

5.

DATA 단계의 SAS 문장들

5.1 자료 입력

5.2 입출력 및 파일 제어 문장

5.3 제어 문장

5.4 기타 문장들

5.5 자동변수 (automatic variable)

5.6 분실값의 처리

5.7 데이터 입력 응용사례

5.

DATA 단계의 SAS 문장들

DATA 단계에서 자주 사용하는 SAS 문장들을 다룬다.

DATA 단계는 DATA 문장으로 시작하여 RUN 문장으로 마감하는 일련의 SAS 문장들로 구성된다. DATA 단계에서는 원시자료(raw data)의 입력, 자료 구조의 변환 등을 수행할 수 있으며 DATA 단계의 결과 생성된 SAS 자료를 토대로 PROC 단계에서 본격적인 자료 해석을 하게 된다. 다른 통계 패키지들과 비교하여 SAS의 강점 중 하나가 바로 자유롭고 융통성 있는 데이터 구조 변환에 대한 지원이다.

5.1 자료 입력

DATA 단계(step)에서 자료 입력/생성 방법

- ① 프로그램 내에 직접 자료 삽입하여 입력
- ② 외부 파일에 수록된 자료를 불러들여 입력 *text형태*
- ③ 이미 만들어진 SAS 자료를 그대로 읽어 입력

5.1.1 프로그램 내 삽입된 원시 자료의 입력

(예) ① 프로그램 내에 직접 원시 자료 삽입 (자료가 크지 않은 경우)

```
DATA students;
```

```
INPUT name $ 1-20 id_num sex $ dept $ exam1 exam2 exam3;
```

```
DATALINES;
```

```
Sung Min Young      2005 f stat 30 30 40
Sung Min Ji         2009 f cs 30 30 40
Hong Gil Dong       1750 m math 29 28 27
Kong Jui            1810 f cs 20 20 30
Pat Jui             1808 f stat 5 10 15
Yun Heung Boo       1670 m math 0 10 5
Yun Nol Boo         1665 m cs 20 30 10
Sun Nyeo            0300 f stat 25 25 35
Shim Chung          1880 f math 29 28 27
```

```
RUN;
```

5.1.2 외부 파일에 수록된 자료의 입력

(예) ② 외부 파일에 수록된, 따로 저장된 데이터 파일 읽기
(SAS 데이터 입력 시 가장 보편적으로 사용되는 입력 방식)

선후
↓
DATA score; 확장자명 꼭.
 INFILE "c:\work\exam.txt";
 INPUT id_num midterm final;
RUN;

[참고] INFILE 문장을 이용하여 불러들일 데이터 파일이 어디에 있는지 경로 및 데이터 파일명을 지정
(c:\work 이라는 디렉토리에 있는 exam.txt 라는 데이터 파일 지정)

외부 파일은 일반 문서편집기(text editor)로 작성된 텍스트 파일인 아스키(ASCII) 파일이다. 만약 확장자가 .hwp 또는 .xls 등으로 텍스트 파일이 아니라면 텍스트 파일로 다시 저장하여 사용한다.

[주의] Tab 문자의 처리

Excel 같은 계산표로부터 데이터 가져오기를 하면 때로 데이터 열 사이에 탭 문자가 삽입되는 경우가 있는데, SAS에서는 탭 문자를 알 수 없는 제어문자로 간주하기 때문에 오류가 발생할 수 있다.

SAS에서는 탭 문자가 숫자 뒤에 있으면 숫자와 탭 문자를 합쳐서 하나의 문자열로 인식하기 때문에 만일 이 위치에서 숫자변수를 입력하라는 명령이 있으면 그 결과는 분실값(missing value)으로 입력된다. 이런 현상을 방지하려면 DATA 단계의 첫 문장으로 다음 문장을 삽입하면 탭 문자가 모두 빈 칸으로 바뀐다.

(cf) id_num \$으로
문자변수화하면
tab이 왔어도 정상적으로
처리됨. (대신 이상한 값이 붙음)

INFILE '경로...파일명' EXPANDTABS;

missing value : .

(예) DATA;

 INFILE "c:\work\exam_tab.txt" EXPANDTABS;

 INPUT id_num midterm final;

RUN;

또는 SAS 시스템에서 지원하는 Import/Export Wizard 기능을 이용하여 변환한다.

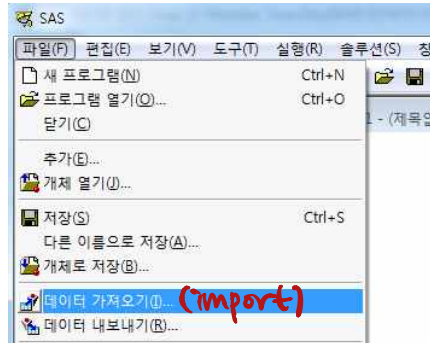
가져오기/내보내기 (Import/Export)

가져오기(Import)는 다른 데이터 포맷으로 저장된 파일을 읽어들이어 SAS 자료 포맷으로 저장하는 것이고, 내보내기(Export)는 SAS 자료를 다른 포맷의 데이터 파일로 저장하는 것이다.

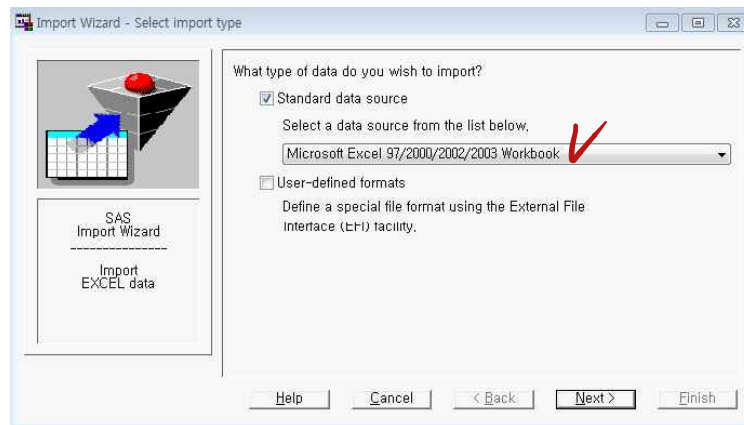
프린터지 이용.

(예) IMPORT (가져오기 마법사) : 외부 파일 포맷(엑셀) 데이터를 SAS 자료로
읽어 들이기

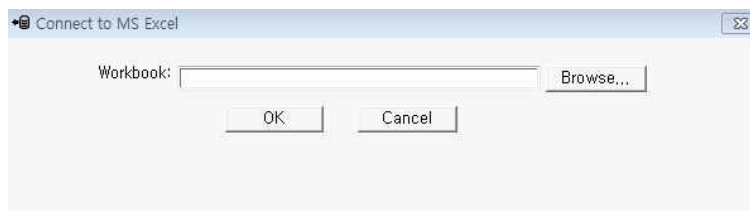
메뉴판에서 파일(File) → 데이터 가져오기(Import Data) 선택



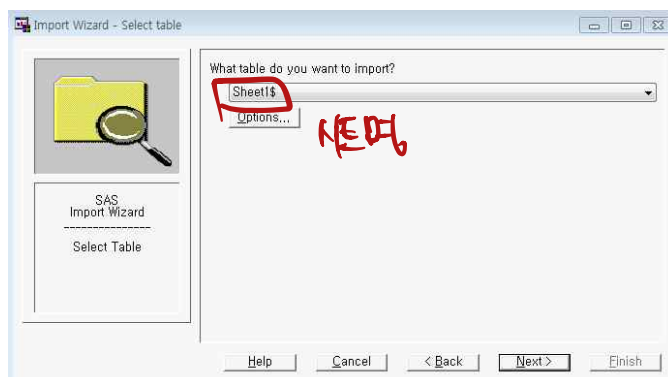
가져올 외부 데이터 파일의 포맷 선택 → Next



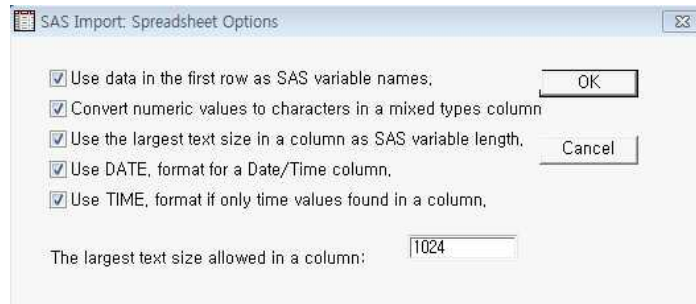
외부 데이터 파일이 저장된 경로와 이름 지정 → OK



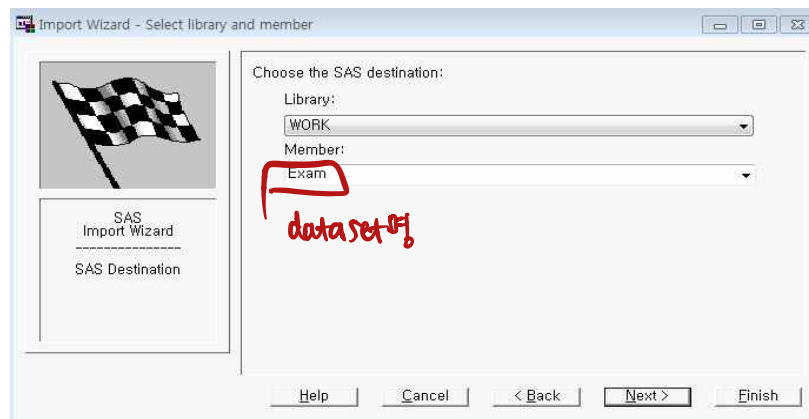
엑셀의 경우 sheet 번호 지정



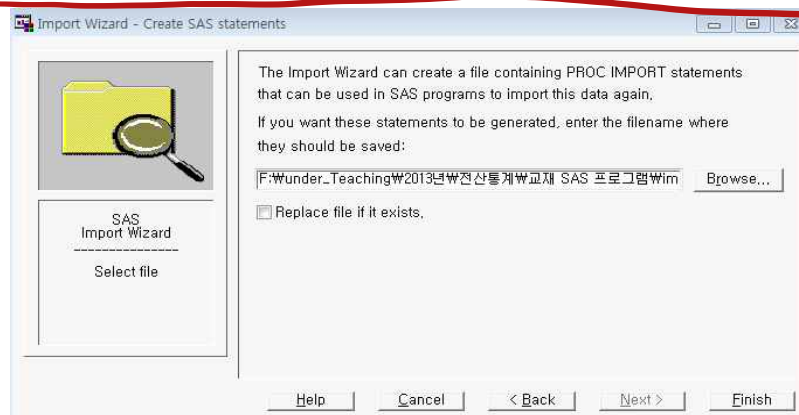
Options 클릭하고 데이터 파일 맨 첫 줄에 변수명이 있으면 'Use data in the first row as SAS variable names'에 체크된 것 확인 → OK → Next
(데이터 파일 내용에서 첫 번째 줄은 실제 데이터값이 아닌 변수명으로 사용한다는 의미)



외부 데이터 파일을 SAS 자료로 저장할 라이브러리와 파일명 지정 → Next



일련의 과정을 SAS 프로그램 코드로 생성하여 저장하려면 저장하고자 하는 디렉토리 및 SAS 프로그램 파일명(import xls.sas)을 지정 → Finish



생성된 SAS 프로그램 파일 (import xls.sas)

즉, 가져오기 마법사 사용 대신 직접 프로그램을 실행해도 됨
(PROC IMPORT)

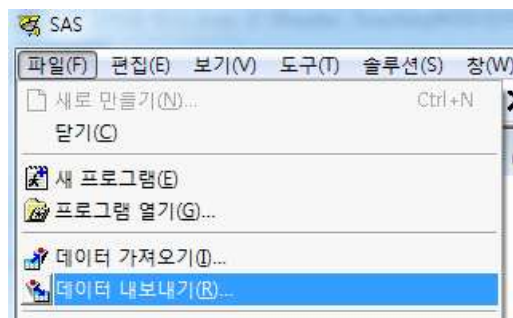
```

프로그램 편집기 - import.xls
PROC IMPORT OUT= WORK.Exam
  DATAFILE= "F:\Hunder_Teaching\2013년\전산통계\교재 SAS 프로그램\exam.xls"
  DBMS=EXCEL REPLACE;
  RANGE="Sheet1$";
  GETNAMES=YES;
  MIXED=YES;
  SCANTEXT=YES;
  USEDATE=YES;
  SCANTIME=YES;
RUN;
  
```

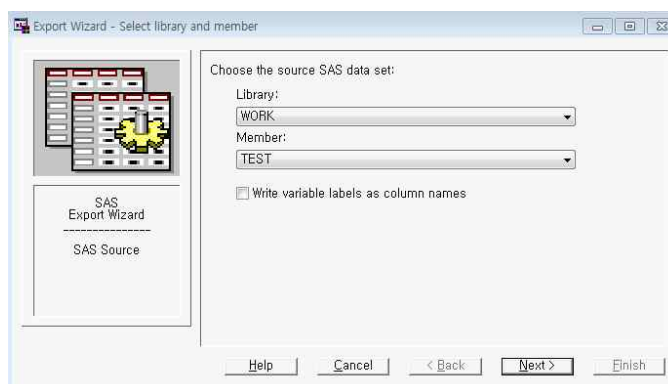
| | | | | SAS 시스템 | | | |
|---|--------|---------|-------|---------|--------|---------|-------|
| | A | B | C | OBS | id_num | midterm | final |
| 1 | id_num | midterm | final | | | | |
| 2 | 5721 | 64 | 74 | 1 | 5721 | 64 | 74 |
| 3 | 5723 | 32 | 28 | 2 | 5723 | 32 | 28 |
| 4 | 5724 | 88 | 92 | 3 | 5724 | 88 | 92 |
| 5 | 5725 | 76 | 66 | 4 | 5725 | 76 | 66 |
| 6 | 5701 | 92 | 98 | 5 | 5701 | 92 | 98 |
| 7 | 4702 | 85 | 75 | 6 | 4702 | 85 | 75 |

