Data Manipulation in `DATA` Step

STAT2005

Chapter 10

Data manipulation

- This chapter is about data manipulation in `DATA` step. We will consider data transformation and assignment statements that modify a variable or assign a new variable. We shall also introduce conditional execution.
- In those manipulations, we shall use special operators and built-in functions in SAS. We start by introducing these operators and functions.

Basic SAS operators

	Symbol	Alt. symbol	Meaning
Arithmetic operators :	+		(prefix) makes value positive, e.g. +2
	-		(prefix) makes value negative, e.g. -2
	**		Exponentation, e.g. 2**4 (means 2^4)
	*		Multiplication, e.g. 2*4 gives 8
	/		Division, e.g. 4/2 gives 2
	+		Addition, e.g. 4+2 gives 6
	-		Subtraction, e.g. 4-2 gives 2
	<>		Maximum, e.g. 2<>4 gives 4
	><		Minimum, e.g. 2><4 gives 2
Character string operation:	||		Concatenation, e.g. 'ab'||'cde' gives 'abcde'

	Symbol	Alt. symbol	Meaning
Comparison Operators :	=	EQ	Equal, e.g. a = b , a eq b (gives value TRUE if and only if a = b)
	^=	NE	Not equal, e.g. a ^= b, a ne b (gives value TRUE if and only if a is not equal to b)
	>	GT	Greater than, e.g. a > b, a gt b
	>=	GE	Greater than or equal, e.g. a >= b, a ge b
	<	LT	Less than, e.g. a < b, a lt b
	<=	LE	Less than or equal, e.g. a <= b, a le b
Logical (Boolean) operators :	^	NOT	(prefix) negation, e.g. ^(a+b = 4)
	&	AND	And, e.g. a = 1 & b = 2 , a = 1 and b = 2
	|	OR	Or, e.g. a = 1 | b = 2, a = 1 or b = 2
Other operator :		IN	List membership, e.g. a in (6, 7, 8) (it gives TRUE value if and only if a is 6, 7 or 8), region in ('NE', 'W', 'S')

Remarks

(1) SAS does not have logical variable. The true/false values are stored as an internal numeric value:

TRUE: 1 FALSE: 0

Rules to use numeric values as logical values:

- Any numeric expression can be used as a logical expression. A value of zero (or missing) is considered FALSE. A value of one (or any nonzero value) is considered TRUE.

The assignment statement `A = B = C;` assigns `A` to `1` if `B = C` and assigns `A` to `0` if `B ^= C`.

Note that "`A = B = C;`" itself is a statement as we have a semicolon. If "`A = B = C`" is an expression in a statement such as `IF A = B = C then D = 1;`, then `A = B = C` means `(A = B) & (B = C)`.

(2) ">=" can also be expressed as "=>" and "<=" can be expressed as "=<".

(3) When an expression contains compact comparison, such as "`21 < age < 45`", the comparisons are connected by `AND`. Therefore,

`21 < age < 45` means `(21 < age) & (age < 45)`

`A = B = C = D` means `(A = B) & (B = C) & (C = D)`

(4) If an operator is asked to work with a missing value, the result of the operation will be missing. Therefore `(4 + b)` gives a missing value if `b` is missing.

(5) Converting character value to numeric value or numeric value to character value:

When a character constant or character variable value is involved in numerical calculation, SAS will try to convert that character value to its corresponding numeric value.

Therefore, character value `'10'` will become numeric `10`, character value `'-1'` will become numeric `-1`, and character value `'1e2'` will become numeric `100`.

When the conversion is not possible, SAS will set the value to missing.

SAS will also convert numeric value to its corresponding character value when the numeric value is used in place where a character value is expected. SAS may insert blanks before the number when converting it to its character value.

In general it is not recommended to make use of the automatic conversion between character value and numeric value in SAS programming since even experienced programmer may make mistake.

SAS built-in functions `INPUT` and `PUT` can perform character conversion explicitly.

Example 1

Consider the following program.

```
DATA AA;
X=2; Y = '1E2'; W='A123456789012'; T='K';
Z=X+Y; W=T||X; X = Y;
RUN;
* Print the data set;
PROC PRINT;
RUN;
```

Output:

Obs	X	Y	W	T	Z
1	100	1E2	K 2	K	102

In the log window, we will see the following notes:

```
NOTE: Character values have been converted to numeric
values at the places given by:
      (Line):(Column).
      323:5    323:20
NOTE: Numeric values have been converted to character
values at the places given by:
      (Line):(Column).
      323:13
```

It is interesting to see that for the value of `W`, there are blanks between `K` and `2`. The blanks are inserted when SAS converts numeric value to character value.

If we change `W = 'A123456789012';` to `W = 'A12345678901';`, the value of `W` becomes `"K"` because the storage assigned to `W` is not long enough to store the value `2` and therefore `2` is truncated.

Note: There are 12 character widths for `X` in `T||X`.

Remarks (continued)

(6) Numeric missing values are always smaller than any non-missing value. Character missing values usually compare low.

(7) Expressions are evaluated as follows:

Subexpressions within parentheses `( )`: numerical expression in the parentheses are evaluated first.

e.g., `2*(4+1)` gives `10`. Use parentheses whenever we do not have full confidence on the order of evaluation.

Note that we cannot use `[ ]` and `{ }`.

Unless using parentheses, the order of evaluation is given in the following table:

Operator	Associativity
Function terms (such as SIN, LOG and SQRT)	
** , + (prefix) , -(prefix), NOT, <>, ><	Right to left
* , /	Left to right
+ (infix), - (infix)	Left to right
||	Left to right
Comparisons (such as >, >=)	Left to right
AND	Left to right
OR	Left to right

Operators in the upper row have higher precedence than the operators in the lower row.

