2+3*x+epsi;
      output;
    end;
run;
symbol i=rl v=star c=black;

proc gplot data=uno;plot y*x / overlay;run;
```




```
proc reg data=uno;model y=x;
output out=dos r=resi p=predi;
run;
```

La salida del procedimiento REG en la ventana OUTPUT es la siguiente:

```
Procedimiento REG
Modelo: MODEL1
Variable dependiente: yte

Number of Observations Read          20
Number of Observations Used          20


                    Análisis de la varianza

                                  Suma de      Cuadrado
Fuente                    DF     cuadrados   de la media   F-Valor

Modelo                     1     809.05052     809.05052    127.54
Error                     18     114.18044       6.34336
Total corregido           19     923.23096

 Análisis de la varianza

Fuente              Pr > F

Modelo              <.0001
Error
Total corregido
```

```
Raíz MSE                 2.51860    R-cuadrado     0.8763
Media dependiente       17.59512    R-Cuad Adj     0.8695
Var Coeff               14.31421


                    Estimadores del parámetro

                 Estimador del         Error
Variable    DF       parámetro      estándar   Valor t   Pr > |t|

Término      1         2.88082       1.41941      2.03     0.0574
Indep        1         2.66680       0.23614     11.29     <.0001
```

El archivo de salida creado en el proc reg se presenta a continuación. La variable resi (residuos) puede ser utilizada en análisis posteriores, como el estudio habitual de normalidad. Al igual sucede con la variable predi (predicciones para la variable respuesta realizadas a través del modelo de regresión).

Obs	x	y	predi	resi
1	5.11941	16.1821	16.5333	-0.35117
2	8.81973	25.2377	26.4013	-1.16354
3	4.59858	13.6723	15.1443	-1.47197
4	9.13268	27.5105	27.2359	0.27461
5	6.47085	17.9158	20.1373	-2.22146
6	4.06954	15.5703	13.7335	1.83682
7	0.18509	2.9817	3.3744	-0.39269
8	4.18366	15.8307	14.0378	1.79294
9	7.77399	27.6662	23.6125	4.05374
10	1.85863	4.5195	7.8374	-3.31791
11	3.82973	12.4492	13.0939	-0.64478
12	7.35940	24.5850	22.5069	2.07812
13	5.38269	19.7198	17.2354	2.48444
14	6.59170	22.1828	20.4596	1.72322
15	2.12042	9.9384	8.5356	1.40288
16	6.33618	21.3367	19.7781	1.55860
17	8.26369	19.6480	24.9184	-5.27044
18	4.01001	11.1769	13.5747	-2.39778
19	6.00627	22.0373	18.8983	3.13900
20	8.23950	21.7413	24.8539	-3.11262

Capítulo 13. Lenguaje Macro

En lenguaje básico SAS, los pasos DATA y los pasos PROC son, de alguna manera, independientes. La información que pasa de unos a otros sólo lo hace a través de archivos SAS. El lenguaje Macro nos permitirá:

- Generar variables cuyos valores perduran a lo largo de diferentes pasos DATA y PROC
- Utilizar variables de texto que podrán reemplazar texto de programación en pasos DATA y PROC.
- Crear largas aplicaciones donde se ejecutan procedimientos y pasos DATA sujetos a sentencias condicionales o incluidos dentro de bucles.
- Construir subrutinas que serán utilizadas en cualquier momento de un programa SAS.

En general, la utilización de macros en un programa SAS viene reflejada en sentencias que comienzan con el símbolo % (como %LET, %MACRO, %DO, %END…) o con el símbolo &, destinado a diferenciar macrovariables de variables usuales (&n, &peso, &archi1,…).

Macrovariables

Una macrovariable es una variable de cadena cuyo valor puede ser recordado de unos a otros pasos DATA o PROC.

El modo más usual de asignar un valor a una macrovariable es con la sentencia %LET, que puede estar situada en cualquier punto dentro o fuera de pasos DATA o PROC.

Sentencia %Let

Su función es crear una nueva macrovariable o modificar el valor de una macrovariable existente. Hay que insistir en que una macrovariable es una variablde de cadena, es decir, en realidad representa un texto. Su sintaxis general es:

```
%LET macrovariable=valor de cadena;
```

Por ejemplo,

```
%LET raiz=data;
%LET n=310;
```

La utilización de la macrovariable creada consiste en nombrarla con el nombre **&macrovariable** en cualquier momento de los pasos DATA o PROC, de manera que **su valor textual** sustituye virtualmente al texto **&macrovariable** a la hora de ejecutarse el programa correspondiente:

Ejemplo: creación de una macrovariable

El programa