(예) **EXPORT (내보내기 마법사)** : SAS 포맷 자료를 외부 파일 포맷(엑셀)으로
변환하기

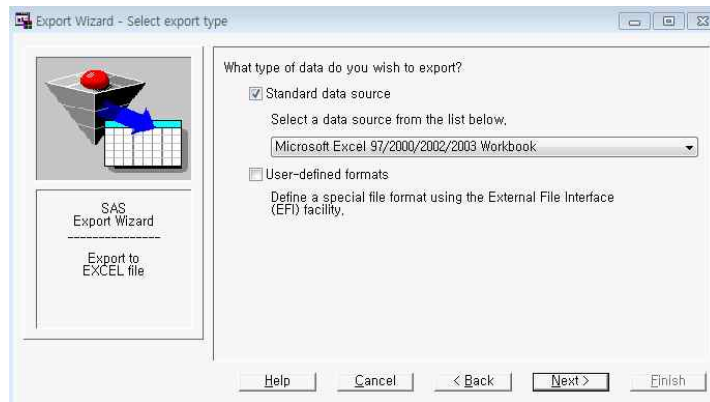
메뉴판에서 파일(File) → 데이터 내보내기(Export Data) 선택



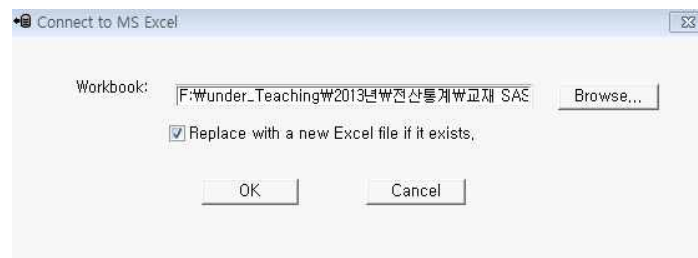
내보낼 SAS 자료 파일을 선택 → Next



내보낼 SAS 자료의 저장 데이터 포맷 지정 → Next



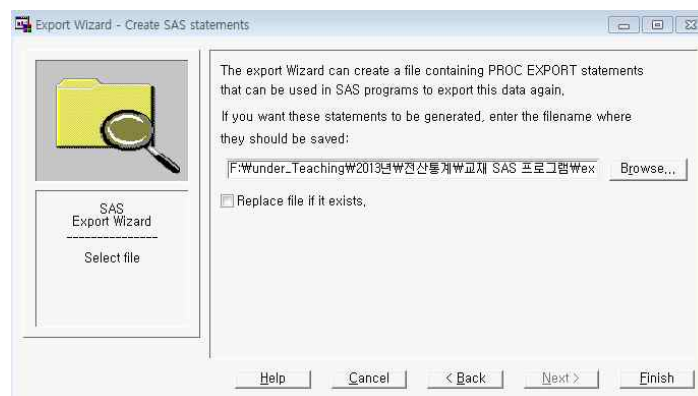
저장할 파일의 경로와 파일명 지정 → OK



엑셀 파일로 내보내는 경우에는 sheet 명 지정 → Next



일련의 과정을 SAS 프로그램 코드로 생성하여 저장하려면 저장하고자 하는 디렉토리 및 SAS 프로그램 파일명(export xls.sas)을 지정 → Finish



생성된 SAS 프로그램 파일 (export.xls.sas)

즉, 내보내기 마법사 사용 대신 직접 프로그램을 실행해도 됨
(PROC EXPORT)

```
프로그램 편집기 - export.xls
PROC EXPORT DATA= WORK.TEST
  OUTFILE= "F:\wonder_Teaching\2013년\전산통계\학교재 SAS 프로그램\test.xls"
  DBMS=EXCEL REPLACE;
  RANGE="test";
RUN;
```

| SAS 시스템 | | | | | | | |
|---------|------|-------|--------|---|------|-------|--------|
| OBS | name | total | credit | | A | B | C |
| 1 | Sung | 100 | A+ | 1 | name | total | credit |
| 2 | Moon | 90 | A+ | 2 | Sung | 100 | A+ |
| 3 | Kim | 30 | F | 3 | Moon | 90 | A+ |
| | | | | 4 | Kim | 30 | F |

5.1.3 이미 만들어진 SAS 자료를 읽어 입력

(예) ③ DATA 단계를 거쳐 이미 만들어진 SAS 자료로 저장된 데이터 파일 읽기

DATA credit;

SET students;

score=exam1+exam2+exam3;

IF score >= 50 THEN credit='Pass';

ELSE credit='Fail';

RUN;

[참고] 기존에 존재하는 students 라는 데이터 파일을 그대로 가져와서
새로운 데이터 파일 credit을 생성하는 과정으로, credit 에는 students
라는 데이터 파일에 존재하는 모든 값 이외에 추가로 새로운 두 개의
변수 score와 credit도 생성된다.

5.1.4 데이터 입력의 수행

DATA 단계는 관측(observation) 단위로 수행된다. 즉, DATA 단계가 길든 짧은 하나의 관측에 대하여 DATA 단계 내 모든 프로그램을 전부 실행한 후 그 다음 관측에 대해서도 마찬가지로 작업을 하고 더 이상 입력할 관측이 없을 때까지 DATA 단계를 각 관측에 대하여 반복 시행하는 것이다. 모든 데이터 입력이 끝나면 그 결과를 새로 생성되는 데이터셋으로 내보낸다.

참고로, 일반 컴퓨터 언어에서는 ARRAY 선언을 통하여 벡터 데이터를 정의하고 읽지만, 일반 컴퓨터 언어에서와는 달리 SAS 자료에서 변수(variable)는 그 자체가 이미 하나의 벡터(vector)이다.

5.2 입출력 및 파일 제어 문장

5.2.1 DATA 문장

DATA 문장은 DATA 단계의 시작을 알리는 문장으로, DATA 단계는 반드시 DATA 문장으로 시작한다.

DATA *SASdataset* (*options*) ;

library이름.dataset이름

SASdataset 은 DATA 단계의 결과로 만들어질 SAS 자료의 이름
변수 이름 짓는 규약에 준하여 사용자가 지정

[참고] 변수명 또는 데이터셋명 짓는 규칙

- 최대 32개 문자열 이용
- 첫 문자는 반드시 영문 또는 밑줄(underscore; _) 사용
- 두 번째 문자부터는 영문, 밑줄, 숫자 사용 가능
- 대소문자 구별 없음

[참고] 이름 지정하지 않으면 SAS 시스템 내부에서 DATA*n* 형식의
디폴트 이름 부여함. *n* 값은 1부터 순차적으로 증가함.

options 는 사용자가 지정 가능한 선택사항으로 자주 사용되는 것은,

DROP = *variable(s)*

자료에서 제거할 변수들을 나열

KEEP = *variable(s)*

자료에 포함할 변수들을 나열

[참고] DROP과 KEEP은 서로 상반되는 역할을 하므로 둘을 동시에
사용 불가

(예) DATA students;

```
INPUT name $ 1-20 id_num sex $ dept $ exam1 exam2 exam3;
```

```
DATALINES;
```

```
Sung Min Young      2005 f stat 30 30 40
```

```
Sung Min Ji         2009 f cs 30 30 40
```

```
Hong Gil Dong       1750 m math 29 28 27
```

```
RUN;
```

⇒ name, id_num, sex, dept, exam1, exam2, exam3 의 7개
변수가 포함된 students 라는 데이터 파일 생성

*-----;

```
DATA total (DROP=exam1 exam2 exam3);
```

```
SET students;
```

```
exam=exam1+exam2+exam3;
```

```
RUN;
```

⇒ 기존 데이터셋 students에서 exam1, exam2, exam3 이라는 변수를 제외한 나머지 변수가 포함되고 새로운 변수 exam이 추가된 total 이라는 데이터 파일 생성

*-----;

```
DATA sum (KEEP=name id_num sex dept exam);
```

```
SET students;
```

```
exam=exam1+exam2+exam3;
```

```
RUN;
```

⇒ 기존 데이터셋 students에 있는 name id_num sex dept 라는 변수를 포함하여 새로 생성된 exam 이라는 변수가 추가된 새로운 데이터 파일 sum 생성
(결과적으로 앞의 total 이라는 데이터셋과 동일)

*-----;

```
DATA students;
```

```
SET students;
```

```
exam=exam1+exam2+exam3;
```

```
RUN;
```

⇒ 기존 데이터셋 students 에 포함된 모든 변수들을 포함하고 추가로 exam 이라는 변수가 포함된 새로운 데이터셋 students 생성
기존의 데이터셋 이름과 동일한 이름으로 새로운 데이터셋을 만들었으므로 기존의 데이터셋은 파괴되고 새롭게 만든 데이터셋으로 대체된다. 즉, 자료의 이름을 전과 동일하게 설정하면 이전 파일에 겹쳐 쓰기가 수행됨.

*-----;

```
PROC PRINT DATA=total;
```

```
RUN;
```

⇒ PRINT procedure(단계)를 이용하면 데이터셋을 출력창에 출력할 수 있다. total 이라는 데이터셋이 출력창에 출력됨.

DATA 단계를 실행하면 작업 결과가 DATA 문장에서 사용자가 부여한 이름의 SAS 자료가 생성된다. SAS 자료는 다음과 같은 두 종류가 있다.

- 임시 SAS 자료 (temporary SAS data set)
 - : 하드디스크 상에 잠시 보관되었다가 SAS 시스템 종료와 함께 지워짐
 - Work 이라는 라이브러리에 임시 보관
- 영구 SAS 자료 (permanent SAS data set)
 - : SAS시스템이 종료되어도 지워지지 않고 디스크에 파일로 영구 보관됨
 - 원하는 경로에 라이브러리를 만들어 원하는 라이브러리에 영구 보관

LIBNAME *libraryname* '경로 지정'; 또는

libraryname 영문 또는 _ 로 시작, 최대 8자리
(숫자, 영문, _ 조합)



(예) LIBNAME mydata 'c:\WmyfilesWdata';

DATA mydata.score; ***[참고] '경로'는 이미 존재하는 경로이어야 함;**

SET students;

RUN;

⇒ 폴더 c:\WmyfilesWdata를 mydata 라는 라이브러리명으로
지정을 해둔 후, DATA 단계에서 score 라는 데이터셋을 만들어
mydata 라는 라이브러리에 영구 저장. (7장 참고)
이 프로그램을 실행시키고 탐색기로 c:\WmyfilesWdata를
검색하면 score 라는 SAS 데이터 파일이 생성되었음을 확인할
수 있음.

(예) DATA test; /* DATA work.test; 와 동일 */

SET students; /* SET work.students; 와 동일 */

RUN;

⇒ test 라는 데이터 파일은 work 이라는 디폴트 라이브러리에
임시로 보관되었다가 SAS 시스템 종료와 함께 사라짐
[참고] SAS 시스템의 디폴트 폴더 work의 물리적 위치는 SAS
시스템 설치 시 지정되었음 (교재 화면 1.8 참고)

한글
1

5.2.2 INPUT 문장

INFILE 문장과 함께 쓰여 외부 데이터 파일을 입력하거나, DATALINES 문장과
함께 쓰여 DATALINES 문장 이후에 나열된 원시 자료를 읽어 들일 때 사용된다.

INPUT *specification* ;

specification 은 자료 입력 형식

[참고] 세 가지 데이터 입력 방식 (혼합 사용도 가능)

- 자유입력(free input) 또는 목록입력(list input) 변수명만 필수
- 열입력(column input) 위치지정
- 포맷입력(formatted input)

[참고] 문자변수 입력 시 변수 이름 뒤에 \$ 붙임

\$ 부호는 INPUT 문장에서만 사용

(예)

| 1열 | 16 | 20 | 25 | 30 |
|------|----|-----|----|----|
| LEE | 33 | 180 | 80 | 2 |
| KIM | 40 | 168 | 65 | 1 |
| CHOI | 25 | 174 | 73 | 0 |
| . | | | | |
| . | | | | |

(1) 자유입력 또는 목록입력

변수 이름들만(목록만) 순서대로 나열

```
INPUT lastname $ age height weight children;
```

각 변수 값 사이에 적어도 하나의 빈 칸이 존재하는 경우에 사용 가능
(몇 번째 열부터 원하는 변수 값이 시작되는지 상관할 필요 없음)

단, 문자 데이터 값 읽을 때 길이가 8자를 초과하면 앞 8 글자만 입력됨
(이런 경우에는 열입력 또는 포맷입력 사용)

(2) 열입력

각 변수가 입력되는 열의 범위(위치)를 지정

```
INPUT lastname $ 1-15 age 16-17 height 20-22 weight 25-26  
children 30;
```

입력될 변수의 시작열과 끝열의 번호를 명기하고 그 사이에 dash를 삽입

lastname 의 경우 관측치마다 그 길이가 같지 않으므로 넉넉하게...
children 은 한 칸으로 충분하므로 시작 열과 끝 열 필요 없음.

(3) 포맷입력

포맷을 지정하여 입력하는 방식

INPUT lastname \$15. age 2. height 5. weight 4. children 4.;

필요자
위치도알려주세요.