An expression involving operators of equal precedence is resolved by reference to the column labeled associativity.

For example, for `*` and `/`, the operator on the left will be executed first. Therefore,

`4 - 1 + 2` means `(4 -1)+2` that gives `5`.

`4 / 2 * 2` means `(4/2)*2` that gives `4`.

`-2**4` means `-(2**4)` that gives `-16`.

`- 2 >< 1` means `-(2><1)` that gives `-1`.

Example 2

Determine the order of the operations and the result of the following expressions when `a = 1, b = 2, c = 3, d = 4, aa = 'abc'`.

```
Expression: a**2/b+c**2*d-2
Order:        2  3 5  1 4 6
Value:  34.5

Expression: (a = b & c < d) | a<=d
Order:          1   3   2    5  4
Value:  1 (true)
```

```
Expression: (b ^= c) = a
Order:          1     2
Value:  1 (true)
```

Note that `b ^= c` gives value `1`.

Special attention must be taken when we have more than one comparison put together.

For example `1 ^= 3 = 3` gives value `1` (true) as it is treated as a compact comparison `(1 ^= 3) & (3 = 3)`.

However, both `(1 ^= 3) = 3` and `1 ^= (3 = 3)` gives value `0`.

```
Expression: aa ^= 'a' || "bc"
Order:          2      1
Value:  0 (false)

Expression: a +(b > c)
Order:        2    1
Value:  1
```

because `b > c` is `0` (false)

```
Expression: 4**sin(2*a-b)/5
Order:       4  3   1 2  5
Value:  0.2
```

```
Expression: a + sqrt(d) ** 1 <> b
Order:        4   1     3    2
Value: 5
```

```
Expression: ^ c - 1
Order:      1   2
Value:     -1
```

```
Expression: c = 1 | 2
Order:        1   2
Value: 1
```

We can check the results of the above expressions using the following program. The `PUT` statement writes lines to the SAS log

```
DATA TESTING;
a=1; b=2; c=3; d=4; aa='abc';
result = expression;
PUT result=;
RUN;
```

Assignment statement

- Assignment statement allows us to create new variables or modify existing variables.
- The syntax for the assignment statement is

```
target = expression;
```

where

`target` is the variable being created or replaced to store the value calculated from the expression.

`expression` is a string of constants, variables, operators, functions, and parentheses arranged in a meaningful way.

- The target variable can appear in the expression, so that `x = x / 4 + 1;` is valid. Therefore, if `x = 4`, then the new x-value is `4/4+1 = 2`.

Example 3

```
residual = observed - expected;
loan_val = prin * (1 + r)**time;
flag = a eq b;
* set flag to 1 if a = b, and 0 otherwise;
name = 'Mr '||name;
* add 'Mr ' before the name.;
```

Note that the variable `name` appears in both the left and the right hand sides of the = symbol. We must be careful in using this kind of statement with character string.

For example if `name` is a character variable of length 6, and `name = 'Lee Yi'`, then the new value of `name` becomes `'Mr Lee'`.

Variable list

Some SAS functions require multiple input arguments, and we need to group them in a variable list.

A simple way to give a variable list is to list all the variables, such as

```
NAME AGE INCOME
```

There are some shortcut forms of defining a variable list:

(1) `varm - varn : var` is user-supplied character string, and `m` and `n` are positive integers such that `n` is larger than `m`. All variable with name implied by the list are included,

e.g. `x2 - x5` means `x2 x3 x4 x5`

`ABC3 - ABC6` means `ABC3 ABC4 ABC5 ABC6`

(2) `vara -- varb` : All variables found physically between `vara` and `varb` in the data set are included, e.g. `age -- income`.

The order of variables in an SAS data set is determined by how the variables were first made known to SAS.

(3) `vara -NUMERIC- varb` : All numeric variables found between `vara` and `varb` in the data set are included,

e.g. `age -numeric- income`

(4) `vara -CHARACTER- varb` : All character variables found between `vara` and `varb` in the data set are included,

e.g. `name -character- grade`

(5) `_CHARACTER_` : All character variables in the data set

(6) `_NUMERIC_` : All numeric variables in the data set

(7) `_ALL_` : All variables in the data set

A variable list can be a mixture of the above forms,

e.g. `name Test1 - Test5 age -- final`

Example 4

Suppose a SAS file contains variables in the following order.

Type	Num.	Char.	Num.	Num.	Num.	Char.	Char.
Variable Name	C5	C7	C4	C2	C1	C3	C6

What are the variables in the following variable list?

i. `C5-CHARACTER-C7`

ii. `_ALL_`

iii. `C7-NUMERIC-C2`

iv. `_CHARACTER_`

v. `C2-C6`

vi. `C2--C6`

vii. `C2--C6 C1`

viii. `C4-CHARACTER-C1`

Answer

i. `C7`

ii. `C5 C7 C4 C2 C1 C3 C6`

iii. `C4 C2`

iv. `C7 C3 C6`

v. `C2 C3 C4 C5 C6`

vi. `C2 C1 C3 C6`

vii. `C2 C1 C3 C6 C1`

viii. no variable in the list.

SAS built-in functions

SAS built-in functions have the following forms:

(i) If it has only one argument:

```
funcname(arg)
```

(ii) If it has more than one argument:

```
funcname(arg {,arg} ...)
```
or
```
funcname(OF arg_list)
```

The first expression in (ii) is used when the arguments are listed one by one, and are separated by commas, e.g. `sum(x1, x2, x3).`

The second expression in (ii) is used when the arguments are given as a list of arguments (no commas), e.g. `sum(OF x1 – x3)`, `sum(OF x1 x2 x3)`, and `sum(OF _numeric_).`

Note that `_numeric_` means all the numeric variables in the data set.