```
%let raiz=dato;
data uno;
&raiz=5;
z=10*&raiz;
run;
```

es equivalente al programa

```
data uno;
dato=5;
z=10*dato;
run;
```

Aunque las macrovariables son en principio variables de cadena, si su valor es un número puede ser tomada como numérica en el programa correspondiente:

Ejemplo: utilización de una macrovariable como constante numérica

El valor de la macrovariable se mantiene constante en la sesión SAS si no se modifica, de modo que una macrovariable que representa una constante de interés puede ser utilizada en diferentes pasos data.

```
%let u=224;

data uno;
 x=5*&u;
 put x=;
run;
```

pondría en el LOG:

```
x=1120
```

pues es equivalente al programa

```
data uno;
 x=5*224;
 put x=;
run;
```

Las macrovariables no se tratan por lo general igual que las variables SAS usuales; su efecto es el de sustitución de texto en un programa, no siendo tratadas como números en general. El siguiente ejemplo ilustra el hecho de que no se puede aplicar la sentencia put sobre una macrovariable, sino la sentencia %put creada específicamente para macrovariables.

Ejemplo: tratamiento incorrecto de una macrovariable

El programa

```
%let n=224;

data uno;
 put &n;
run;
```

daría error en el LOG, por utilizarse la sentencia put, orientada a variables SAS. Sin embargo, el programa que utiliza la macro sentencia %put:

```
data uno;
 %put &n;
run;
```

pondría en el LOG:

```
224
```

A la macrovariable se le pueden concatenar textos al nombrarla, utilizando el caracter punto ".", si la concatenación es a la derecha de la macrovariable. Si la concatenación es a la izquierda no es necesario.

Ejemplo: concatenación de texto combinado con macrovariable

```
%let n=100;
%let r=data;
proc print &r=&r.1 (obs=&n);
run;
proc print &r=&r.2 (obs=&n);
run;
```

Es equivalente al programa

```
proc print data=data1 (obs=100);
run;
proc print data=data2 (obs=100);
run;
```

En resumen, las macrovariables son TEXTO a sustituir en un programa. Son de gran utilidad, por ejemplo, si en varios pasos data se utilizan arrays de cierta dimensión subceptible de ser cambiada por el programador:

Ejemplo: utilización de macrovariables en las dimensiones de arrays

```
%let dim1=10;
%let dim2=30;
%let obse=200;

data uno;
array x{&dim1};
array y{&dim2};
do j=1 to &obse;
  do i=1 to &dim1;x{i}=rannor(0);end;
  do i=1 to &dim2;y{i}=ranuni(0);end;
  output;
end;
run;
proc means data=uno;var x1-x&dim1 y1-y&dim2;output out=sal
mean=mex1-mex&dim1 mey1-mey&dim2;
run;
proc print data=sal;run;
```

El programa genera un archivo con &obse observaciones, &dim1 variables X y &dim2 variables Y. Posteriormente halla sus medias y las incluye como variables en un archivo con una sola observación.

Este programa se puede realizar con el número de observaciones y de variables X,Y que se quiera sin más que cambiar los valores en las sentencias %let situadas al comienzo del programa.

Como se estudió en el proc contents, se puede obtener un listado de las variables presentes en un archivo (de esta manera, utilizando "copiar y pegar" no hay necesidad de escribir manualmente los nombres de las variables). Este listado se asignará a una macrovariable, para realizar procesos sin necesidad de escribir los nombres de todas las variables de interés.

Ejemplo: Crear grupos de variables de interés asignándolos a macrovariables

Creamos el archivo de prueba *uno*:

```
data uno;
input edad sexo altura salario hijos;
cards;
23 1 170 20000 1
28 2 190 15000 0
;
```

Supongamos que nos interesa crear una macrovariable que contenga el listado de todas las variables del archivo *uno*.

En primer lugar, con el proc contents se extrae el nombre de las variables del archivo *uno*:

```
proc contents data=uno out=dos noprint;
run;
```

El proceso de creación de la macrovariable se puede realizar de una manera simple, obteniendo la lista de las variables con

```
data;
 set dos nobs=nume;
 put name @@;
run;
```

En la ventana LOG sale :

```
altura edad hijos salario sexo
```

Se crea la macrovariable con la sentencia %let, copiando y pegando el texto del listado de variables que está en la ventana LOG.