크기

점말다

처음 15칸은 문자변수 lastname, 그 다음 2칸은 age, 그 다음 5칸은 height, 그 다음 4칸은 weight, 그 다음 4칸은 children에 할당되며 age, height, weight, children 각각은 모두 소숫점 아래 값이 없는 숫자변수들.

\$15. 이나 2. 등은 INFORMAT (교재 5.2.8 절 참고)

(4) 혼용

자유입력, 열입력, 포맷입력 혼용하는 방식

INPUT lastname \$ age 16-17 height 5. weight 4. children 30;

(5) 포인터

관측 하나가 두 줄 이상을 차지하는 경우 또는 관측 중 일부 데이터 값들만 골라 입력하는 경우에 유용

(포인터(pointer)는 읽을 지점 또는 입력이 시작되는 지점)

열 포인터 (column pointer): @n

줄 포인터 (line pointer): #n

열 포인터 @n 는 입력 지점이 n번째 열(column)로 이동

줄 포인터 #n 는 입력 지점이 n번째 줄(line)의 첫째 칸(열)으로 이동

(줄 포인터는 단일 관측이 최소 두 줄 이상에 걸쳐 있을 경우만 사용)

(예) LEE SOON SHIN

33 180 80

2

KIM JUNG HO

40 168 65

1

CHOI CHEE WON

25 174 73

0

위 자료에서 name, children, weight 만 입력하되 이 순서대로 입력하려면?

(첫줄에 name, 둘째줄에 age, height, weight, 셋째줄에 children)

➡ INPUT #1 name \$ 1-15 #3 children #2 @10 weight 2. #3;

#1 수행되면 첫째 줄 첫째 칸으로 포인터가 이동하여 이 지점에서 문자변수 name을 열입력하고 포인터는 첫째 줄 16번째 칸에 위치

→ #3 수행되면 셋째 줄 첫째 칸으로 포인터 이동하여 숫자변수 children을 자유입력하고

→ #2 수행되면 둘째 줄 첫째 칸으로 포인터 이동하고 @10 실행되면 열째 칸으로 포인터 이동하여 숫자변수 weight를 포맷입력

→ 마지막 #3에 의해 셋째 줄 첫째 칸으로 포인터 이동

[주의] 새로운 INPUT을 만나면 (즉, 다음 관측을 처리할 때) 포인터는 항상 현재 위치한 줄의 다음 줄 첫째 칸으로 이동한다. 즉, 맨 마지막 #3 은 포인터를 단일 관측이 차지하는 가장 마지막 줄로 보내기 위함이다. 만약 #3 을 맨 마지막에 덧붙이지 않으면 포인터가 두 번째 줄에 머물게 되므로 그 다음 관측을 읽을 때 포인터의 위치가 자동적으로 세 번째 줄로 가게 되어 자료 입력이 제대로 되지 않는다.

따라서 여러 줄에 걸친 관측을 읽을 때는 입력 포인터의 위치를 마지막 줄로 바꾸어 놓아야 한다.

[참고] 단, input 문장에서 동일한 관측이 입력되어 있는 맨 마지막 줄(이 예에서는 3번째 줄)에 있는 데이터값을 읽었다면 그 줄까지에 동일한 관측에 대한 데이터값이 입력된 것으로 인식. (즉, 별도로 포인터를 마지막 줄로 옮겨 놓지 않아도 무방. 즉, 이 예의 경우에는 input 문장 맨 마지막에 #3 생략 가능)

(6) 줄 포인터의 이동 정지

한 줄에 둘 이상의 관측이 존재할 경우 하나의 관측을 읽은 뒤 줄 포인터를 그 다음 줄로 이동하지 못하게 정지 또는 이동 보류

@ - 다른 INPUT 문장이 나올 때까지 기다린다.

@@ - 한 줄에 존재하는 모든 관측을 다 읽을 때까지 기다린다.

(예) 한 줄에 여러 개 관측이 있는 경우

LEE 33 180 80 2 KIM 40 168 65 1 PARK 25 174 73 0

.....

⇒ INPUT lastname \$ age height weight children @@;

읽으려는 모든 변수들 나열하고 맨 마지막에 @@ 덧붙이면, 하나의 관측을 다 읽은 후에도 줄 포인터가 다음 줄로 이동하지 않고 데이터 값이 있는 한 계속 다음 관측을 입력

(예) 한 줄을 읽을 때 상황에 따라 입력 형식이 바뀔 경우

^{1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25}
 P STAT-101 SUNG NAE KYUNG
 S STAT-101 051123 KIM AH MOO GAE 22
 S MATH-205 051242 HONG NOO GOO
 11 17 26.

P는 교수, S는 학생을 나타내고, STAT-101, MATH-205 등은 과목, 051123 등은 학생의 학번, 마지막 문자열은 이름.

각 관측을 읽어 P(교수)이면 과목과 이름을 입력하고

S(학생)이면 과목, 학번, 이름을 입력하려면?

⇒ INPUT status \$ 1 @;

IF status='P' THEN INPUT course \$ profname \$ 12-32;

ELSE IF status='S' THEN INPUT course \$ id_num name \$ 19-39;

입력값은 정수 (int는. 문자는 빈칸)

(7) 그룹포맷 입력

(예) 동일 포맷을 공유하는 변수들에 대해 각 변수마다 개별적으로 포맷을 지정하는 것은 비효율적 → 그룹포맷 활용

23414598385703845126

.....

x1, x2, x3, x4, x5 각각 4 자리의 정수

(데이터 값 사이에 빈칸 없음)

⇒ INPUT x1 4. x2 4. x3 4. x4 4. x5 4.;

⇒ INPUT (x1 x2 x3 x4 x5) (4. 4. 4. 4.);

⇒ INPUT (x1-x5) (5*4.);

⇒ INPUT (x1-x5) (4.);

(cf) 변수들 개수가 포맷 수보다 많으면 주어진 포맷이 처음부터 다시 재사용됨

[주의] 변수 목록과 포맷 목록에 항상 괄호 사용

(ex) 잘못 사용된 포맷 지정 방법

INPUT (x1-x5) 4.;

INPUT x1-x5 (4.);

INPUT x1-x5 4.;

괄호!

(예) INPUT (x1-x9) (2. 3.);

⇒ INPUT (x1-x9) (2. 3./2. 3./2. 3./2. 3./2.);

(예) INPUT (name x1-x5) (\$20. 5*4.);

⇒ name 은 처음 20 칸의 문자변수, 그 다음 21번째 칸부터 x1, x2, x3, x4, x5 각각 4자리씩의 숫자변수

(8) 기타 <<----- Skip

& 수정자

: 문자변수 입력 시 빈 칸 두 개가 연이어 나와야 문자의 끝임을 알리는 역할

(예) Alien vs Predator 2004

Terminator I 1984

Mission Impossible 2002

.....

영화제목과 개봉연도

(두 변수값 사이에 적어도 두 칸 이상 빈칸)

(영화제목은 하나의 빈 칸을 여럿 가질 수 있으며 최대 몇 글자가 될지 알 수 없음)

⇒ DATA;

LENGTH movie \$ 20; /* 변수의 성격 및 길이 지정 */

INPUT movie & \$ year; /* ⇔ INPUT movie \$20. year; */

DATALINES;

Alien vs Predator 2004

.....

5.2.3 DATALINES 문장

SAS 프로그램 내에 직접 원시자료를 삽입하여 입력할 때 자료가 시작됨을 알리는 것으로, DATALINES 문장 바로 다음 줄부터 실제 자료가 뒤따른다.

DATALINES;

DATALINES 문장 다음부터 입력되는 자료의 끝은 세미콜론(;)으로 표시하거나, RUN; 문장으로 마감

(단, 단계의 끝은 항상 RUN; 문장으로 마감하는 것이 좋음)

(예) DATA a;

INPUT x y;


```

    DATALINES;
        1 2
        3 4
RUN;
⇔ DATA a;
    INPUT x y;
    DATALINES;
        1 2
        3 4
    ;
RUN;

```

[참고] 자료값에 세미콜론(;) 부호 자체가 데이터로 포함된 경우 DATALINES 문장 사용 불가
 ⇒ DATALINES4 문장 사용하고
 데이터의 끝은 4개의 세미콜론(;;;;)으로 표시

(예) DATA a;
 INPUT number citation \$50.;
 DATALINES4;
 1 SMITH, 1982
 2 ALLEN ET AL., 1975; BRADY, 1983
 3 BROWN, 1990; LEWIS, 1984; WILLIAMS, 1982
 ;;;;

[참고] 하나의 DATA 단계에 둘 이상의 DATALINES 문장 사용 불가

5.2.4 INFILE 문장

외부 텍스트 파일로 저장되어 있는 데이터 파일을 입력할 때 사용한다.

INFILE *filespec options*;

filespec 은 읽을 데이터 파일 이름

방법1 - 보관된 외부 파일 경로와 이름을 **따옴표** 사이에 넣기

방법2 - **FILENAME** 문장 이용 (노트 7.8절 참고)

options 는 파일을 어떻게 읽을지에 관한 사용자 선택사항

- **FIRSTOBS**=*linenumber*

파일의 몇 번째 줄부터 읽을지 지정 (디폴트는 1 즉, 첫줄)

- **LINESIZE**=*linesize* 또는 **LS**=*linesize*

한 줄에서 몇 째 칸까지 데이터로 간주할지 지정

linesize 의 최대값은 32767

(e.g.) LS=72 ⇒ 72번째 칸 이후 값은 데이터로 취급 안함

- **OBS**=*linenumber*

파일에서 몇 째 줄까지 읽을지 지정

- **MISSOEVER** (5.2.5 참고)

- **STOPOVER** (5.2.5 참고)

(예) C:\WCLASS\StatisticalCom 에 다음과 같은 내용의 ex.txt 파일이 있고, 첫 두 줄과 마지막 줄은 데이터에 대한 설명으로, 자료만 읽으려면?

```
The following is a data set for Regression Analysis,  
Independent = Amout of Fertilizer,   Dependent=Yield  
1.23  24.56  
1.56  32.55  
1.78  33.70  
2.22  35.50  
Observed at Lab. C
```

⇒ DATA regdata;

INFILE 'c:\Wclass\Statisticalcom\Wex.txt' **FIRSTOBS**=3 **OBS**=6;

INPUT fert yield;

RUN;

5.2.5 INFILE DATALINES - INFILE의 특수 처리

INFILE DATALINES 문장이 사용되면 반드시 DATALINES 문장이 사용되고 실제

자료가 뒤따르는데, INFILE DATALINES 문장의 역할은 DATALINES 문장 이후의 데이터를 외부 파일에서 읽는 것처럼 착각하게 하는데 있다. 대부분 자료 입력에서 INFILE 문장이 사용되면 DATALINES 문장이 불필요하고, INFILE 문장이 사용되지 않으면 DATALINES 문장이 필요하다. 그러나 **INFILE 문장과 DATALINES 문장이 함께 사용되면 INPUT 문장에서 지원하지 않지만 INFILE 문장에서는 지원하는 특수한 지정을 자료 입력에 이용할 수 있어서 단순한 INPUT 문장으로는 처리 불가능한 구조의 자료 입력이 가능하다는 이점**이 있다.

```
DATA SASdataset;
    INFILE DATALINES options;
    INPUT ... ;
        . (SAS 문장들)
        .
    DATALINES;
        . (데이터)
        .
RUN;
```

options 는 파일을 어떻게 읽을지에 관한 사용자 선택사항

- FIRSTOBS=*linenumber*
- OBS=*linenumber*
- LINESIZE=*linesize*
- MISSOVER

INPUT 문장에서 지정한 변수에 대응하는 값이 존재하지 않는 경우에 입력 포인터가 **다음 줄로 내려가는 것을 방지** (데이터 값이 입력되지 않은 변수들은 **분실값으로 처리**)

- STOPOVER

INPUT 문장에서 지정한 변수에 대응하는 값이 존재하지 않는 경우 더 이상의 DATA 단계 수행을 중지하고 문제가 발생한 줄을 Log 창에 출력
(자료 입력 단계의 data coding이 제대로 되었는지 검색할 때 사용)

- LRECL=*record-length*

단일 관측 레코드의 길이(단위: 바이트)를 지정
(256 칸을 초과하는 경우에 지정 필요)

LRECL로 지정할 수 있는 값의 범위는 1~1,048,576 (1MB)

```
(예) DATA minji1;
      INPUT x1-x4;  /* 자유(목록)입력 방식 */
      DATALINES;
```

```
        1 4 7
        2 3 8 0
        ④7 3 9 1
        2 6 6 5
```

RUN;

⇒

| Obs | x1 | x2 | x3 | x4 |
|-----|----|----|----|----|
| 1 | 1 | 4 | 7 | 2 |
| 2 | ④ | 7 | 3 | 9 |
| 3 | 2 | 6 | 6 | 5 |

NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.MINJI1 has 3 observations and 4 variables.

*-----;

```
DATA minji2;
  INFILE DATALINES MISSOVER;
  INPUT x1-x4;  /* 끝부분 변수(들)에 대해 결측이 빈칸으로 된 경우 */
  DATALINES;  /* 자유입력 방식으로 읽는 경우만 MISSOVER 필요 */
```

```
        1 4 7
        2 3 8 0
        4 7 3 9 1
        2 6 6 5
```

RUN;

⇒

| Obs | x1 | x2 | x3 | x4 |
|-----|----|----|----|----|
| 1 | 1 | 4 | 7 | . |
| 2 | 2 | 3 | 8 | 0 |
| 3 | 4 | 7 | 3 | 9 |
| 4 | 2 | 6 | 6 | 5 |

[참고] 열입력/포맷입력 방식 이용 또는 자료 입력시 결측자리에 점(.) 표시
==> SAS 프로그램 참고

(예) STOPOVER 옵션 ==> SAS 프로그램 참고 (지정한 변수값 존재하지 않는
경우 더 이상의 DATA 단계 수행 중단하고 Log 창에 문제 발생 출력)

5.2.6 FILE 문장

FILE 문장은 PUT 문장과 함께 사용되어 DATA 단계에서 실행 결과를 어떤 형태의
출력 파일로 내보낼지 지정한다.