Arithmetic functions

`SQRT(arg)` : square root of `arg` (`arg ≥ 0`), e.g., `sqrt(9)` gives `3`.

`ABS(arg)` : absolute value of `arg`, e.g., `abs(-5)` gives `5`; `abs(5) = 5`.

`SIGN(arg)` : return

`-1` when `arg < 0`;

`0` when `arg = 0`;

`1` when `arg > 0`.

e.g., `sign(4)=1`, `sign(-4) = -1`, `sign(0) = 0`.

`MOD(arg1,arg2)` : the remainder value from dividing `arg1` by `arg2`, e.g., `mod(11,3)=2`.

`MIN([OF] arglist)`: minimum value of the arguments in `arglist`, e.g., `min(a, b, c, d)`, `min(of a -- c d)`.

`MAX([OF] arglist)`: maximum value of the arguments in `arglist`, example: `max(a, b, c)`, `max(of c a -numeric- b)`.

`SIN(arg)`: sine of the `arg` which is expressed in radians, e.g., `sin(2.5)`.

`COS(arg)`: cosine of the `arg` which is expressed in radians, e.g., `cos(1.2)`.

`TAN(arg)`: tangent of the `arg` which is expressed in radians, e.g., `tan(1.3)`.

`ARSIN(arg)`: arcsine of `arg (-1≤arg≤1)`, e.g., `arsin(0.8)`.

`ARCOS(arg)`: arccosine of `arg` (`-1≤arg≤1`), e.g., `arcos(0.6)`.

`ATAN(arg)`: arctangent of `arg`, e.g., `atan(3)`.

`EXP(power)`: exponential function, e.g., `exp(3)`.

`LOG(arg)`: natural logarithm of `arg`, e.g., `log(5)`.

`FLOOR(arg)`: largest integer less than or equal to `arg`, e.g., `floor(-3.8)` gives `-4`.

`ROUND(arg [, amount])`: round `arg` to the nearest `amount` (`amount > 0`). The default value for `amount` is `1`. e.g., `round(3.23, 0.1) = 3.2`, `round(2.23) = 2`, `round(8.613,0.5) = 8.5`.

Statistical functions

`N([OF] arglist)`: number of arguments in `arglist` with non-missing values (the arguments must be numeric). e.g., `count = n(of round1 - round10); no = n(a, b, c);`

`NMISS([OF] arglist)`: number of arguments in `arglist` with missing values (the arguments must be numeric). e.g., `count = nmiss(of round1 - round10); no = nmiss(a, b, c);`

`SUM([OF] arglist)`: sum of the arguments in `arglist`, e.g., `average = sum(a, b, c)/n(a, b, c);`

`MEAN([OF] arglist)`: average of the arguments in `arglist`, e.g., `wtavg = 0.7 * mean(of chi --math) + 0.3 * mean(geo, chem); avg = mean(of_numeric_);`

`STD([OF] arglist)`: standard deviation of the arguments in `arglist`, e.g. `s = 0.7 * std(of chi -- math) + 0.3 * std(geo, chem);`

`PROBBNML(p,n,x)`: $\Pr(B(n,p) \leq x)$, where $B(n,p)$ is a random variable with binomial distribution with parameters n and p. e.g., `probbnml(0.6,10,3)` gives `0.05476`.

`PROBNORM(arg)`: the standard normal distribution function, e.g. `p = probnorm(1.96)` gives `0.9750021049`.

`PROBIT(p)`: the inverse standard normal distribution function, i.e., the value x such that $\Pr(N(0,1) \leq x) = p$, where $N(0,1)$ is a standard normal random variable. e.g., `probit(0.975)` gives `1.9599639845`.

Random number generation

An important argument in any random number function is the random seed that can be any positive integer less than $2^{31} - 1$.

If we set seed to 0 or a negative integer, the system time will be used as the seed. Therefore, we will get different sequence of random numbers in different runs of the program if we set seed to 0.

`RANBIN(seed,n,p)`: generate binomial random number with parameters n and p, e.g., `y = ranbin(0,5,0.6);`

`RANNOR(seed)`: generate standard normal random number, e.g., `nor = rannor(12345);`

`RANUNI(seed)`: generate Unif(0,1) random number, e.g., `unif = ranuni(12333);`

Date and time functions

`TODAY()` : returns the current date from the system as a SAS date value

`TIME()` : returns the current time of day from the system clock.

`YEAR(arg)` : returns a numeric 4-digit year value from a SAS date value, e.g., `year(14255)` gives `1999` (note that `14255` is the numeric value representing the date `11JAN1999`).

`MONTH(arg)` : returns a numeric month value (`1` through `12`) from a SAS date value, e.g., `month(14255)` gives `1` which means that the date is in January.

`DAY(arg)` : returns the day of the month from a SAS date value, e.g. `day(14255) = 11`.

`MDY(month, day, year)`: returns the numeric value for a day, e.g. to find the numeric value for `11JAN1999`, we can use `mdy(1, 11, 1999)` which gives `14255`.

Another way to find the numeric value for `11JAN1999` is `'11jan1999'd`, note that we can get the same result using `'11Jan1999'd` or `'11JAN1999'd`.