```
%let listavar='altura edad hijos salario sexo';
```

A continuación ya se puede trabajar con esa lista de variables en cualquier procedimiento o macro, indicando &listavar en lugar de escribir el nombre de esas variables. Se puede por lo tanto crear tantos grupos como se desee, asignándolos a macrovariables con la sentencia %let y trabajar con ellos con comodidad.

En el capítulo dedicado a la creación de macros se muestra cómo crear una macro que, dado un archivo, transforma todas sus variables a variables indexadas .

Macrovariables y variables SAS en Pasos Data.

Antes de estudiar sentencias más complejas, es necesario saber cómo relacionar los valores que determinadas variables toman dentro de un paso data, con los valores de macrovariables.

Asignación de valores de macrovariables a variables de un paso data

Para realizar esta operación simplemente se utiliza la sentencia de asignación "=".

Ejemplo: asignar el valor de una macrovariable a una variable SAS en un paso data

```
%let x=10;
data uno;
 y=&x;
 put y=;
run;
```

Pone en el LOG:

```
y=10
```

Asignación de valores de variables de un paso data a macrovariables. Subrutina CALL SYMPUT

A veces nos interesa asignar a una macrovariable el valor que toma cierta variable en un paso data. Pero si por ejemplo ejecutamos el siguiente programa:

```
data uno;
 tiempo=10;
 %let w=tiempo;
run;
```

El valor de la macrovariable &x es, a partir de ahora, el texto "tiempo", y no el valor 10 como en principio nos interesaría.

Para conservar en una macrovariable el valor que determinada variable de un paso data toma en cierto momento, es necesario utilizar la subrutina SYMPUT, que asigna el valor

tomado por una variable del paso data a una macrovariable. Tanto si la variable del paso data es de cadena como si es numérica, la sintaxis básica es

```
CALL SYMPUT (‘macrovariable’, variable);
```

o bien, de manera alternativa pero más precisamente, como se verá:

```
CALL SYMPUT (‘macrovariable’, left(variable));
```

Es importante saber que la macrovariable no es creada realmente hasta que el paso data haya finalizado (después de la sentencia run; por ejemplo), y por lo tanto no se puede utilizar como macrovariable dentro del paso data donde se crea.

Ejemplo: asignar el valor de una variable SAS a una macrovariable en un paso data (1)

Variable alfanumérica

```
data uno;
 nombre=’Pedro’;
 call symput(‘npila’,nombre);
run;
```

A partir de ahora la macrovariable &npila toma como valor Pedro para el resto de la sesión SAS.

Variable numérica

Otro ejemplo, esta vez con una variable numérica, es:

```
data uno;
 edad=21;
 call symput(‘años’,edad);
run;
```

A partir de ahora la macrovariable &años toma como valor 21 para el resto de la sesión SAS.

Ejemplo: asignar el valor de una variable SAS a una macrovariable en un paso data (2)

Si la variable SAS tiene asociado un formato de escritura, esto tiene consecuencias sobre el valor de la macrovariable, pues ésta representa texto.

Si el formato que toma la variable edad es el valor justificado a la derecha 12 espacios, el valor de la macrovariable se escribirá por defecto en las sentencias put y ciertos procedimientos con ese formato:

```
proc format
data uno;
 edad=21;
 call symput(‘años’,edad);
run;

data;
z=&años;
put z;
run;
```

escribiría en el LOG:

```
           21
```

Por lo tanto es más usual utilizar la función SYMPUT con la siguiente sintaxis, utilizando la sentencia left de justificación a la izquierda:

```
CALL SYMPUT(‘macrovariable’,left(variable));

data uno;
 edad=21;
 call symput(‘años’,left(edad));
run;

data;
 z=&años;
put z;
run;
```

Escribiría en el LOG:

```
21
```

En el siguiente ejemplo se presenta uno de los errores más frecuentes en programación básica con macrovariables.

Ejemplo: La macrovariable asignada con CALL SYMPUT no es creada hasta finalizar el paso data

Nótese que el programa

```
data uno;
 edad=21;
 call symput('z',left(edad));
 %put &z;
run;
```

origina un error de compilación, pues la macrovariable &años no se considera creada hasta ser ejecutado el paso data correspondiente (con la sentencia run;) y por lo tanto la sentencia %put &z; no es tratada correctamente.

Sin embargo,