(cf) FILE 문장 - PUT 문장
INFILE 문장 - INPUT 문장

[참고] DATA 단계의 실행 결과는 Log 창에, PROC 단계의 실행 결과는 Output 창에 출력된다. 일반적으로 DATA 단계에서는 SAS 자료가 만들어질 뿐 출력되는 내용이 없으나 PUT 문장을 사용하면 DATA 단계에서도 실행 결과를 Log 창 등에 출력할 수 있다.

```
FILE filespec;
```

*filespec*에는 다음 세 가지 지정 가능
PRINT, LOG, '*filename*'

- FILE PRINT;

PUT 문장의 모든 실행 결과가 Output 창(출력창)에 출력.
PUT 문장이 존재하는 DATA 단계마다 삽입해 놓으면 PROC 단계의 결과뿐 아니라 PUT 문장의 결과(즉, DATA 단계의 결과)도 함께 Output 창에 출력되므로 작업 종료후 Output 창 내용만 파일로 저장하면 간편함.

- FILE LOG;

Log 창에 출력되는 것이 디폴트이므로 이 문장을 따로 지정할 필요는 없으나, FILE PRINT; 문장을 이용하여 작업하다가 어느 지점 이후에는 PUT 문장의 결과를 Output 창에 보내지 않고 다시 원래대로 Log 창에 출력할 때 사용

- FILE '*filename*' PRINT;

PUT 문장의 결과가 지정한 이름의 파일로(또는 경로 포함 파일명으로) 저장되며 Output 창에는 아무것도 출력되지 않음.
저장된 파일에는 줄을 띄우거나 페이지를 넘기는 등의 인쇄기 조절 코드가 자동적으로 삽입되는데 이런 파일을 인쇄 파일(print file)이라 함.

- FILE '*filename*';

FILE '*filename*' PRINT; 문장과 동일하나 저장되는 파일은 비인쇄 파일(non-print file)임. 비인쇄 파일은 인쇄기 조절 코드를 포함하지 않는 순수한 아스키 파일.

(예) DATA test;

```
FILE 'simple.lst' ;
```

```
DO i=1 TO 10;
```

```
    ii=i*i;
```

```
    PUT i ii;
```

END;

RUN;

⇒ 1 부터 10 까지 열 개의 숫자와 각 숫자의 제곱들로 이루어진 모두 열 줄의 결과가 simple.lst 란 이름의 인쇄 파일로 저장된다.

Output 창으로의 출력은 없다.

Log 창을 보면 저장 경로 확인 가능.(또는 직접 경로포함 파일 지정 가능)

NOTE: The file 'simple.lst' is:
File Name=C:\Documents and Settings\Administrator\simple.lst,
RECFM=P, LRECL=99

NOTE: 10 records were written to the file 'simple.lst'.

5.2.7 PUT 문장

DATA 단계에서 프로그램의 실행 결과를 출력하는 문장으로, FILE 문장을 이용하여 별도의 출력 경로를 지정하지 않는한 **Log 창**에 출력된다. PUT 문장의 유효범위는 PUT 문장이 존재하는 DATA 단계에 한정된다.

PUT *specification*;

specification 은 사용자가 정의하는 선택사항으로, 변수, 따옴표로 둘러친 문자열, 출력 방식 등을 지정

[참고] 세 가지 출력 방식 (혼합 사용도 가능)

- 목록출력(list output)
- 열출력(column output)
- 포맷출력(formatted output) (교재 5.2.8절 참고)

(예) DATA class;

INPUT name \$ 1-10 sex \$ 12 age 15-16;

PUT name sex age; *<---- 목록출력 방식;

DATALINES;

Kim M 20
Hwangbo F 23
Park M 19

RUN;

⇒ Log 창에 다음과 같은 결과 출력

목록출력 방식을 사용하면 인접한 두 값 사이의 빈 칸 하나로 각 데이터 값을 구분

```
Kim M 20
Hwangbo F 23
Park M 19
```

(예)

PUT name 'is ' age 'years old.'; *문자열 삽입;

..... /* log 창에 출력되는 내용은, name ▯ "내용 age ▯ "내용 */

⇒

```
Kim is 20 years old.
Hwangbo is 23 years old.
Park is 19 years old.
```

(예)

PUT name 21-30 sex 32 age 40-41; *<---- 열출력 방식;

.....

[참고] 디폴트 출력창이 Log 창인데, Log 창에는 프로그램의 실행 상황 등 잡다한 결과가 출력된다. 따라서 PUT 문장 결과를 Output 창에 독립된 페이지로 출력하고 각 데이터 값이 어떤 변수에 해당되는지 변수 이름을 각 데이터 값들 위에 쓰는 것이 편리한 경우가 있다. 이런 **출력 조절을 위한** 유용한 것들로 다음과 같은 **포인터 제어 지정자**들이 있다.

```
_PAGE_ : 줄 포인터를 다음 페이지 첫째 줄로 이동
#n : 줄 포인터를 n 번째 줄로 이동
/ : 줄 포인터를 다음 줄 첫째 칸으로 이동
@n : 열 포인터를 n 번째 열로 이동
+n : 열 포인터를 현 위치에서 n 칸 만큼 오른쪽으로 이동
```

(예) DATA _NULL_;

/* _NULL_ 은 SAS에서 사용하는 특별한 이름으로, DATA 단계
실행 후 생성된 자료를 보관할 필요가 없음을 의미 */

INPUT name \$ 1-10 sex \$ 12 age 15-16;

FILE PRINT;

/* PUT 문장의 실행결과를 output 창에 보내기 위하여 지정 */

IF _N_=1 THEN PUT _PAGE_

// @21 20*“=”

/ @26 “Class Data”

/ @21 20*“=”

```

/// @21 "Name      Sex  Age"
/ @21 "_____ _" // ;
/* _N_ 은 SAS에서 사용하는 특별한 변수이름으로, 관측마다의
   고유한 관측번호를 나타내는 변수.
   첫 번째 관측을 읽으면 페이지를 넘기고 두 줄을 띄운 후
   21번째 칸부터 문자 '='을 20개 출력하고, 그 다음 줄의 26번째
   칸부터 'Class Data'를 쓰는 등 표제와 변수 이름을 출력.*/
/* 첫 번째 관측을 읽고 다음의 INPUT 문장, PUT 문장을 실행하고
   두 번째 관측부터는 자료가 다할때까지 _N_=1 이란 조건을
   만족하지 못하므로 곧바로 INPUT 문장, PUT 문장을 실행 */
PUT @21 name @32 sex @39 age;
DATALINES;
KIM           M  20
HWANGBO       F  23
PARK          M  19
RUN;
⇒

```

```

=====
      Class Data
=====

Name      Sex  Age
-----
KIM       M   20
HWANGBO   F   23
PARK      M   19

```

(예) 포맷 출력 방식은 5.2.8 참고

5.2.8 INFORMAT과 FORMAT

INFORMAT : (INPUT 문장에서만 유효) 자료를 읽어들이는 입력 방식

FORMAT : (PUT 문장에서만 유효) 처리된 결과를 내보내는 출력 방식

(cf) DATA 단계에서,

INPUT 문장 : (INFILE 문장과 함께) 자료 입력시 사용

PUT 문장 : (FILE 문장과 함께) 결과 출력시 사용

| 목적 | 문장 | 기능 |
|-------|--------|---------------------|
| 자료 입력 | INPUT | 입력 방식(INFORMAT)을 지정 |
| | INFILE | 입력 파일을 지정 |
| 결과 출력 | PUT | 출력 방식(FORMAT)을 지정 |
| | FILE | 출력 파일을 지정 |

입력 방식: informat $w.d$

출력 방식: format $w.d$

w 는 입출력되는 값이 차지하는 칸 수 (부호와 소수점 포함 전체 칸 수)

d 는 입출력되는 값이 차지하는 소수점 이하 자리수 ($w > d$)

문자열의 입출력에서는 지정하지 않음

[참고] 숫자 입출력시, 전체 자리수와 소수점 이하 자리수 지정

(e.g.) -123.456 의 경우,

전체 자리수는 8자리, 소수점 이하 자리수는 3자리 필요

(부호와 소수점 자리도 전체 자리수에 포함)

문자 입출력시, 문자열 길이 지정

(문자열이 빈 칸으로 시작하는지에 따라 방식 달라짐)

(1) 입력 방식 (INFORMAT 방식)

- $w.d$ 또는 $Fw.d$: 숫자 읽는 표준형 exam 5. name \$20.

w . 방식은 소수점 이하 자리수 없는 정수 입력에 사용

(부호 포함 최대 32자까지 읽을 수 있음)

(부호와 숫자 사이에 빈 칸 있을 수 없음)

$w.d$ 방식은 w 로 지정된 칸들 사이에 아무데나 숫자 넣을 수 있고,

소수점이 없으면 10^d 으로 나눈 값이 입력되고,

소수점이 이미 붙은 숫자는 보이는 그대로 입력됨.

숫자 좌우 빈 칸은 있더라도 무시됨.

분실 값인 경우에는 아무 데나 마침표(.)를 넣음.

(예) DATA a;

```
INPUT @1 x 7. ; /* 1열에서 7열까지 7자리 정수 입력 */
```

```
DATALINES;
```

```
-23□□□□ /* -23 으로 입력됨 ( □ 는 한 칸의 빈칸 의미) */
```

```
□□□□-23 /* -23 으로 입력됨 */
```

```
□□-23□□ /* -23 으로 입력됨 */
```

```
□-23.0□ /* -23 으로 입력됨 */
```

```
-2.3E01 /* -23 으로 입력됨 */
```

```
-2.3□□□ /* -2.3 으로 입력 */
```

```
RUN;
```

(예) DATA b;

```
INPUT y 6.2; /* 총 6자리 중 소숫점자리 2 */
```

```
DATALINES;
```

```

1234□□    /* 12.34 로 입력됨 */
□□1234    /* 12.34 로 입력됨 */
□12.34    /* 12.34 로 입력됨 */
□12.3□    /* 12.3  로 입력됨 */
□□□.□□    /* 결측 */
RUN;

```

- **\$w.** 또는 **\$Fw.** : 문자 읽는 표준형 (최대 32,767 자까지 입력 가능)
 왼쪽 빈 칸들은 무시,
 입력된 문자열은 왼쪽 기준으로 정렬,
 분실 데이터 값(결측값)은 마침표로 표시

(예) DATA c;

```

INPUT @1 country $7.;
/* 또는 INPUT country $ 1-7; */
DATALINES;
KOREA□□    /* KOREA□□ 로 입력됨 */
□KOREA□    /* KOREA□□ 로 입력됨 */
□□KOREA    /* KOREA□□ 로 입력됨 */
□□□.□□□    /* 결측 */
RUN;

```

- **\$CHARw.** : 문자 읽는 방식
 \$w. 방식과 동일하나 빈 칸이 무시되지 않음

(예) DATA c;

```

INPUT @1 country $CHAR7.;
DATALINES;
KOREA□□    /* KOREA□□ 로 입력됨 */
□KOREA□    /* □KOREA□ 로 입력됨 */
□□KOREA    /* □□KOREA 로 입력. 3개 값들은 각각 다른 값 */
RUN;

```

(2) 출력 방식 (FORMAT 방식)

- **w.d** 또는 **Fw.d** : 숫자 출력 표준형
 출력될 숫자의 소수점 이하 자리수가 d 미만이면 나머지 자리수는 0으로 채워짐.
 출력될 숫자의 길이가 w (또는 d)보다 크면 반올림된 결과 출력.
 숫자가 너무 커서 지정 방식으로 출력 불가능할 경우 **BESTw.** 출력방식으로 자동 전환.
BESTw. 방식으로도 처리 불가능하면 * 표시.
 (가능한 큰 값의 w를 지정하는 것이 유리)

(예) DATA d;

x=123.45; y=123.45;

PUT x 8.4; /* 123.4500 이 출력됨 */

PUT y 8.1; /* 123.5 출력(소숫점 둘 째 자리에서 반올림처리) */

PUT y 3.; /* (PUT y 3.0; 과 마찬가지로) 123 출력 */

PUT y 3.1; /* 123 출력 (BEST 출력방식으로 전환) */

RUN;

- **BESTw.** : 숫자 출력 디폴트 포맷

출력 자리수 w 만 지정 (나머지는 SAS 가 알아서 표현)

(예) DATA d;

x=1238567;

PUT x BEST6.; /* 1.24E6 이 출력됨(반올림처리) */

RUN;

⇒ 제대로 x 값 출력하려면 7칸 필요하지만, 6칸만 사용 가능하므로
자동적으로 지수 부호 E 사용

(예) DATA d;

x=1234567;

PUT x BEST3.; /* 1E6 이 출력됨 */

RUN;

(예) DATA d;

x=1234567;

PUT x BEST2.; /* ** 이 출력됨 (출력 포기 선언) */

RUN;

- **\$w.** 또는 **\$Fw.** : 문자 출력 표준형

왼쪽으로 정렬됨

(예) DATA e;

c="ㅁKOREA";

PUT c \$5.; /* ㅁKORE 이 출력됨 */

RUN;

5.2.9 SET 문장

이전 DATA 단계에서 이미 만들어진 SAS 자료를 읽어 들일 때 사용한다.