Similar to the suffix `d`, we can use suffix `t` to get the time and `dt` to get date and time.

e.g., `'10:30't` is `37800`, which is (10×60+30)×60 seconds from `00:00:00`, and

`'2Jan1960:10:30'dt` is `124200`, which is 24×60×60 + 37800 seconds from `1Jan1960:00:00:00`.

`WEEKDAY(arg)`: returns a numeric day of the week (gives value 1 for Sunday, 2 for Monday, 3 for Tuesday, etc.)

e.g., `weekday(mdy(2,19,2008))` gives a value `3` meaning `19Feb2008` it is a Tuesday.

Character functions

`LENGTH(string)`: the position of the rightmost non-blank character in the string. Note that `length(' ')` is `1` rather than `0`. `length(' a bcd ')` gives `6`.

`LOWCASE(string)`: converts all uppercase characters in the string to lowercase, e.g. `lowcase('aB, c')` gives `'ab, c'`.

`UPCASE(string)`: converts all lowercase characters in the string to uppercase, e.g. `upcase('aB, c')` gives `'AB, C'`.

`INDEX(source,arg)`: Searches the source for the character or character string specified in the argument. A value of `0` is returned when no characters are found. Otherwise, a value corresponding to the position of the first character found is returned. E.g., `INDEX('abcdefg','de')` gives `4` as `'de'` is found in the 4th position of `'abcdefg'`.

`INDEXC(source, arg)`: Searches the source for the first occurrence of **any** character in the `arg`. A value of `0` is returned when no characters are found. Otherwise, a value corresponding to the position of the first character found is returned. E.g., To find the position of the first digit in `PASSWORD`, we use `INDEXC(password,'0123456789')`.

`INDEXW(source,arg)`: Searches the source for a specified word. A value of `0` is returned when no such word is found. E.g., `indexw('actor or', 'or')` gives `7` as the `'or'` in `'actor'` is not a word.

`SUBSTR(arg, start [,length])`: Extracts a portion of a character string expression from the position `start` with given `length`. If `length` is omitted, it continues extracting character until the end of the string. If `A='abcdef'`, then `B=substr(A,3,2)` gives `B = 'cd'` (start extracting substring of length 2 from the 3rd character of `A`), and `B=substr(A,3)` gives `B = 'cdef'`.

`TRIM(arg)`: Removes trailing blanks from a character string expression, e.g., `TRIM(' ab c ')` gives `' ab c'`.

`SCAN(arg, n)` : Returns the n-th word from a character string. The default delimiters are as follows.

`blank ! $ % & ( ) * + , - . / ; < ^ |`

- If `n` is positive, `SCAN` counts words from left to right in the character string.
- If `n` is negative, `SCAN` counts words from right to left in the character string.

E.g., if `A='black and white'`, then `A1=SCAN(A,1)` gives `A1='black'`, `A2=SCAN(A,2)` gives `A2 = 'and'`, `A3=SCAN(A,-1)` gives `A3='white'`.

If `B='black & white'`, then `B1=SCAN(B,1)` gives `B1='black'`, `B2=SCAN(B,2)` gives `B2 = 'white'`, `B3=SCAN(B,-1)` gives `B3='white'`.

The following two statements, that use different formats, are equivalent. They both set `A` to be the sum of `x1`, `x2` and `x3`.

```
a = sum (of x1 - x3); a = sum(x1, x2, x3);
```

Some remarks:

- Notice that `a = sum(x1 - x3)`, whose meaning differs from the above statements, sets `a = x1 - x3`. The "–" sign here means subtraction because SAS treats "`x1 - x3`" as a single argument rather than a variable list.
- Functions which compute descriptive statistics use only non-missing numeric values. These functions return a missing value only if all arguments are missing. Therefore, the following two statements are different.

  ```
  A = SUM(B,C); A = B+C;
  ```

  If `B` is observed and `C` is missing, the first statement assigns the value of `B` to `A`, while the second statement assigns a missing value to `A`.

Example 5

```
DATA ;
INPUT C1 - C3;
A1 = LOG(C1) - 1; A2 = MEAN(OF C1 - C3); C3 =
MIN(A2, C3+2);
CARDS;
. 64 56
3 30 .
4 22 45
RUN;
```

The data set is

Obs	C1	C2	C3	A1	A2
1	.	64	58.0000	.	60.0000
2	3	30	16.5000	0.09861	16.5000
3	4	22	23.6667	0.38629	23.6667

Notice that the variable C3 is modified because it appears in the right hand side and the left hand side of the last assignment statement.

Example 6

```
DATA;
X=TODAY(); Y=YEAR(X); M=MONTH(X);
D=DAY(X);
PUT Y= M= D=;
RUN;
```

The above program sets `Y` to be the current year, `M` the current month, and `D` the current day. The values of the three variables will be displayed in the log window.

Note that the above SAS program does not have `INPUT` statement.

Example 7

```
DATA;
X='STAT 2005 Programming Languages for Statistics';
W=LENGTH(X); Y=UPCASE(X);
Z=LOWCASE(X); T=INDEX(X,'ram');
R=INDEXC(X,'Daot'); G=INDEXW(X,'Programming');
K=SUBSTR(X,18); PUT _ALL_;
RUN;
```

Values displayed in the log window are:

```
X=STAT 2005 Programming Languages for Statistics
W=46
Y=STAT 2005 PROGRAMMING LANGUAGES FOR STATISTICS
Z=stat 2005 programming languages for statistics
T=15 R=13 G=11 K=ming Languages for Statistics
_ERROR_=0 _N_=1
```

Question: What is the value of `T` if

```
T=INDEXW(X,'ram');?
```

Answer: `T` will be equal to `0`.