```
data uno;
 edad=21;
 call symput('z',left(edad));
run;

%put &z;
```

o bien

```
data;
 g=&z;
 put g;
run;
```

sí serían correctos.

Creación de Macros

Una macro es una subrutina que realiza una serie arbitraria de pasos DATA y/o pasos PROC. La macro se define por las sentencias situadas entre la sentencia %macro y la sentencia %mend:

```
%macro nombre (parámetros,…);
  Sentencias…
%mend;
```

y se ejecuta en cualquier momento con la sentencia

```
%nombre(parámetros,…);
```

Los parámetros serán tratados como macrovariables en las sentencias de programación presentes dentro de la macro. Es decir, serán tratados como texto y nombrados con el signo previo &. No es necesario utilizar parámetros si no se desea.

Ejemplo: macro con el archivo SAS de interés como parámetro

Crear una macro que calcule con un procedimiento las medias de las 10 variables X1-X10 presentes en el archivo dado como parámetro, y guarde esas medias en el archivo de texto 'c:\medias.txt'.

```
%macro medias(archi);
  proc means data=&archi;var X1-X10;output out=salida mean=med1-
med10;
  run;
  data;
   array med{10};
   set salida;
   file 'c:\medias.txt';
   do i=1 to 10;
     put med{i}=;
   end;
  run;
%mend;
```

Es necesario compilar la macro antes de poder ejecutarla (una compilación por sesión SAS). Simplemente se marca el texto anterior y se ejecuta y la macro quedará compilada.

A continuación, si se desea ejecutar la macro con el archivo SAS llamado disco.datos1 (se supone que este archivo contiene las variables x1, x2,…,x10), se ejecuta la línea:

```
%medias(disco.datos1);
```

Y como consecuencia será creado el archivo de texto 'c:\medias.txt' con las medias de las variables expuestas así:

```
med1=23.5
med2=45.2
…
```

Ejemplo: macro con archivo SAS y lista de variables como parámetros

Realizar el ejemplo anterior tratando también como macrovariables las variables para las que se quieren las medias (listvar=lista de variables) y el número de variables (nvar).

```
%macro medias(archi,listvar,nvar);
  proc univariate data=&archi;var &listvar;output out=salida
  mean=med1-med&nvar;
  run;
  data;
  array med{&nvar};
  set salida;
   file 'c:\medias.txt';
  do i=1 to &nvar;
    put med{i}=;
  end;
  run;
%mend;
```

A continuación, y después de compilar la macro, si se desea ejecutar la macro anterior con el archivo SAS llamado disco.datos1 y las variables x, y, z, edad, ejecuto la línea:

```
%medias(disco.datos1,x y z edad,4);
```

Si deseo ejecutar la macro con el archivo SAS llamado disco.datos1 y las variables x1,…,x15, ejecuto la línea:

```
%medias(disco.datos1,x1-x15,15);
```

Ejemplo: macro para obtener un listado ordenado de correlaciones entre variables de un archivo

Supongamos que disponemos de un archivo con una serie de variables con raíz x e indexadas x1, x2,… y se desea obtener una lista ordenada de la correlación enre esas variables. Se utilizará como base el programa realizado como ejemplo para el proc corr.

```
%macro corres(archi,dim);

proc corr data=&archi outp=dos;
var x1-x&dim;
run;

data tres;
set dos; if _type_='CORR' then output;
run;

data cuatro (keep=i j corre correabs);
array x{&dim};
set tres;
j=_n_;
do i=1 to &dim;
 if i<=j then do;
 correabs=abs(x{i});
 corre=x{i};
 output;end;
end;
run;

proc sort data=cuatro;by descending correabs;run;

data;
 set cuatro;
 if _n_=1 then put 'i' @5 'j' @10 'Abs(Correlación)' @40
'Correlación' //;
 if i ne j then put i @5 j @10 correabs @40 corre;
run;

%mend;
```

Si el archivo de interés se llama *uno* y las variables presentes son X1,…,X10, se ejecutará la macro así:

```
%corres(uno,10);
```

En el ejemplo siguiente se muestra cómo las macros no necesariamente tienen que tener asociados parámetros o macrovariables.

Ejemplo: macro sin parámetros

Supongamos que se desea ejecutar una serie de acciones sobre un archivo SAS que nos es aportado cada cierto tiempo, se llama siempre igual (datos5) y siempre tiene las mismas variables X1-X10.

Supongamos que las acciones consisten en un análisis estadístico básico de todas las variables presentes en el archivo, identificar cuál de ella tiene la media más alta., además de hacer un análisis de correlaciones de todas las variables.