기존의 자료를 활용하여 새로운 SAS 자료를 출력하는 과정에서 자료 구조의 변형,
분석에 필요한 자료의 선별적 처리 등이 가능하다.

SET SASdatasets (options);

SASdatasets 는 SET 문장을 이용하여 읽어들이는 SAS 자료의 이름
options 는 사용자 선택사항으로,

DROP, KEEP, FIRSTOBS, OBS 등

(1) SAS 자료의 복사

(예) DATA temporary;

SET class;

RUN;

⇒ 기존의 자료 class를 복사하여 새로운 자료 temporary를 생성
(class 라는 자료와 temporary 라는 자료는 완전히 동일한 자료)

(2) 변수 선택

(예) DATA one;

SET class (KEEP=name id_num sex);

RUN; /* 기존 class 자료에서 name, id_num, sex 라는 변수만을
선택하여 새 자료 one을 생성 */

DATA two;

SET class (KEEP=name score);

RUN; /* 기존 class 자료에서 name, score 라는 변수만을
선택하여 새 자료 two를 생성 */

⇔ DATA one (KEEP=name id_num sex) two (KEEP=name score);

SET class;

RUN; /* 약간의 차이가 있을 수 있음 주의 */

[참고] 제거할 변수 이름을 나열하는 것이 더 간편한 경우에는 KEEP 대신
DROP 사용 가능

(예) DATA one;

SET class (DROP=exam1 exam2 exam3 score);

RUN;

[주의] DROP 은 변수를 제거

DELETE 는 관측을 제거 (5.2.15 on p.44 참고)

[cf] DATA 문장에서의 KEEP/DROP vs. SET 문장에서의 KEEP/DROP

(3) 관측 선택

(예) SET class (FIRSTOBS=2 OBS=5); <<=== 관측 선택

(예) DATA males;

```

SET class;
IF sex='M';
RUN;  /* 기존 class 자료에서 sex라는 변수값이 M 인 자료만 모아 새
      자료 males 를 생성 */
DATA females;
SET class;
IF sex='F';
RUN;  /* 기존 class 자료에서 sex라는 변수값이 F 인 자료만 모아 새
      자료 females 를 생성 */
⇔ DATA males females;
SET class;
IF sex='M' THEN OUTPUT males;
IF sex='F' THEN OUTPUT females;
RUN;

```

(4) SAS 자료의 결합/병합

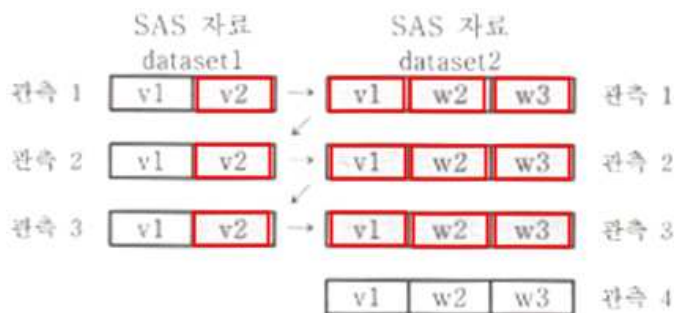
둘 이상의 입력원(input source)를 동시에 처리하여 단일 데이터 생성

SET 문장을 이용한 데이터 결합 방식1 (예): [둘 이상의 SET 문장 이용]

```

DATA concat1;
SET dataset1;
SET dataset2;  /* (cf) SET dataset2; SET dataset1; 결과는? */
RUN;

```



⇒ 자료 concat1 은 4개 변수(v1, v2, w2, v3)와 3개 관측들을 가짐

[참고] 둘 이상의 SET 문장을 이용한 데이터 결합 시,

- 결합된 최종 자료에 기억되는 관측수는 가장 관측수가 작은 자료의 관측수
- 자료들 간 겹치는 변수가 존재하는 경우, 먼저 입력된 변수값은 지워지고 나중에 입력되는 변수값이 기억됨

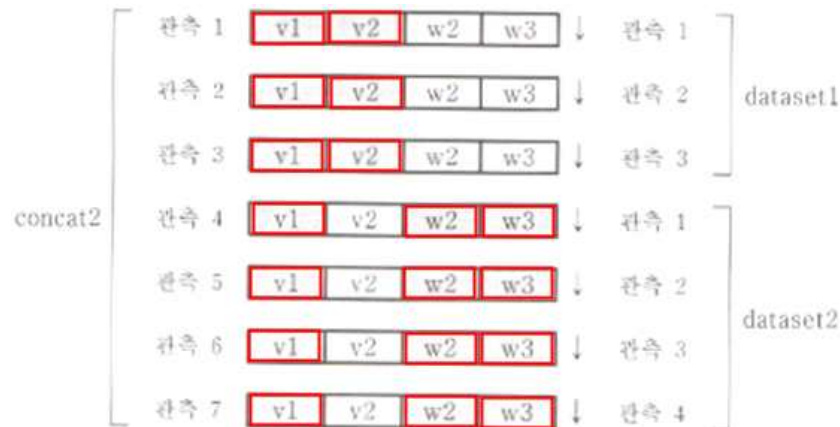
SET 문장을 이용한 데이터 결합 방식2 (예): [단일 SET 문장 이용]

DATA concat2;

SET dataset1 dataset2; /* (cf) SET dataset2 dataset1; 결과는? */

RUN; * (비교) MERGE dataset1 dataset2;

⇒ 자료 concat2는 4개 변수(v1, v2, w2, w3)와 7개 관측들 가짐
빨강색 박스로 되어 있지 않은 변수값들은 결측치로 처리됨.



[참고] 단일 SET 문장을 이용한 데이터 결합 시,

- 총관측수는 결합되는 각 자료의 관측수의 총합
- 자료들 간 서로 다른 변수가 존재하는 경우는 분실값으로 처리

[참고] PROC APPEND <<<<<<----- Skip

: 자료들의 관측수만 다를 뿐 구조 동일하다면 SET 문장보다 효율적

(예) PROC APPEND BASE=master NEW=slave; RUN;

/* master 자료에 slave 자료를 덧붙임 */

/* master와 slave는 동일 변수 갖고 있어야 함 */

(5) 둘 이상의 SET 문장을 사용한 결합 (예제)

(예) DATA s1;

INPUT name \$ 1-20 id_num sex \$ dept \$;

DATALINES;

Sung Min Young 2005 f stat
Sung Min Ju 2009 f cs
Hong Gil Dong 1750 m math

RUN;

DATA credit;

INPUT exam1 exam2 exam3;

SET s1;

```

score=exam1+exam2+exam3;
IF score >= 50 THEN credit='Pass';
    ELSE credit='Fail';
DROP id_num score;
DATALINES;
    30 30 40
    30 30 40
    29 28 27

```

RUN;

PROC PRINT DATA=credit;

RUN;

⇒

| OBS | exam1 | exam2 | exam3 | name | sex | dept | credit |
|-----|-------|-------|-------|----------------|-----|------|--------|
| 1 | 30 | 30 | 40 | Sung Min Young | f | stat | Pass |
| 2 | 30 | 30 | 40 | Sung Min Ju | f | cs | Pass |
| 3 | 29 | 28 | 27 | Hong Gil Dong | m | math | Pass |

⇔ DATA s1;

INPUT name \$ 1-20 id_num sex \$ dept \$;

DATALINES;

```

Sung Min Young      2005 f stat
Sung Min Ju         2009 f cs
Hong Gil Dong       1750 m math

```

RUN;

DATA s2;

INPUT exam1 exam2 exam3;

DATALINES;

```

    30 30 40
    30 30 40
    29 28 27

```

RUN;

DATA credit;

SET s2;

SET s1;

score=exam1+exam2+exam3;

IF score >= 50 THEN credit='Pass';

ELSE credit='Fail';

DROP id_num score;

RUN;

```

(예) DATA s1;
      INPUT name $ 1-20 id_num sex $ dept $;
      DATALINES;
Sung Min Young      2005 f stat
Sung Min Ju         2009 f cs
Hong Gil Dong       1750 m math
RUN;
DATA credit1;
      INPUT exam1 exam2 exam3;
      SET s1;
      score=exam1+exam2+exam3;
      IF score >= 50 THEN credit='Pass';
      ELSE credit='Fail';
      DROP id_num score;
      DATALINES;
      30 30 40
      30 30 40
RUN;
PROC PRINT DATA=credit1;
RUN;
⇒

```

| OBS | exam1 | exam2 | exam3 | name | sex | dept | credit |
|-----|-------|-------|-------|----------------|-----|------|--------|
| 1 | 30 | 30 | 40 | Sung Min Young | f | stat | Pass |
| 2 | 30 | 30 | 40 | Sung Min Ju | f | cs | Pass |

(6) 단일 SET 문장을 사용한 결합 (예제)

(예) 관측수는 달라도 동일한 변수들을 갖는 둘 이상의 자료 결합
(단일 SET 문장 사용) ((cf) PROC APPEND)

```

DATA class1;
      INPUT name $ 1-20 id_num sex $ dept $ exam1 exam2 exam3;
      DATALINES;
Sung Min Young      2005 f stat 30 30 40
Sung Min Ji         2009 f cs   30 30 40
Hong Gil Dong       1750 m math 29 28 27
RUN;
DATA class2;
      INPUT name $ 1-20 id_num sex $ dept $ exam1 exam2 exam3;
      DATALINES;

```



```
Kong Jui          1810 f cs    20 20 30
Pat Jui           1808 f stat  5 10 15
Yun Heung Boo     1670 m math  0 10 5
Yun Nol Boo       1665 m cs    20 30 10
Sun Nyeo          0300 f stat  25 25 35
Shim Chung        1880 f math  29 28 27
```

RUN;

DATA new;

```
SET class1 class2; * (비교) SET class2 class1;
```

RUN;

⇒

| OBS | name | id_num | sex | dept | exam1 | exam2 | exam3 |
|-----|----------------|--------|-----|------|-------|-------|-------|
| 1 | Sung Min Young | 2005 | f | stat | 30 | 30 | 40 |
| 2 | Sung Min Ji | 2009 | f | cs | 30 | 30 | 40 |
| 3 | Hong Gil Dong | 1750 | m | math | 29 | 28 | 27 |
| 4 | Kong Jui | 1810 | f | cs | 20 | 20 | 30 |
| 5 | Pat Jui | 1808 | f | stat | 5 | 10 | 15 |
| 6 | Yun Heung Boo | 1670 | m | math | 0 | 10 | 5 |
| 7 | Yun Nol Boo | 1665 | m | cs | 20 | 30 | 10 |
| 8 | Sun Nyeo | 300 | f | stat | 25 | 25 | 35 |
| 9 | Shim Chung | 1880 | f | math | 29 | 28 | 27 |

(cf) 관측수는 달라도 동일한 변수들을 갖는 둘 이상의 자료 결합

(두 개 SET 문장 사용한다면? ⇒ 엉뚱한 결과 초래)

DATA new1;

```
SET class2;
```

```
SET class1;
```

RUN;

⇒

| OBS | name | id_num | sex | dept | exam1 | exam2 | exam3 |
|-----|---------------|--------|-----|------|-------|-------|-------|
| 1 | Kong Jui | 1810 | f | cs | 20 | 20 | 30 |
| 2 | Pat Jui | 1808 | f | stat | 5 | 10 | 15 |
| 3 | Yun Heung Boo | 1670 | m | math | 0 | 10 | 5 |

(예) 서로 다른 변수들을 갖는 둘 이상의 자료 결합

(단일 SET 문장 사용 시)

DATA s1;

```
INPUT name $ 1-20 id_num sex $ dept $;
```

```
DATALINES;
```

```
Sung Min Young    2005 f stat
```

```
Sung Min Ji       2009 f ca
```

```
Hong Gil Dong     1750 m math
```

RUN;

DATA s2;

```

INPUT exam1 exam2 exam3;
DATALINES;
    30 30 40
    30 30 40
    29 28 27
RUN;
DATA credit2;
    SET s1 s2;
    score=exam1+exam2+exam3;
    IF score >= 50 THEN credit='Pass';
    ELSE credit='Fail';
    DROP id_num score;
RUN;

```

⇒

| OBS | name | sex | dept | exam1 | exam2 | exam3 | credit |
|-----|----------------|-----|------|-------|-------|-------|--------|
| 1 | Sung Min Young | f | stat | . | . | . | Fail |
| 2 | Sung Min Ji | f | ca | . | . | . | Fail |
| 3 | Hong Gil Dong | m | math | . | . | . | Fail |
| 4 | | | | 30 | 30 | 40 | Pass |
| 5 | | | | 30 | 30 | 40 | Pass |
| 6 | | | | 29 | 28 | 27 | Pass |

[참고] SAS에서, 숫자변수의 분실값은 마침표(.), 문자변수의 분실값은 빈칸으로 출력된다.

(7) 끼워넣기

: 둘 이상의 자료에 공통된 변수를 기준으로 순서화하여 합침

[참고] PROC SORT;

끼워넣기 작업 전에 먼저 결합할 자료들을 특정한 순서화 변수 기준으로 정렬해야 한다. 순서화 작업은 SORT procedure 활용한다.