Example 8

Set A to 1 if the value of a character variable B does not contain any numerical digits, and set A to 0 otherwise.

```
A = (INDEXC(B, '0123456789') EQ 0);
```

Set C to 1 if the value of a character variable B contains alphabetic character, and set C to 0 otherwise.

```
C = (UPCASE(B) ^= LOWCASE(B));
```

Example 9

You observe ten non-missing random sample `X1 - X10` from a normal population. Compute

$$P = \Pr\left(N(0,1) \le \frac{\bar{x}}{\sqrt{s^2/10}} \right),$$

where $\bar{x}$ and s^2 are the sample mean and the sample variance of `X1 - X10` respectively.

```
XMEAN = MEAN(OF X1 - X10);
SD = STD(OF X1 - X10);
P = PROBNORM(XMEAN*SQRT(10)/SD);
```

Conditional execution: IF statement

The syntax of the IF-THEN and ELSE statements:

```
IF condition THEN action1; [ELSE action2;]
```

action1 is performed when the condition is true.

The ELSE statement is optional. If it is used, action2 will be performed if the condition is false. Both action1 and action2 can be a SAS statement or a DO-group (to be discussed later).

Nested IF-THEN statement can be used. Comparison operators (such as >, >=) and logical operators (such as AND, OR) are commonly used to define the conditions.

Example 10

The following statements assign `GRADE = "A"` if `SCORE >= 90`, `GRADE = "B"` if `90 > SCORE >= 70`, and `GRADE = "C"` otherwise.

```
IF SCORE >= 90 THEN GRADE = 'A';
ELSE IF SCORE >= 70 THEN GRADE = 'B';
ELSE GRADE = 'C';
```

Example 11

Let STYLE be a character variable of length 1. We want to set B to 1 if STYLE is alphabetic and set B to 2 otherwise.

We can use the lowcase and upcase functions to assign value to B.

```
B = 1 + (upcase(STYLE) = lowcase(STYLE));
```

Alternatively, we can also use the following IF statement.

```
IF 'a' <= style <= 'z' | 'A' <= style <= 'Z'
THEN B = 1 ; ELSE B = 2;
```

Notice that all ASCII characters are ordered. Their orderings from the "smallest" to the "largest" are:

```
blank ! " # $ % & ' ( ) * + , - / 0 1 2 3 4 5 6
7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N
O P Q R S T U V W X Y Z [ \ ] ^ _ ` a b c d e f
g h i j k l m n o p q r s t u v w x y z { | } ~
```

Example 12

We have a text file in `D:\SAS\student.txt` storing raw data of three variables as follows:

Variable	Type	Column	Description
CLASS	Character	1-3	Class in School
NAME	Character	5-25	Student's name
AGE	Numeric	26-27	Student's age

Write a SAS program to create a temporary SAS data set `TEMP` storing the data. Set `AGE` to missing if it lies outside `10` and `20` (i.e. `AGE < 10` or `AGE > 20`). Set `CLASS` to `"?"` if it is not in the following list: `F1A`, `F1B`, `F2A`, `F2B`, `F3A`, `F4A`, and `F5A`.

```
DATA TEMP;
INFILE 'D:\SAS\STUDENT.TXT';
INPUT CLASS $ 1-3 NAME $ 5-25 AGE 26-27;
IF (AGE < 10) | (AGE > 20) THEN AGE = .;
IF ^(CLASS IN ('F1A', 'F1B', 'F2A', 'F2B',
'F3A', 'F4A', 'F5A')) THEN CLASS = '?';
RUN;
```

Example 13

Suppose the following world records of men swimming (long course, excluding relays) are stored in a text file `swim.txt` in `D:\`.

The first line, which is not part of the data, is a ruler to help us to easily locate the column number

```
----+----1----+----2----+----3----+----4----+----5----+
50m freestyle              20.91 Cesar Cielo
100m freestyle             46.91 Cesar Cielo
200m freestyle           1:42.00 Paul Biedermann
400m freestyle           3:40.07 Paul Biedermann
800m freestyle           7:32.12 Zhang Lin
1500m freestyle         14:34.56 Grant Hackett
50m backstroke             24.04 Liam Tancock
100m backstroke            51.94 Aaron Peirsol
200m backstroke          1:51.92 Aaron Peirsol
50m breaststroke           26.67 Cameron van der Burgh
100m breaststroke          58.58 Brenton Rickard
200m breaststroke        2:07.31 Christian Sprenger
50m butterfly              22.43 Rafael Munoz
100m butterfly             49.82 Michael Phelps
200m butterfly           1:51.51 Michael Phelps
```

Write a SAS program to read `swim.txt` and create a temporary data set `world_record`.

The data set should contain, among others, three variables, namely `NAME`, `EVENT` and `TIME` of which the values are given below. The unit of the variable, `TIME`, is second.

name	event	time
Cesar Cielo	50m freestyle	20.91
Cesar Cielo	100m freestyle	46.91
Paul Biedermann	200m freestyle	102.00
Paul Biedermann	400m freestyle	220.07
Zhang Lin	800m freestyle	452.12
Grant Hackett	1500m freestyle	874.56
Liam Tancock	50m backstroke	24.04
Aaron Peirsol	100m backstroke	51.94
Aaron Peirsol	200m backstroke	111.92
Cameron van der Burgh	50m breaststroke	26.67
Brenton Rickard	100m breaststroke	58.58
Christian Sprenger	200m breaststroke	127.31
Rafael Munoz	50m butterfly	22.43
Michael Phelps	100m butterfly	49.82
Michael Phelps	200m butterfly	111.51

This task has the problem that we cannot read the time directly. The value 1:42.00 cannot be correctly interpreted by SAS as 1 minute 42 seconds.