```
%macro basica;
  proc univariate data=datos5;output out=salida mean=med1-med10;
  var x1-x10;
  run;
  data;
   array med{10};
   set salida;
   do i=1 to 10;
     if med{i}>mediamax then do;vari=i;mediamax=med{i};end;
   end;
   put ‘la media más alta corresponde a la variable X‘ vari
   ‘ y es ‘ mediamax;
  run;
  proc corr data=datos5;
  run;
%mend;
```

Suponiendo que el archivo SAS datos5 existe con las variables X1-X10, la macro se ejecutaría así:

```
%basica;
```

En el ejemplo anterior no es necesario crear la macro, se puede ejecutar el texto interior cada vez que se necesiten realizar esas acciones, pero es más cómodo tenerlas organizadas en forma de macro con un nombre específico.

La razón es que a menudo se desean realizar una serie de acciones rutinarias cada cierto tiempo. Si estas acciones ocupan mucho texto y existe por ejemplo un archivo de texto donde estas acciones están organizadas en forma de varias macros o subrutinas, para el usuario es más cómodo compilar todas las macros al iniciar la sesión y cada vez que se necesite ejecutar una de ellas simplemente escribir el nombre de la macro.

En lugar de utilizar los nombres descriptivos de referencia de las variables, es frecuente la necesidad de utilizar nombres indexados del tipo x1,x2,…,xn. La siguiente macro transforma un archivo de manera que las variables quedan renombradas a variables indexadas.

Ejemplo: Transformar una lista de variables a variables indexadas en un archivo SAS

Se recurrirá al ejemplo anteriormente visto para crear una macrovariable con la lista de variables presentes en el archivo. Se utilizará el mismo archivo de ejemplo *uno*.

La macro *cambio* tiene como parámetros el archivo de entrada, el de salida, el número de variables y la raiz de las variables indexadas. Es necesario previamente haber definido la macrovariable &listavar con la lsita de variables de interés.

La sentencia scanq para variables alfanuméricas extrae las palabras separadas por espacios en blanco (en este caso los nombres de las variables), que son asignadas a las sucesivas macrovariables con nombres &var1, var2, etc. en la sentencia call symput (donde se ha utilizado la concatenación || que añade al número i al texto var.

El segundo paso data de la macro crea el archivo de salida, renombrando las variables a través de un bucle. Véase como se concatenan los textos: el texto simple mediante un punto, en &raiz.1, y el texto de varias macrovariables directamente, como &raiz&i.

En este segundo paso data, es necesario utilizar la notación &&var&i indicando el valor de las macrovariables var1,var2, etc. pues la expresión &var&i significaría concatenar el valor de la macrovariable &var con el de la macrovariable &i, dando error pues la macrovariable &var, puramente llamada, simplemente no existe.

```
%macro cambio(entrada,salida,nvar,raiz);
data;
do i=1 to &nvar;
 vari=scanq(&listavar,i," ");put vari=;
 call symput('var'||left(i),left(vari));
end ;
run ;

data &salida (keep=&raiz.1-&raiz&nvar);
set &entrada;
%do i=1 %to &nvar;
  &raiz&i=&&var&i;
%end ;
run;
%mend;
```

Si se quiere ejecutar la macro sobre el archivo uno, obteniendo las variables indexadas z1 a z5, previamente a la macro se define la macrovariable &listavar:

```
%let listavar='edad sexo altura salario hijos';
%cambio(uno,sal,5,z);
```

Para observar el contenido del archivo de salida:

```
proc print data=sal;run;
```

obteniendo en la ventana OUTPUT:

Obs	z1	z2	z3	z4	z5
1	23	1	170	20000	1
2	28	2	190	15000	0

Macros con parámetros definidos explícitamente

Otra manera de nombrar los parámetros dentro de la macro es utilizando el signo "=": En un ejemplo anterior, se puede definir la macro de la siguiente manera:

```
%macro medias(archi=,listvar=,nvar=);
```

Y a la hora de ejecutar la macro, escribiendo