(예) DATA class1;

```

INPUT name $ 1-20 id_num sex $ dept $ exam1 exam2 exam3;
DATALINES;

```

```

Sung Min Young      2005 f stat 30 30 40
Sung Min Ji         2009 f cs   30 30 40
Hong Gil Dong       1750 m math 29 28 27

```

RUN;

DATA class2;

```

INPUT name $ 1-20 id_num sex $ dept $ exam1 exam2 exam3;
DATALINES;

```

```

Kong Jui          1810 f cs    20 20 30
Pat Jui           1808 f stat   5 10 15
Yun Heung Boo     1670 m math   0 10 5
Yun Nol Boo       1665 m cs    20 30 10
Sun Nyeo          0300 f stat   25 25 35
Shim Chung        1880 f math   29 28 27

```

```
RUN;
```

```
PROC SORT DATA=class1;
```

```
  BY id_num;
```

```
RUN;
```

```
PROC SORT DATA=class2;
```

```
  BY id_num;
```

```
RUN;
```

```
DATA whole;
```

```
  SET class1 class2;
```

```
  BY id_num; /* BY 문장 사용하기 전에 BY 문장에서 지정된 변수를
              기준으로 (SORT 프로시저에서) 먼저 정렬해야 함 */
```

```
RUN;
```

⇒

| OBS | name | id_num | sex | dept | exam1 | exam2 | exam3 |
|-----|----------------|--------|-----|------|-------|-------|-------|
| 1 | Sun Nyeo | 300 | f | stat | 25 | 25 | 35 |
| 2 | Yun Nol Boo | 1665 | m | cs | 20 | 30 | 10 |
| 3 | Yun Heung Boo | 1670 | m | math | 0 | 10 | 5 |
| 4 | Hong Gil Dong | 1750 | m | math | 29 | 28 | 27 |
| 5 | Pat Jui | 1808 | f | stat | 5 | 10 | 15 |
| 6 | Kong Jui | 1810 | f | cs | 20 | 20 | 30 |
| 7 | Shim Chung | 1880 | f | math | 29 | 28 | 27 |
| 8 | Sung Min Young | 2005 | f | stat | 30 | 30 | 40 |
| 9 | Sung Min Ji | 2009 | f | cs | 30 | 30 | 40 |

(8) 복습

(예) DATA a1;

```
  INPUT name $ sex $ @@;
```

```
  DATALINES;
```

```
    Sung M    Park F    Kim F    Lee F
```

```
RUN;
```

DATA a2;

```
  INPUT name $ sex $ @@;
```

```
  DATALINES;
```

```
    Moon F    Yoon M    Oh    M    Jang F
```

```
RUN;
```

DATA b;

```
INPUT name $ age @@;
```

```
DATALINES;
```

```
Sung 33 Park 23 Kim 12 Lee 45
```

```
RUN;
```

```
DATA c;
```

```
INPUT name $ sex $ age @@;
```

```
DATALINES;
```

```
Sung F 33 Park M 23 Lee F 45
```

```
RUN;
```

| OBS | name | sex | OBS | name | sex | OBS | name | age | OBS | name | sex | age |
|-----|------|-----|-----|------|-----|-----|------|-----|-----|------|-----|-----|
| 1 | Sung | M | 1 | Moon | F | 1 | Sung | 33 | 1 | Sung | F | 33 |
| 2 | Park | F | 2 | Yoon | M | 2 | Park | 23 | 2 | Park | M | 23 |
| 3 | Kim | F | 3 | Oh | M | 3 | Kim | 12 | 3 | Lee | F | 45 |
| 4 | Lee | F | 4 | Jang | F | 4 | Lee | 45 | | | | |

<a1> <a2> <c>

```
DATA d;
```

```
SET a1 a2;
```

```
RUN;
```

⇒

| OBS | name | sex |
|-----|------|-----|
| 1 | Sung | M |
| 2 | Park | F |
| 3 | Kim | F |
| 4 | Lee | F |
| 5 | Moon | F |
| 6 | Yoon | M |
| 7 | Oh | M |
| 8 | Jang | F |

```
DATA e;
```

```
SET a2 b;
```

```
RUN;
```

⇒

| OBS | name | sex | age |
|-----|------|-----|-----|
| 1 | Moon | F | . |
| 2 | Yoon | M | . |
| 3 | Oh | M | . |
| 4 | Jang | F | . |
| 5 | Sung | | 33 |
| 6 | Park | | 23 |
| 7 | Kim | | 12 |
| 8 | Lee | | 45 |

```
DATA f;
```

```
SET a1 c;
```

```
RUN;
```

⇒

| OBS | name | sex | age |
|-----|------|-----|-----|
| 1 | Sung | M | . |
| 2 | Park | F | . |
| 3 | Kim | F | . |
| 4 | Lee | F | . |
| 5 | Sung | F | 33 |
| 6 | Park | M | 23 |
| 7 | Lee | F | 45 |

DATA g;

SET a1;

SET b;

RUN;

⇒

| OBS | name | sex | age |
|-----|------|-----|-----|
| 1 | Sung | M | 33 |
| 2 | Park | F | 23 |
| 3 | Kim | F | 12 |
| 4 | Lee | F | 45 |

첫 번째 SET 문장이 실행되면 자료 a1로부터 첫 번째 관측을 읽어 변수 name에 Sung, 변수 sex에 M이 들어온다. 두 번째 SET 문장이 실행되면 자료 b의 첫 번째 관측을 읽어 변수 name에 Sung, 변수 age에 33이 들어온다. 이렇게 두 자료에서 첫 번째 관측 입력이 끝나면 다시 DATA 단계 처음으로 되돌아가 각 자료의 두 번째 관측들을 입력하게 된다. 자료 a1과 b에는 변수 name이 공통인데, 이런 경우에는 **나중에(데이터셋 b에) 입력되는 name의 데이터 값이 살아있게 된다.** 물론, 이 예제의 경우에는 어느 자료든 변수 name 값이 같으므로 구별이 되지는 않는다.

DATA h;

SET a1;

SET c;

RUN;

⇒

| OBS | name | sex | age |
|-----|------|-----|-----|
| 1 | Sung | F | 33 |
| 2 | Park | M | 23 |
| 3 | Lee | F | 45 |

자료 a1의 관측수는 4개, 자료 c의 관측수는 3개이므로 자료 h의 관측수는 3개(관측수가 작은 자료의 관측수)가 된다. 그리고 자료 c가 a1보다 나중에 입력되었으므로 동일한 이름의 변수에 대한 데이터 값은 모두 자료 c에서 들어온다.

(9) END 변수의 지정 <<----- Skip

: END 변수는 SET 문장의 선택사항으로 지정되는 것으로, 데이터 입력이 모두 끝난 후에 추가로 처리할 루틴이 있는 경우에 사용

`SET SASdataset END=variable;`

END=*variable* 현재 입력 관측이 입력 파일의 가장 마지막일 때 그 값을 1로 놓을 변수 이름을 지정한다. 마지막 관측이 아니면 *variable* 값은 0 이다. 이 변수는 자료로 출력되지 않으며, 이 지정은 마지막 관측의 입력 이후에 다시 다른 형태로 입력 처리할 내용이 있을 때 사용한다.

(예) DATA mask4c;

SET mask4 END=eof;

OUTPUT;

IF eof THEN DO;

P_loff=.

DO WL_Ratio=4.28 TO 7.40 BY 0.01;

DO AreaNew=3 TO 5 BY 0.05;

OUTPUT;

END;

END;

END;

RUN;

PROC RSREG DATA=mask4c OUT=mask4c NOPRINT;

MODEL P_loff=WL_Ratio AreaNew / PREDICT;

RUN;

/* 자료 mask4를 전부 읽어들인다. mask4의 마지막 관측을 읽고나면 변수 eof의 값은 1이 된다. eof=1 이라 함은 곧 eof의 진리값이 참에 해당한다. 따라서 이 경우에는 THEN 이하의 문장들을 실행한다. */

/* 이런 형식의 프로그램은 회귀분석(PROC REG)이나 반응표면분석(PROC RSREG)에서 예측값(predicted value)을 계산케 하여 삼차원표면도 등 적합한 모형의 그래프를 그리는 경우에 자주 활용된다.*/

5.2.10 SAS 자료의 랜덤 처리 <<----- Skip

지금까지는 자료의 순차처리(sequential processing)에 관련된 SET 문장의 용법을 해설하였으나, 때로 자료의 랜덤처리(random processing 또는 직접처리(direct

processing))가 요구되는 경우가 있다.

SET SASdataset POINT=variable;

POINT=variable 처리할 **관측 번호**를 지정하는 변수 이름 선언

(예) 자료 a에서 8번, 16번, 2번, 35번 관측들만 따로 떼어 새로운 자료 b를 만들려면?

⇒ DATA b;

DO k=8, 16, 2, 35;

SET a POINT=k;

OUTPUT; /* output 문장이 수행되어도 point 변수는 자료로
출력되지 않음 */

END;

STOP; /* 이것이 생략되면 DATA 단계는 끝없이 반복 수행됨 */
/* 순차처리의 경우에는 자료의 끝에 이르면 자동적으로
읽기를 중단하지만, 랜덤처리의 경우에는 언제 자료 처리를
중단할지 명확히 선언하지 않으면 DATA 단계를 벗어날 수
없다. (POINT 옵션과 함께 SET 문장을 사용하면 랜덤처리
형식으로 데이터셋을 읽으므로 POINT 옵션을 사용하려면
STOP 문장이 필요함) */

RUN;

[cf] SET SASdataset (FIRSTOBS=n OBS=m);

/* n번째 관측부터 m번째 관측까지만 가져옴 */

5.2.11 MERGE 문장

둘 이상의 자료들을 **관측별로** 병합(merge)하여 새로운 자료를 만든다. 자료들에 공통적인 변수가 있으면 공통 변수를 BY 문장에서 지정할 수 있는데, BY 문장을 사용하려면 SORT 절차를 먼저 호출하여 데이터를 순서화해야 한다.

MERGE SASdataset SASdataset ... ;

SASdataset 들은 병합할 SAS 자료들의 이름

[cf] SET SASdataset SASdataset ... ; (단일 SET 문장)

- ▶ 단일 SET 문장은 자료들을 **세로로** 합한다(덧붙인다).
(반면, MERGE 문장은 **가로로** 합한다(병합한다).)
- ▶ 단일 SET 문장은 자료를 결합하되 **변수를 기준으로** 한다.
(반면, MERGE 문장은 **관측을 기준으로** 한다.)

| |
|----------|
| SAS 자료 a |
| SAS 자료 b |

SET a b;
(변수 기준의 결합)

| | |
|----------|----------|
| SAS 자료 a | SAS 자료 b |
|----------|----------|

MERGE a b;
(관측 기준의 결합)

(1) 일대일 병합

: BY 문장이 뒤따르지 않는 MERGE 문장으로 결과되는 병합

[참고] 각 자료의 관측수가 다르면 그 중 관측수가 많은 쪽 기준으로 병합
즉, 관측수가 모자라는 변수들의 데이터 값은 모두 분실값으로 처리됨

(예) DATA owner;

INPUT name \$;

DATALINES;

Naekyung

Minyoung

Aree

RUN;

DATA car;

INPUT year make \$ model \$;

DATALINES;

1993 Kia Sephia

2005 Hyungdai Sonata

2001 Kia Optima

RUN;

DATA color;

INPUT color \$ @@;

DATALINES;

Purple Silver Pearl Red

RUN;

| OBS | name |
|-----|----------|
| 1 | Naekyung |
| 2 | Minyoung |
| 3 | Aree |

<owner>

| OBS | year | make | model |
|-----|------|----------|--------|
| 1 | 1993 | Kia | Sephia |
| 2 | 2005 | Hyungdai | Sonata |
| 3 | 2001 | Kia | Optima |

<car>

| OBS | color |
|-----|--------|
| 1 | Purple |
| 2 | Silver |
| 3 | Pearl |
| 4 | Red |

<color>

DATA driver;

MERGE owner car color;

RUN;

⇒

| OBS | name | year | make | model | color |
|-----|----------|------|---------|--------|--------|
| 1 | Naekyung | 1993 | Kia | Sephia | Purple |
| 2 | Minyoung | 2005 | Hyundai | Sonata | Silver |
| 3 | Aree | 2001 | Kia | Optima | Pearl |
| 4 | . | . | . | . | Red |

- [참고] - 자료들 간 겹치는 변수가 존재하면 가장 나중에 지정된 자료의 데이터 값이 부여
- 일대일 병합은 자료에 수록된 관측 순서가 정확히 일치할 때만 실용성 있음 (관측 순서가 일치하는지 확실하지 않으면 다음의 BY 문장을 이용한 짝짓기 병합 활용)

(2) 짝짓기 병합

- : BY 문장으로 기준 변수를 지정하여 병합
- : 한 개체에 관한 자료가 둘 이상의 자료에 일정한 순서없이 뒤죽박죽 수록되어 있는 경우의 병합에 사용 (각 자료에 필히 짝짓기의 기준이 되는 공통 변수가 하나 이상 존재해야 함)

(예) DATA car;

INPUT name \$ year make \$ model \$;

DATALINES;

```
Naekyung    1993 Kia      Sephia
Minyoung    2005 Hyundai Sonata
Aree        2001 Kia      Optima
```

RUN;

DATA color;

INPUT name \$ color \$;

DATALINES;

```
Minyoung    Purple
Naekyung    Silver
Aree        Pearl
Minji       Red
```

RUN;

/* Merge 하기 전에 SORT 절차를 통한 자료의 순서화가 선행되어야함 */

/* SORT 절차를 통해 자료를 순서화하지 않고 merge를 실행하면 다음과 같은 오류 메시지가 Log 창에 뜬다.