Therefore, we need to write statements to calculate the times.

```
DATA TEMP;
INFILE 'D:\SAS\SWIM.TXT';
INPUT EVENT $ 1-22 A $ 24-31 NAME $ 33-55 ;
T=INDEX(A,':');
IF T THEN TIME=60*SUBSTR(A,1,T-1)+SUBSTR(A,T+1);
ELSE TIME=A;
RUN;

DATA WORLD_RECORD(KEEP = NAME EVENT TIME);
SET TEMP;
RUN;
```

Bracketing related statements: DO-Groups

We can bracket a set of statements using a DO statement. This statement can be nested. The syntax of the simple DO statement is

```
DO; <put SAS statements here> END;
```

Example 14

```
DATA TEMP;
SET RECORDS;
IF SEX = 1
THEN DO; SEXNAME = 'M'; COLOUR='Blue'; END;
ELSE DO; SEXNAME = 'F'; COLOUR='Pink'; END;
RUN;
```

This program retrieves a temporary data set, `RECORDS`. Modify it and store it in another temporary data set, `TEMP`. The `IF` statements assign `SEXNAME` to be `'M'` and `COLOUR` to be `'Blue'` if `SEX` is `1` and `SEXNAME = 'F'` and `COLOUR = 'Pink'` otherwise.

Question: What will be the values of SEXNAME and COLOUR if SEX is missing?

Answer: If SEX is missing, SEXNAME = 'F' and COLOUR = 'Pink'.

Question: Modify the above program so that it assigns SEXNAME = 'F' and COLOUR = 'Pink' only when SEX = 2, and set SEXNAME = 'U' and COLOUR to be missing if SEX is neither 1 nor 2.

```
DATA TEMP;
SET RECORDS;
IF SEX = 1
THEN DO; SEXNAME = 'M'; COLOUR='Blue'; END;
ELSE IF SEX = 2
THEN DO; SEXNAME = 'F'; COLOUR='Pink'; END;
ELSE DO; SEXNAME = 'U'; COLOUR=''; END;
RUN;
```

Question: What happens if the IF statement is as follows?

```
IF SEX = 1
THEN DO; SEXNAME = 'Male'; COLOUR='Blue'; END;
ELSE DO; SEXNAME = 'Female'; COLOUR='Pink'; END;
```

Answer: If SEXNAME is a new variable (i.e. not a variable in the data set, RECORDS), the statement SEXNAME = 'Male' is the place where the variable SEXNAME is first known to SAS, and thus SAS will assign a length to the variable SEXNAME.

As the assignment statement sets SEXNAME to 'Male', SAS assigns a length of 4 to SEXNAME. Therefore, the value of SEXNAME for records with SEX = 2 will take value 'Fema'.

A way to solve the problem is to rewrite the IF statement as follows:

```
IF SEX ^= 1
THEN DO; SEXNAME = 'Female';
COLOUR='Pink'; END;
ELSE DO; SEXNAME = 'Male';
COLOUR='Blue'; END;
```

Another way is to add a LENGTH statement

```
LENGTH SEXNAME $ 6;
```

before the IF statement.

Conditional execution: SELECT statements

Other than the IF-THEN and ELSE statements, an alternative form of conditional execution statement is the SELECT groups. It gives a clearer structure than a nested set of IF-THEN-ELSE statements. Its syntax is

```
SELECT;
{WHEN (condition [{, condition} ... ] )
    action;} ...
[OTHERWISE action;]
END;
```

- SELECT goes through the list of WHEN conditions in order until it finds one that is true (a nonzero number). It then executes the corresponding WHEN action. SELECT executes only one WHEN action; if more than one WHEN condition is true, SELECT executes only the first one.
- We can specify more than one conditions separated by comma in each WHEN statement. The conditions are combined together with an OR operator.
- If none of the WHEN conditions is true, the OTHERWISE action is executed. The action can be a null statement (e.g. ;), which asks SAS to do nothing. It can also be a DO-group.
- If none of the WHEN conditions is true and there is no OTHERWISE statement, the SELECT block generates a runtime error.

Example 15

```
SELECT;
WHEN (A > 0 & B > 0) X = 2; WHEN (A > 0 , B > 0) X
= 1;
OTHERWISE ;
END;
```

A value of 2 is assigned to X when both A and B are positive.

X will be 1 if either A or B (but not both) is positive.

No values will be assigned to X if both A and B are less than or equal to 0.

Example 16

GRADE in Example 10 can be rewritten as follows.

```
SELECT;
WHEN (SCORE >= 90) GRADE ='A'; WHEN (SCORE >= 70) GRADE
='B'; OTHERWISE GRADE = 'C';
END;
```

or equivalently,

```
SELECT;
WHEN (SCORE >= 90) GRADE ='A'; WHEN (90 > SCORE >= 70)
GRADE ='B'; OTHERWISE GRADE = 'C';
END;
```

but not

```
SELECT;
WHEN (SCORE >= 70) GRADE ='B'; WHEN (SCORE >= 90)
GRADE ='A'; OTHERWISE GRADE = 'C';
END;
```

since GRADE = 'A' would never be executed.

The following comparison SELECT group is a variation on the SELECT group that simplifies the use of comparisons.

```
SELECT(select_expression);
{WHEN(expression [{,expression} ...]) action;} ...
[OTHERWISE action;]
END;
```

select_expression can be any valid SAS expression that gives a single value.