```
%medias(archi=disco.datos1,listvar=x y z edad,nvar=4);
```

Relación entre macros y Pasos Data.

Las mayores confusiones en el manejo de macros se producen cuando se combinan Macros con Pasos Data. El orden de compilación y ejecución depende de los programas que estén anidados, pero hay dos cuestiones básicas que generan gran parte de los errores más comunes:

Los Pasos Data no se ejecutan hasta que se encuentra el final del Paso Data (es decir, alguna de las sentencias RUN, PROC, CARDS o DATA). Como consecuencia, hay que cerciorarse de cerrar apropiadamente el Paso Data en cada caso en el lugar que convenga. Lo más importante desde el punto de vista de la operación con macros es que las referencias a macrovariables dentro del mismo paso data en que estas macrovariables han sido creadas, dan error de compilación. Si se desea utilizar esas macrovariables en un paso data, es necesario cerrar el actual paso data (para que sea ejecutado y por lo tanto asignado el valor a las macrovariables) y abrir otro paso data para operar con las macrovariables en cuestión, que ya existen y tienen valor asignado.

Entorno de referencia

Cuando se trabaja con macros que involucran pasos DATA, es conveniente definir el entorno donde van a ser utilizadas las macrovariables mediante las sentencias

%GLOBAL macrovariables

%LOCAL macrovariables

Las macrovariables definidas como globales conservarán sus valores a lo largo de diferentes pasos DATA y PROC y otras macros. Las macrovariables definidas como locales sólo conservarán su valor dentro de la macro en que están definidas.

En general, se puede utilizar el comando %GLOBAL macrovariables por defecto, salvo cuando en algún caso queremos utilizar el mismo nombre de macrovariable para ser cambiada

de valor fuera de la macro con la sentencia %LET o en otra macro. En estos últimos casos se utilizará la sentencia %LOCAL macrovariables.

Las macrovariables creadas dentro de una macro suelen quedar definidas en el entorno Local, por lo que no son reconocidas fuera de la macro. Conviene aplicarles la sentencia %Global si se desea utilizarlas en procedimientos o Pasos Data fuera de la macro.

Estas dos cuestiones se tendrán en cuenta a la hora de construir un programa en el que se combinan macros y Pasos Data , dependiendo de la estructura del programa que planteamos:

Macro previa al Paso Data

En este caso si se desean utilizar las macrovariables creadas en la macro en el paso data presente a continuación, es necesario añadir la sentencia %GLOBAL nombrando las macrovariables de interés:

```
%global…
%macro muno;
…
%mend muno;

%muno;

data…;
…
run;
```

Paso Data previo a la Macro

```
data…;
…
run;

%macro muno;
  …
%mend muno;

%muno;
```

En este caso, si se van a utilizar dentro de la macro macrovariables creadas en el paso data a través de la sentencia CALL SYMPUT, hay que cerciorarse de haber cerrado el Paso Data antes de la ejecución de la macro, con la sentencia RUN como en este caso o con otras sentencias de fin del Paso Data.

Macro con Paso Data interno

```
%macro muno;
…
  data…;
    …
```

```
  run;
…
%mend muno;

%muno;
```

En este caso es necesario controlar también el fin del Paso Data (no es conveniente dejarlo sin cerrar dentro de la macro). Si se desean introducir sentencias macro sobre macrovariables como por ejemplo las sentencias condicionales %IF-%THEN-%ELSE, dentro del propio Paso Data, hay que tener cuidado de que **no** versan sobre macrovariables **creadas dentro de ese Paso Data** (véase la siguiente Sección).

Sentencias Macro Básicas

Las sentencias macro presentadas a continuación pueden aparecer en cualquier momento en el interior de una macro. Se aplican a macrovariables que deben estar bien definidas. Es importante recordar que las sentencias que se verán en este capítulo no pueden ser utilizadas “por libre”, es decir, fuera de una macro. Tienen que estar situadas entre una sentencia %macro y una sentencia %mend.

Sentencias condicionales %IF-%THEN-%ELSE

Dentro de una macro pueden utilizarse las sentencias condicionales

```
%IF (condición lógica sobre macrovariables)
%THEN (sentencias ejecutables);
%ELSE (sentencias ejecutables);
```

Estas pueden englobar pasos DATA y PROC, a diferencia de las mismas sentencias usualmente utilizadas dentro de un paso data. Las condiciones lógicas se impondrán sobre macrovariables.

Para condicionar a la realización de un bloque de programación , se utilizarán las sentencias macro :

```
%IF (condición lógica sobre macrovariables) %THEN
  %DO;
   (sentencias ejecutables);
  %END;
```

Ejemplo: sentencias macro condicionales (1)

Crear una macro que lea un archivo, controle su número de observaciones y realice un proc means si el archivo tiene más de 20 observaciones y un proc print si el archivo tiene menos de 20 observaciones.