ERROR: By variables are not properly sorted on data set
WORK.CAR. */

PROC SORT DATA=car; /* 디폴트: 오름차순(ascending) 정렬 */
BY name; /* BY 변수는 두 개 이상을 지정할 수도 있다. */

RUN;

PROC SORT DATA=color;

BY name; /* BY 변수는 두 개 이상을 지정할 수도 있다. */

RUN;

DATA driver;

MERGE car color;

BY name; /* BY 변수는 두 개 이상을 지정할 수도 있다. */

RUN;

⇒

| OBS | name | year | make | model |
|-----|----------|------|---------|--------|
| 1 | Aree | 2001 | Kia | Optima |
| 2 | Minyoung | 2005 | Hyundai | Sonata |
| 3 | Naekyung | 1993 | Kia | Sephia |

<car>

| OBS | name | color |
|-----|----------|--------|
| 1 | Aree | Pearl |
| 2 | Minji | Red |
| 3 | Minyoung | Purple |
| 4 | Naekyung | Silver |

<color>

| OBS | name | year | make | model | color |
|-----|----------|------|---------|--------|--------|
| 1 | Aree | 2001 | Kia | Optima | Pearl |
| 2 | Minji | . | . | . | Red |
| 3 | Minyoung | 2005 | Hyundai | Sonata | Purple |
| 4 | Naekyung | 1993 | Kia | Sephia | Silver |

<driver>

5.2.12 OUTPUT 문장

OUTPUT 문장이 나오기 직전까지 입력된 데이터 값들을 지정된 자료의 관측으로 출력시킨다. 필요에 따라서 하나의 DATA 단계 내에 몇 번이라도 나타날 수 있다.

OUTPUT SASdatasets;

SASdatasets 는 현재 데이터 값들이 관측으로 출력될 SAS 자료의 이름들로, 2 개 이상 지정도 가능, 경우에 따라 생략도 가능

(예) OUTPUT 문장이 필요 없는 경우:

DATA sample;

INPUT x y @@;

```

z=x+y;
DATALINES;
    1  23  2  15  3  36

```

RUN;

⇒ 첫 번째 관측을 읽어 변수 x, y 값을 입력하고 z 값을 계산한 다음 DATALINES 외에 더 이상의 문장이 존재하지 않으므로 지금까지의 결과를 일단 자료 sample 로 출력(output)한다. 그리고 다시 첫 문장 INPUT 으로 되돌아가 두 번째 관측을 처리한다. 즉, 이런 DATA 단계에서는 DATALINES 문장과 만나는 순간 명시하지 않았어도 자동적으로 OUTPUT 문장이 한 번 실행된다. 즉, OUTPUT 문장이 묵시적으로 사용된 경우이다.

(예) DATA 단계에서 동시에 둘 이상의 자료를 생성하는 경우:

DATA stat math;

```
INPUT course $ name $ score1 score2 score3;
```

```
DROP course; /* 자료 stat과 math 양쪽에 다 유효 */
```

```
IF course='Stat' THEN OUTPUT stat;
```

```
ELSE OUTPUT math;
```

```
* (참고) OUTPUT ;
```

```
* (참고) OUTPUT stat math;
```

```
DATALINES;
```

```

Stat Sung      30 30 40
Math Kim       20 20 30
Stat Lee       10 10 20
Math Park      25 25 35

```

RUN;

⇒ DROP 문장에서 지정된 변수들은 생성될 자료에 포함되지 않으며, 이 문장은 실행문이 아닌 선언문이므로 나타나는 위치는 중요하지 않다. IF 문장에서는 만일 course가 'Stat'이면 읽어들이는 관측을 자료 stat으로, 아니면 math로 출력(output)시킨다. OUTPUT 문장이 사용되면 DATA 단계를 한 문장씩 처리해 나가다가 OUTPUT 문장과 만날 때만 그 동안 입력된 데이터 값들을 하나의 관측으로 만들어 자료로 출력한다.

```
PROC PRINT DATA=stat; /* 출력창에 stat 이란 자료 출력 */
```

```

RUN; /* DATA=stat 지정하지 않으면 가장 최근에
      생성된 자료 출력됨 */

```

⇒

| OBS | name | score1 | score2 | score3 |
|-----|------|--------|--------|--------|
| 1 | Sung | 30 | 30 | 40 |
| 2 | Lee | 10 | 10 | 20 |

(예) DATA newstat;

SET stat;

DROP score1-score3;

score=score1;

OUTPUT; /* 또는 OUTPUT newstat; */

/* 생성되는 자료가 newstat 하나이므로 newstat 은 생략 가능 */

score=score2;

OUTPUT; /* 또는 OUTPUT newstat; */

score=score3;

OUTPUT; /* 또는 OUTPUT newstat; */

RUN;

⇒

| OBS | name | score |
|-----|------|-------|
| 1 | Sung | 30 |
| 2 | Sung | 30 |
| 3 | Sung | 40 |
| 4 | Lee | 10 |
| 5 | Lee | 10 |
| 6 | Lee | 20 |

새로운 변수 score 에 score1 값을 대입하고 그때까지 입력된 모든 변수들에 대한 데이터 값들을 하나의 관측으로 출력되고, 그 다음 변수 score 가 갖고 있던 score1 값은 score2 값으로 대체되어 하나의 관측으로 출력되고, 마지막으로 변수 score 가 갖고 있던 score2 값은 score3 값으로 대체되어 세 번째 관측으로 출력된다.

그런 다음 stat 의 두 번째 관측에 대한 처리가 반복된다.

물론, score1-score3 변수들은 drop 된다.

5.2.13 DROP 문장

SAS 자료에 포함시키고 싶지 않은 변수를 지정한다.

KEEP 문장과 상반된 역할을 수행하며, 이 문장은 실행문이 아닌 선언문이므로 DATA 단계 내 어디에 나타나도 상관없다.

DROP *variables*;

variables 는 빼고 싶은 변수들의 이름

(예) 생성될 자료 모두에 다 유효:

DATA stat math; /* DATA stat(DROP=course) math(DROP=course); */

INPUT course \$ name \$ score1 score2 score3;

```

DROP course; /* 자료 stat 과 math 양쪽에 다 유효 */
IF corse='Stat' THEN OUTPUT stat;
      ELSE OUTPUT math;
DATALINES;
      Stat Sung      30 30 40
      Math Kim       20 20 30
      Stat Lee       10 10 20
      Math Park      25 25 35
RUN;

```

(예) 생성될 자료 각각에 적용:

```

DATA stat(DROP=course) math(DROP=name);
      /* 자료 stat에서는 변수 course 제거,
      자료 math에서는 변수 name 제거 */
INPUT course $ name $ score1 score2 score3;
IF course='Stat' THEN OUTPUT stat;
      ELSE OUTPUT math;
DATALINES;
      Stat Sung      30 30 40
      Math Kim       20 20 30
      Stat Lee       10 10 20
      Math Park      25 25 35
RUN;

```

5.2.14 KEEP 문장

DROP 문장과 정반대의 효과로, SAS 자료에 포함할 변수를 지정한다.

| |
|-------------------------|
| KEEP <i>variables</i> ; |
|-------------------------|

variables 는 포함시키고 싶은 변수들의 이름

5.2.15 DELETE 문장

현재 관측에 관한 모든 SAS 문장의 실행을 중지시키고 현재 관측을 SAS 자료에 포함시키지 않는다.

DELETE;

(예) DATA males;

INPUT name \$ sex \$ age;

IF sex='F' THEN DELETE;

type='Friend'; /* 삭제되는 관측에 대해서는 이 문장은 실행되지 않음*/

DATALINES;

Sung M 33

Moon F 29

Go M 34

RUN;

⇒

| OBS | name | sex | age | type |
|-----|------|-----|-----|--------|
| 1 | Sung | M | 33 | Friend |
| 2 | Go | M | 34 | Friend |

5.2.16 ABORT 문장

현재 DATA 단계의 실행을 중지시키고 그 직전까지 입력된 데이터를 SAS 자료로 출력한 다음 그 다음 단계로 넘어가게 한다. 경우에 따라서는 SAS 시스템을 종료시킬 수도 있다.

ABORT [RETURN];

ABORT RETURN을 사용하면 SAS 시스템이 완전히 종료됨

(예) DATA division;

INPUT x y @@;

IF y=0 THEN ABORT; * (cf) ABORT RETURN;

z=x/y;

DATALINES;

1 2 2 0 2 1

RUN;

⇒ 두 번째 관측의 y 값이 0 이므로 ABORT가 실행됨과 동시에 DATA 단계는 끝나지만 _ERROR_=1 생성하여 오류 메시지가 로그창에 나온다. 즉, 로그창에 z 값은 결측으로 표시된다. 그리고 division 이라는 데이터셋은 그 전까지 읽고 생성된 관측치들만으로 구성된 데이터셋이 만들어진다.

⇒

| OBS | x | y | z |
|-----|---|---|-----|
| 1 | 1 | 2 | 0.5 |

```

ERROR: ABORT 문장에 의해 실행이 중단되었습니다.(위치: 행 198 칼럼 13)
RULE: -----1-----2-----3-----4-----5-----6-----
201      1 2 2 0 2 1
x=2 y=0 z=, _ERROR_=1 _N_=2
NOTE: 오류가 발생하여 SAS 시스템은 현재 스텝의 실행을 중지합니다.
WARNING: 데이터 셋 WORK.DIVISION이(가) 불완전합니다. 이 스텝은 1개의 관측치와
3개의 변수가 있을 때 중단되었습니다.
  
```

[참고] ABORT 문장은 통계 자료를 수집하여 파일로 만든 후 입력 오류가 있는지 여부 등을 검색할 때 주로 사용된다.

(e.g.) 숫자 데이터 값에 문자가 잘못 들어가지 않았는지,
일정 범위의 값을 갖는 숫자 데이터 값에서 범위를 벗어나는 값이 입력되지 않았는지 등 확인

5.2.17 STOP 문장

DATA 단계의 실행을 중지시키는 기능을 하지만, ABORT와 다른 점은 **로그창에 오류메시지를 생성하지 않는다**는 점이다.

STOP;

(예) DATA division;

INPUT x y @@;

IF y=0 THEN STOP;

z=x/y;

DATALINES;

1 2 2 0 2 1

RUN;

⇒ 두 번째 관측의 y 값이 0 이므로 STOP이 실행된다. 따라서 z=x/y; 는 실행되지 않고 **오류 메시지 없이** 현 DATA 단계를 빠져나가고 다음 단계로 넘어간다.

⇒

| OBS | x | y | z |
|-----|---|---|-----|
| 1 | 1 | 2 | 0.5 |

5.2.18 PROC TRANSPOSE <<-- 주제별 topic 에서

5.3 제어 문장

DATA 단계에 나열된 SAS 문장들의 실행 흐름을 선택적으로 조절하는 목적으로 사용된다. SAS 시스템에서 지원하는 제어 문장에는 다음과 같은 것들이 있다.

- IF, SELECT, WHERE 같은 조건문장
- DO 와 같은 반복 문장
- GOTO, LINK 와 같이 실행 흐름을 건너뛰는 문장 등

5.3.1 IF - THEN/ELSE 문장

뒤따르는 조건 수식의 결과에 따라 SAS 문장의 실행 흐름을 선택적으로 제어한다. 사용 형식은 다음과 같다.

| |
|--|
| ① IF <i>expression</i> ; |
| ② IF <i>expression</i> THEN <i>statement</i> ; |
| ③ IF <i>expression</i> THEN <i>statement</i> ; ELSE <i>statement</i> ; |
| ④ IF <i>expression</i> THEN <i>statement</i> ; ELSE IF <i>expression</i> THEN <i>statement</i> ; |
| ELSE <i>statement</i> ; |

expression 은 적법한 SAS 수식

statement 는 SAS 문장

- ① 은 부분집합화 IF (subsetting IF)로, 입력 관측들 중 IF 다음 조건에 합치되는, 즉, IF 다음의 수식이 참인 관측들만 골라 새로운 SAS 자료를 만들 때 사용
- ② 는 IF 다음 수식이 참일 경우에 한하여 THEN 이후 문장을 실행
- ③ 은 IF 다음 수식이 참이면 THEN 이후 문장을 실행하고, 거짓이면 ELSE 이후 문장을 실행
- ④ 은 IF 다음 수식이 참이면 THEN 이후 문장을 실행, 그 외 조건 중 IF ELSE 다음 수식 참이면 THEN 이후 문장 실행,, 위 조건 모두 거짓이면 ELSE 이후 문장 실행

(예) ① 관련

```
DATA class;
  INPUT name $ sex $ age year @@;
  DATALINES;
      Sung M 33 1981   Park M 28 1986   Huh F 19 1995
      Moon F 29 1985   Ahn  M 34 1980   Kim M 42 1972
  RUN;
DATA males (DROP=year);
  SET class;
```



```
IF sex='M';
```

```
RUN;
```

⇒

| OBS | name | sex | age |
|-----|------|-----|-----|
| 1 | Sung | M | 33 |
| 2 | Park | M | 28 |
| 3 | Ahn | M | 34 |
| 4 | Kim | M | 42 |

(예) ② 관련

```
DATA new (DROP=year);
```

```
SET class;
```

```
IF sex='M' THEN sex='Male';
```

```
RUN;
```

⇒

| OBS | name | sex | age |
|-----|------|------|-----|
| 1 | Sung | Male | 33 |
| 2 | Park | Male | 28 |
| 3 | Huh | F | 19 |
| 4 | Moon | F | 29 |
| 5 | Ahn | Male | 34 |
| 6 | Kim | Male | 42 |

(예) ③ 관련

```
DATA class (DROP=year);
```

```
SET class;
```

```
IF sex='M' THEN sex='Male';
```

```
ELSE sex='Female'; * IF~ THEN~ 문장없이 독립적으로 사용 불가;
```

```
RUN;
```

⇒

| OBS | name | sex | age |
|-----|------|--------|-----|
| 1 | Sung | Male | 33 |
| 2 | Park | Male | 28 |
| 3 | Huh | Female | 19 |
| 4 | Moon | Female | 29 |
| 5 | Ahn | Male | 34 |
| 6 | Kim | Male | 42 |

(예) ④ 관련

```
DATA class;
```

```
SET class;
```

```
IF sex='M' THEN sex='Male';
```

```
ELSE sex='Female';
```

```
IF MOD(year, 400)=0 THEN February=28;
```

```
ELSE IF MOD(year, 4)=0 THEN February=29;
```

```
ELSE February=28;
```

RUN;

⇒ 연도를 400으로 나누어 떨어지면 2월이 28일,
400으로 나누어 떨어지지 않는 4의 배수인 연도는 2월이 29일(윤년),
그 이외는 2월이 28일

[참고] MOD 는 나머지 값을 계산하는 SAS 함수

5.3.2 DO - WHILE | UNTIL - END 문장

DO 이하부터 대응되는 END 문장 사이의 모든 SAS 문장들(DO group)을 하나의 단위 프로그램으로 실행하고자 할 때 사용한다. DO 그룹은 또 다른 DO 그룹을 포함할 수 있으며 개수 제한이 없다.