The value of the select_expression will be checked for equality with the value of expression in the WHEN statement. If more than one expressions are given, they will be combined using the OR operator.

SAS will execute action when the comparison in the WHEN statement gives a true value.

Example 17

```
SELECT (A);
WHEN (1) X = X*10;
WHEN (2, 3) ;
OTHERWISE X = 0;
END;
```

Note that SAS will do nothing when A = 2 or 3.

DO-Loops

There are two forms of DO-loops.

```
DO index = begin TO end [BY increment];
    <Put SAS statements here>
END;
```

and

```
DO index = value [{, value} ... ] ;
    <Put SAS statements here>
END;
```

The set of statements lying between the DO statement and the END statement will be executed repeatedly, one for each value of index specified in the DO statement.

- In the first type of DO-loop, index is a numeric variable. It is allowed, but usually not advisable, to change the value of index within the range of the loop.
- The user-supplied item begin is a numeric variable, constant, or expression that defines the beginning value taken by index.
- The value end is a numeric variable, constant, or expression that defines the last value taken by index.
- The increment is a numeric variable, constant, or expression that controls the increment of index.
- For example the statement

```
DO a = b TO c BY d;
```

asks SAS to execute the set of statements first for a = b, then for a = b + d, then for a = b + 2*d, and so on until the value of a exceeds c.

The statement,

```
DO a = 2 TO 12 BY 2;
```

means `a` will first be `2`, then `4`, `6`, `8`, `10`, `12`.

If `BY increment` is not specified, SAS assumes that the increment is `1`.

The values `begin`, `end`, and `increment` cannot be missing numeric value '`.`'.

The statement

```
DO x = a-8 to a+8 BY 2;
```

tells SAS that `x` is first set to `(a-8)`, then `(a-6)`, `(a-4)`, …, `(a+6)`, `(a+8)`.

For the second type of DO-loop, index can be numeric or character variable depending on the values after the = sign. The values after = can be a numeric or character variable, constant, or expression.

Example

```
DO x = 'A', 'B', 'C', 'D';
```

The two forms of expression of index can be mixed, e.g.,

```
DO x = 0, 1 TO 12 BY 3, 50 TO 55;
```

The values that x will take are 0, 1, 4, 7, 10, 50, 51, 52, 53, 54, 55.

Example 18

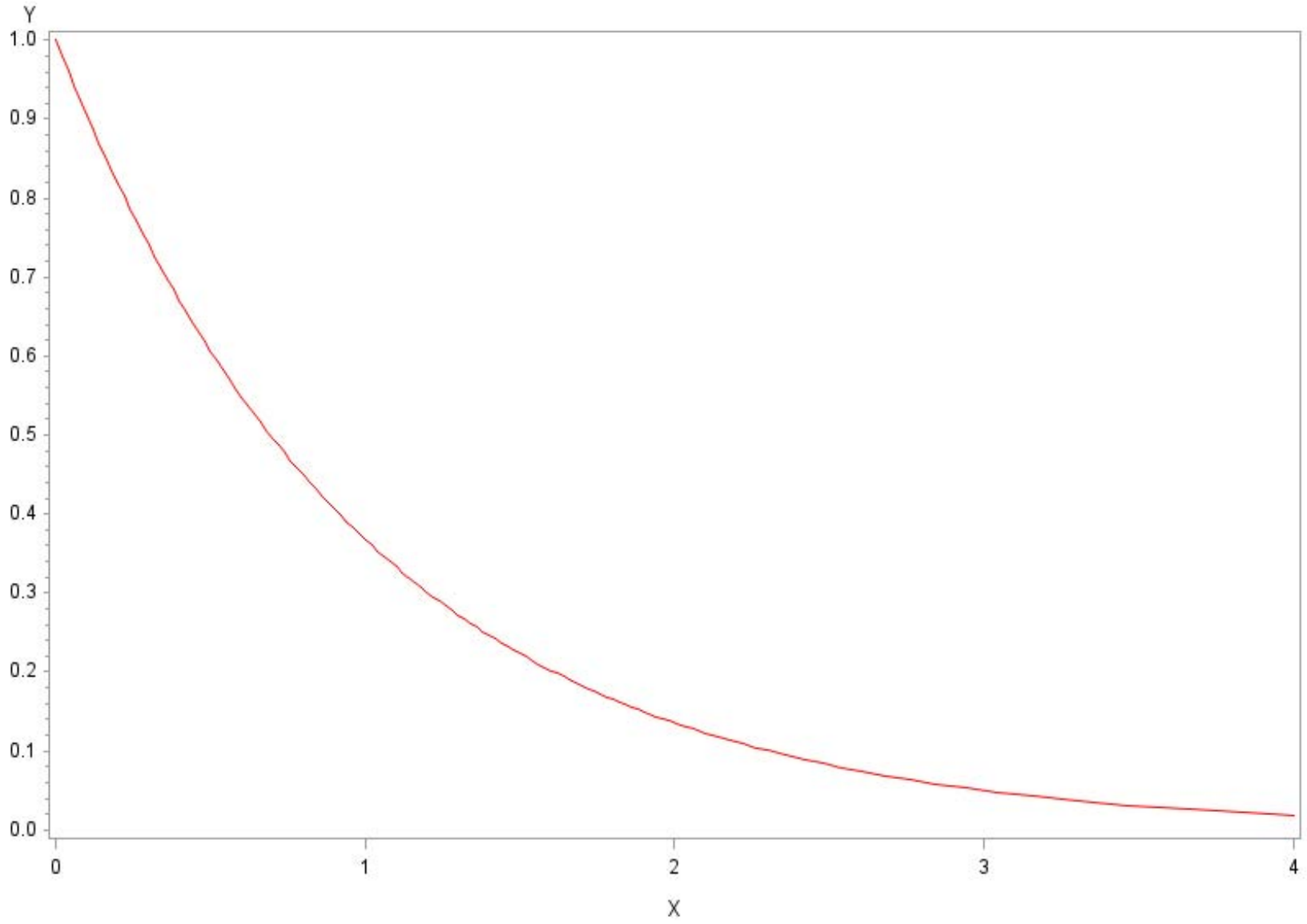
Suppose that we want to plot the density function of the exponential distribution with parameter ($\theta = 1$) in the range $[0, 4]$.