```
%macro obser(archi);
  data;
    set &archi nobs=numero;
    call symput('nume',left(numero));
  run;
  %if &nume>20 %then %do;
       proc means data=&archi;run;
      %end;
  %else %do;
    proc print data=&archi;run;
   %end;
%mend;
```

Si queremos ejecutarla con el archivo uno, ejecutaremos

```
%obser(uno);
```

Ejemplo: sentencias macro condicionales (2)

Crear una macro que lea un archivo SAS con una sola observación y, dependiendo del valor de la variable X,

1) Si X >0 realiza un paso DATA donde se genera aleatoriamente un archivo con 100 observaciones y la variable Z=Normal con la media y desviación típica dadas, como parámetros en la macro.

2) Si X=0 ejecuta el procedimiento print sobre el archivo original.

3) Si X<0 crea un archivo de texto con el nombre dado como parámetro en la macro, con el valor actual de X.

```
%macro triple(archi,media,dtip,texto);
data;
   set &archi;
   call symput('y',left(x));
   if x<0 then do;file &texto;put x;end;
run;
   %if &y>0 %then %do;
  data genera;do i=1 to 100;z=rannor(0)*&dtip+&media;output;end;run;
   %end;
   %if &y=0 %then %do;proc print data=uno;run;%end;
%mend;
```

Si se quiere ejecutar la macro sobre el archivo uno, con parámetros 3 y 4 para la generacion de la Normal y para el archivo de texto 'dat.txt':

```
%triple(uno,3,4,'c:\dat.txt');
```

En el anterior ejemplo, hay que hacer ciertas puntualizaciones. Para la primera sentencia condicional

```
if x<0 then do;file &texto;put x;end;
```

no se utiliza el lenguaje macro (sentencia %if) , pues se utilizan sentencias usuales del paso data sobre variables sas. Si sustituimos esa línea de programa por la línea

```
%if &y<0 then %do;file &texto;put x;%end;
```

obtenemos un error de compilación de macro, pues la macrovariable 'y' está creada dentro de ese paso data y al no haberse llegado todavía a la sentencia de fin del paso data no se pueden establecer operaciones macro con la macrovariable &y.

En cada caso de las tres condiciones planteadas, el paso data inicial está correctamente cerrado puesto que en la primera condición

```
data…
….
if x<0 then do;file &texto;put x;end;
run;
```

existe a continuación una sentencia run. En la segunda condición hay un nuevo paso data cerrado con un run:

```
%if &x2>0 %then %do;
   data genera;do i=1 to 100;y=rannor(0)*&dtip+&media;output;end;run;
   %end;
```

y en la tercera un paso proc también cerrado.

```
%if &x2=0 %then %do;proc print data=uno;run;%end;
```

Esto garantiza que al ejecutar la macro no quedan ejecuciones de pasos data pendientes ni macrovariables no asignadas.

Evaluación numérica de macrovariables

Una restricción importante sobre las macrovariables es que la evaluación de cualquier condición lógica sobre sus valores numéricos sólo actúa sobre valores enteros.

Ejemplo: evaluación numérica de macrovariables

El programa

```
%let x=5;
%let y=10;
%macro uno;
  %if &x<&y %then %put X es menor que Y;
  %else %put X es mayor que Y;
%mend;
%uno;
```

Tiene por resultado en el LOG:

```
X es menor que Y
```

Sin embargo, si cambiamos las dos primeras sentencias por

```
%let x=5.0;
%let y=10.1;
```

el programa tiene por resultado en el LOG:

```
X es mayor que Y
```

pues las condiciones lógicas sobre números y evaluaciones numéricas sólo pueden plantearse sobre valores enteros. Tanto el valor 5.0 y el valor 10.1 son tomados como valores de variable de cadena y no como números debido al punto, de ahí que se considere X mayor que Y (pues 5.0>10.1 en el orden léxicográfico).

Si en algún programa se desea establecer una condición numérica complicada sobre macrovariables se debe utilizar un paso data y una variable SAS usual como variable de paso.

Ejemplo: sentencia condicional sobre valores numéricos de una macrovariable

Si queremos plantear la condición sobre si el valor de log(&x+&y) será mayor o menor que 2.34, para realizar un proc print del archivo *datos* condicionado a esa condición, se utilizará un paso data y la variable de paso SAS que hemos llamado c y la macrovariable de paso &z:

```
%macro uno;
  data;
    if log(&x+&y)>2.34 then c=1;else c=0;
    call symput('z',left (c));
  run;
```

```
  %if &z=1 %then %do;proc print data=datos;run;%end;
%mend;
```

Para ejecutar la macro, cada vez que se aporten valores a &x e &y en las sentencias %let, por ejemplo:

```
%let x=5.32;
%let y=10.67;

%uno;
```

Sentencias %DO-%END

Se pueden realizar bucles sobre macrovariables. Estos bucles se pueden utilizar dentro de una macro, con la sintaxis

```
%DO macrovariable=inicial %TO  final %BY incremento;
(sentencias ejecutables);
%END;
```

Es conveniente insistir en que la variable índice es una macrovariable.

Ejemplo: creación de varios archivos indexados a través de macrovariables

Queremos crear tres archivos SAS diferentes con diferente nombre y diferentes variables:

```
%macro crear;
%do i=1 %to 3;
   data archi&i;
      do j=1 to 10;x&i=rannor(0)*&i;output;end;
   run;
%end;
%mend;
```

La macro se ejecuta así:

```
%crear
```

El programa anterior crea los archivos archi1, archi2 y archi3 con 10 observaciones y las variables respectivas x1 (Normal(0,1)), x2(Normal(0,2)), x3(Normal(0,3)). Es decir, ejecuta 3 pasos data diferentes.

Ejemplo: listado sucesivo de varios archivos indexados utilizando macrovariables

Queremos realizar un procedimiento print con cada uno de los archivos data1, data2, data3,…,data 10:

```
%macro lista;
%do i=1 %to 10;
   proc print data=data&i;run;
%end;
%mend;
```

Para ejecutar la macro:

```
%lista;
```

Ejemplo: utilización de un mismo procedimiento sobre una serie de archivos no indexados (1)

En el ejemplo anterior, supongamos que los nombres de los archivos a presentar mediante el *proc print* están dentro del archivo SAS *archis*, en la variable de cadena *arc:*:

```
data archis;
 input arc $ @@;
cards;
uno dos datos3 semanal
;

%macro lista;
%do i=1 %to 4;
  data;
    x=&i;
    set archis point=x;
    call symput('archivo',arc);
    stop;
  run;
  proc print data=&archivo;run;
%end;
%mend;


%lista;
```

Es importante señalar que no se podrían leer dentro del paso data todas las observaciones del archivo *archis* de manera secuencial (con la sentencia set usual) para realizar un proc print con cada una de ellas, debido a que no se puede englobar un proc print dentro de un paso data (en cuanto aparece la sentencia proc el paso data finaliza). El programa creado ejecuta por lo tanto 4 pasos data diferentes.

Por otra parte, la sentencia

```
set uno point=x;
```

sólo permite variables SAS (la variable x en este caso), y no valores constantes. Como las macrovariables son tratadas como texto (en este caso &i toma valores 1, 2, 3, 4) no hemos podido poner directamente

```
set uno point=&i;
```

pues esta sentencia equivaldría,por ejemplo en la primera iteración, a la sentencia

```
set uno point=1;
```

que no está permitida en la sintaxis SAS de la sentencia set *archivo* point=*var*. De ahí que se pase la macrovariable '&i' a la variable SAS del paso data 'x'.

El programa anterior puede ser simplificado utilizando la función %qscan para macrovariables, que funciona de manera similar a la función scanq sobre variables SAS, es decir, va leyendo uno por uno los elementos de la lista.

Ejemplo: utilización de un mismo procedimiento sobre una serie de archivos no indexados (2)

```
%macro lista;
%do i=1 %to 4;
 %let archivo=%qscan(&listado,&i," ");
 proc print data=&archivo;
 run;
%end;
%mend;


%let listado=uno dos datos3 semanal;
%lista;
```

Bibliografía

R. Aster, R. Seidman. Professional SAS Programming Secrets. McGraw Hill.

E. W. Tilanus. Working with the SAS System. SAS Institute.

P. E. Spector. SAS Programming for Researchers and Social Scientists. Sage Publications.

M. M. Burlew. SAS Macro Programming made easy.

A. Carpenter. Carpenter’s Complete Guide to the SAS Macro Language. SAS Institute.

Ron Cody, Ray Pass. SAS Programming by Example. SAS Institute.

Pérez, César. El sistema estadístico SAS. Prentice-Hall.