First of all, we need to create a data set storing the values of the density function, and then ask SAS to plot it. The following program creates such a data set.

```
DATA EXPONENTIAL;
DO X = 0 TO 4 BY 0.02;
     Y = EXP(-X);
     OUTPUT;
END;
RUN;
```

- First, X is set to 0. The corresponding Y is computed and the current values of X and Y are saved in the data set EXPONENTIAL by the OUTPUT statement.
- Then SAS returns to the top of the DO-loop and increases X by 0.02 (thus the new X = 0.02). A new value of Y is computed, X and Y are then stored in EXPONENTIAL.
- The statement is repeated again and again until X reaches a value larger than 4. At the end, EXPONENTIAL stores the (X,Y) values for X = 0.00, 0.02, 0.04, …, 4.00.
- We can use the following procedure to draw a graph of the density function after creating the data set,

```
SYMBOL1 COLOR=RED VALUE=NONE INTERPOL=JOIN;
PROC GPLOT DATA=EXPONENTIAL;
     PLOT Y*X;
RUN;
```


Y
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
1
2
3
4
X

Question: What happens if the OUTPUT statement is removed?

Answer: If the OUTPUT statement is omitted, an implicit OUTPUT statement would be added at the end of the program. that is after the END statement.

In such case only one pair of (X,Y) is stored, and the stored value of X is 4.02 because SAS stops executing the DO-loop when X is increased to 4.02.

Question: If the DO statement is replaced by

```
DO X = 0 TO 4;
```

what will happen?

Answer: The increment will be 1. Variable X takes values 0.00, 1.00, 2.00, 3.00, 4.00.

Question: How should we modify the program if we want `X` to have values `0`, `1`, `1.02`, `1.04`, …, `3.98`, `4`?

Answer: Replace the `DO` statement by

```
DO X = 0, 1 TO 4 BY 0.02;
```

Question: Can the increment be negative?

Answer: Yes. If we want, we can rewrite the `DO` statement as

```
DO X = 4 TO 0 BY -0.02;
```

Example 19

It can be shown that

$$\frac{2}{\pi}=\sqrt{\frac{1}{2}}\times\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}\times\sqrt{\frac{1}{2}+\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}}\times\cdots$$

Define

$$B_1=\sqrt{\frac{1}{2}},\quad B_2=\sqrt{\frac{1}{2}}\times\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}},\cdots$$

We have B_n converges to $2/\pi$ as n approaches infinity. Write a SAS program to compute $C = B_{20}$ and $D = 2/B_{20}$ (which should be close to π).

Let $A_0 = 0, B_0 = 1$

$$A_j = \sqrt{\frac{1}{2} + \frac{1}{2} A_{j-1}}, \quad B_j = B_{j-1} \times A_j$$

for $j = 1,2, \ldots$

```
DATA T;
A=0; B=1;
DO I=1 TO 20;
    A=SQRT(0.5+A/2); B=B*A;
END;
D= 2/B; PUT B= D=;
RUN;
```

The results are

```
B=0.6366197724 D=3.1415926536
```

Example 20

It is known that normal distribution can be used to approximate binomial distribution using the following formula

$$\Pr\left(B\left(n,p\right)\le m\right)=\Pr\left(N\left(np,np(1-p)\right)\le m+0.5\right)$$

for $0\le m\le n$.

Write a SAS program to compute the maximum absolute error of the above approximation with respect to values of m for given n and p.

$$A=\max_{0\le m\le n}\left|\Pr\left(B\left(n,p\right)\le m\right)-\Pr\left(N\left(np,np(1-p)\right)\le m+0.5\right)\right|$$

```
DATA NORMAL_APPROX;
INPUT N P;
A=0;
DO K=0 TO N;
A=MAX(ABS(PROBBNML(P,N,K) - PROBNORM((K+0.5-
N*P)/SQRT(N*P*(1-P)))),A);
END;
PUT N= P= A=;
CARDS;
10 0.5
8 0.3
RUN;
```

The results are

```
N=10 P=0.5 A=0.0026861603
N=8 P=0.3 A=0.0210252524
```

DO-WHILE statement

A DO-WHILE loop specifies a condition to tell SAS to repeatedly execute the set of statements when the condition holds.

The syntax of the DO-WHILE statement is

```
DO WHILE (expression);
    <put SAS statements here>
END;
```

The expression is tested before the process of executing the set of SAS statements. When expression gives false value, SAS goes to execute the statement after the END statement.

Example 21

Consider the Fibonacci sequence defined as $F_1 = 1$, $F_2 = 1$, and $F_n = F_{\mathrm{n-1}} + F_{n-2}$ for $n > 2$. Write a SAS program to compute the largest Fibonacci number which is smaller than 1000.

```
DATA FIBONACCI;
F1=1; F2=1;
DO WHILE (F2 < 1000);
A=F1+F2; F1=F2; F2=A;
END;
PUT F1=;
RUN;
```

The result is `F1=987`.