단순 DO, 반복 DO, DO WHILE, DO UNTIL 문장 등 여러 유형이 있으나, DO 그룹의 끝은 반드시 END 문장으로 마감한다.

(1) 단순 DO (simple DO) 문장

: DO 그룹 내 SAS 문장들을 한 단위로 묶어 처리할 때 사용

| |
|--------------------------------------|
| DO; <i>SAS statements</i> END; |
|--------------------------------------|

SAS statements 는 실행 가능한 SAS 문장들

[참고] DO 문장은 흔히 IF-THEN/ELSE 문장과 연계되어 사용된다.

IF 문장에서 THEN 이나 ELSE 다음에는 원래 하나의 실행문만 지정할 수 있으나 둘 이상의 실행문이 필요할 경우에는 단순 DO 문장을 사용하여 둘 이상의 실행문들을 한 단위로 만들어 준다.

```
(예) IF k>0 THEN xsquare=x*x ;  
      PUT @20 x= @40 xsquare= ;  
      ELSE ysquare=y*y ;  
      PUT @20 y= @40 ysquare= ;
```

```
/* ELSE 문장과 대응되는 IF THEN 구문이 없다는 error 발생 */  
/* 둘 이상의 실행문을 사용하였으므로 잘못된 프로그램 */  
/* DO 문장을 활용하여 둘 이상의 실행문들을 한 단위로 만들어야함 */
```

```

⇒ IF k>0 THEN DO;
    xsquare=x*x;
    PUT @20 x= @40 xsquare= ; /* x= ⇔ "x=" x */
END;
ELSE DO;
    ysquare=y*y;
    PUT @20 y= @40 ysquare= ;
END;

```

/* k 값이 0보다 크면 변수 x를 제공하여 그 결과를 put하고,
아니면 변수 y 값을 제공하여 그 결과를 put */

```

⇔ IF k>0 THEN xsquare=x*x;
    ELSE ysquare=y*y;
IF k>0 THEN PUT @20 x= @40 xsquare= ;
    ELSE PUT @20 y= @40 ysquare= ;

```

/* 위의 프로그램과 결과는 동일하나 IF 문장을 많이 쓸수록 효율성이
떨어진다. */

(2) 반복 DO (iterative DO) 문장

: DO 그룹을 여러 번 반복 실행할 때 사용

```

DO index=start To stop BY inc [WHILE | UNTIL(expression)];
    SAS statements;
END;

```

SAS statements 는 실행 가능한 SAS 문장

expression 은 SAS 수식

index variable(지표변수)에 따라 DO 그룹의 반복 실행 횟수 달라짐

지표변수 값은 END 문장에 닿으면 증가분(변화분)만큼

증가(변화)해서 DO 문장으로 올라감

start 는 지표변수의 초기값

stop 은 지표변수의 상한값

inc 는 지표변수의 증가분(increment)/변화분으로, 디폴트는 +1

WHILE을 덧붙여 사용할 때는 WHILE 다음에 지정된 수식이 참일때에
한하여 DO 그룹 실행

UNTIL을 덧붙여 사용할 때는 UNTIL 다음 수식이 참이 될 때까지 DO 그룹
실행

[참고] TO와 BY 구문은 불필요하면 생략 가능

(예1) DO i=3;

j=i*i;

PUT i j;

END;

⇒ DO 그룹은 i=3 일 때 단 한 번 실행된다.

(결과 값은, i=3, j=9)

⇔ i=3;

j=i*i;

PUT i j;

(예2) DO i=1 TO 10 by 1;

.

(SAS 문장들)

.

END;

⇒ DO 그룹이 10번 실행 (∵ increment 디폴트는 +1)

(예3) DO count=1, 3, 5, 7, 9;

z=count+1;

END;

⇒ count 값이 1, 3, 5, 7, 9로 바뀌면서 DO 그룹이 5번 실행

(z의 결과 값은, 2, 4, 6, 8, 10로 바뀜)

(예4) DO i=1 TO 10 BY 2;

PUT i;

END;

⇒ i 가 1부터 10까지 2씩 증가하며 DO 그룹이 실행

(i 결과 값은, 1, 3, 5, 7, 9로 바뀜)

(예5) DO i=1 TO 3 , -2 TO -1;

PUT i ;

END;

⇒ 지표 변수가 변하는 범위를 동시에 두 가지 지정

i 값이 1, 2, 3, 그리고 -2, -1 로 바뀌며 DO 그룹이 실행

(예6) DO count='One', 'Two', 'Three';

PUT count;

END;

⇒ **지표 변수로 문자값을 사용**

(문자값은 따옴표로 둘러치고 문자값 사이에 쉼표를 사용)

count 는 One, Two, Three 로 값이 바뀌면서 DO 그룹 실행

(예7) k=0; n=5; m=0.5;

DO i=k TO n BY m;

y=SIN(i);

PUT i y;

END;

⇒ i가 0부터 5까지 0.5 단위로 증가하면서 SIN 함수 값 계산

(예8) DO i=1.23 TO 2.5 BY 0.023;

y=LOG(i);

PUT i y;

END;

⇒ 지시변수 i 값이 2.5를 넘지 않는 범위에서 1.23부터 0.023 단위로
증가하면서 LOG 함수 값 계산

(초기값, 상한값, 증가분 모두 실수 사용 가능)

(예9) k=1;

DO i=1 TO 500 WHILE(k<5);

k=k+1;

PUT i k;

END;

⇒ 지시변수 값이 500 이 넘지 않을 때까지 DO 그룹을 실행하되
동시에 k<5 조건도 만족하는 한 DO 그룹 반복 실행

1 2

2 3

3 4

4 5

(예10) k=1;

DO i=1 TO 500 UNTIL(k>=5);

k=k+1;

PUT i k;

END;

⇒ (예9)와 동일한 put 결과 (UNTIL 과 WHILE 은 보상적인 관계)

1 2

2 3

3 4

4 5

[참고] 지표변수로 주로 i 를 사용했지만 편의상 그런 것으로, 지표변수 이름은 사용자가 정의할 수 있다.

(예) DO k=1 TO 5;

. (SAS 문장들)

.

END;

PUT k;

⇒ 6

k 값이 1부터 5까지 1 단위로 증가하면서 DO 그룹이 실행된다. DO 그룹 밖에 PUT k; 가 있으므로 DO 문장을 벗어난 후에 k 값이 출력되므로 k=6. (∴ 지표변수는 END 문장 만나서 자동적으로 증가분(여기에서는 지정하지 않았으므로 +1) 만큼 증가해서 올라감)

(예) DATA random;

DO n=1 TO 10;

x=UNIFORM(0);

PUT n x; * 이 부분에 PUT 문장 대신 OUTPUT 문장있으면?;

END;

RUN;

⇒ 자료 random 에 포함된 것은 2개 변수(n, x)와 한 개 관측치

(n=11, x=(n이 10일 때 생성되는)난수)

Log 창에는 모두 10개의 균등난수가 출력되지만, 자료 random 에는 2개 변수와 한 개 관측만 존재한다. (∴ DO 그룹의 실행을 벗어나면서 비로써 DATA 단계가 단 한 번 실행되므로 DO 그룹을 벗어나면서 마지막으로 할당된 n 값과 x 값이 자료 random에 수록)

[참고] UNIFORM(*seed*)는 난수(random number)함수 (교재 4.4절 참고)

(0,1) 구간에서 균등분포를 따르는 변량을 만들

(e.g.) DO i=1 TO 50;

 x=UNIFORM(4169679);

END;

⇒ 균등분포를 따르는 변량이 50개 만들어지는데, 초기 값으로 0보다 큰 정수 4169679를 사용한다.

(예) DATA 단계에서 반복 DO 문장을 사용하면 지표변수도 SAS자료로 자동 출력되는데, 이를 억제하려면 DATA 문장에서 DROP을 사용한다.

```
DATA random (DROP=n);
```

```
    DO n=1 TO 10;
```

```
        x=UNIFORM(0);
```

```
        OUTPUT;     * 이 부분에 OUTPUT 문장 없는 경우와 결과 비교 ;
```

```
    END;
```

```
RUN;
```

(예) DO 그룹 내부에서 지표변수 값 변경 가능

```
DATA test;
```

```
    DO n=1 TO 10;
```

```
        x=UNIFORM(0);
```

```
        OUTPUT;
```

```
        IF x>0.9 THEN n=10;
```

```
    END;
```

```
RUN;
```

(예) DO 그룹 내에 또 다른 DO 그룹

```
DO i=1 TO 10;
```

```
    DO j=1 TO 10;
```

```
        DO k=1 TO 5;
```

```
            test=i*j*k;
```

```
            PUT i j k test;
```

```
        END;
```

```
    END;
```

```
END;
```

(3) DO WHILE 과 DO UNTIL 문장

- : DO 문장((2))에서 WHILE, UNTIL 구문과 동일한 기능을 갖지만, 차이점은
지표변수가 없는 것
- : DO WHILE 문장은 그 다음 수식 조건이 참인 경우에 DO 그룹 실행,
DO UNTIL 문장은 그 다음 수식 조건이 참이 될 때까지 DO 그룹이 실행
(서로 보상적 관계)

① DO WHILE(*expression*);
 SAS statements
END;

② DO UNTIL(*expression*);
 SAS statements
END;

(예) DATA one;

```
n=4;    /* n=4; 라면?? */
```

```
DO WHILE(n<3);
```

```
    nn=n*n;
```

```
    PUT n nn;
```

```
    n=n+1;  /* ⇔ n+1; */  /* [참고] 합산문장 p.65 참고 */
```

```
END;
```

```
RUN;
```

```
⇔ DATA one;
```

```
    n=4;    /* n=4; 라면?? */
```

```
    DO UNTIL(n>=3);
```

```
        nn=n*n;
```

```
        PUT n nn;
```

```
        n=n+1;  /* ⇔ n+1; */
```

```
    END;
```

```
    RUN;
```

5.3.3 SELECT - WHEN/OTHERWISE - END 문장

SELECT 문장은 지정된 변수 또는 수식의 값에 따라 여러 개의 SAS 문장 중 어느 하나를 골라서 실행하고자 할 때 사용한다.

[비교] IF~THEN/ELSE IF~THEN/ELSE 문장


```

SELECT (expression);
    WHEN (expression) statement,
    WHEN (expression) statement,
        .
        .
    OTHERWISE statement,
END;

```

OTHERWISE 는 생략 가능한 경우가 있으나 주의!! (예제들 참고)

(예1) DATA test;

INPUT a x y @@;

SELECT (a);

WHEN (1) z=x+y; /* 변수 a가 문자변수라면, WHEN (“값“) */

WHEN (2) z=x-y;

WHEN (3) z=x*y;

WHEN (4) z=x/y;

OTHERWISE z=y/x; /* ⇔ WHEN (5, 6) z=y/x; 로 대체 가능 */

/*그러나 OTHERWISE 문장 자체만 없애면 안됨

(∵ a=5, 6 인 경우에 대한 처리 방법이 없으므로 오류 발생)*/

/*그냥 OTHERWISE; 만 하면 a=5, 6 인 경우에는 z 값 결측으로 처리*/

/* IF-THEN/ELSE 문장으로 바꾸면??? */

END;

DATALINES;

1 2 3 2 3 4 3 4 5 4 5 10 5 10 5 6 3 9 2 10 5

RUN;

⇒ SELECT 다음의 괄호 속 수식을 계산하여 그 값과 일치하는 WHEN 다음의 수식을 찾아 뒤따르는 SAS 문장을 실행하되 합치되는 수식이 없으면 OTHERWISE 다음의 SAS 문장을 실행

| OBS | a | x | y | z |
|-----|---|----|----|------|
| 1 | 1 | 2 | 3 | 5.0 |
| 2 | 2 | 3 | 4 | -1.0 |
| 3 | 3 | 4 | 5 | 20.0 |
| 4 | 4 | 5 | 10 | 0.5 |
| 5 | 5 | 10 | 5 | 0.5 |
| 6 | 6 | 3 | 9 | 3.0 |
| 7 | 2 | 10 | 5 | 5.0 |

(예2) DATA test2;

INPUT a x y @@;

SELECT (a+1);

WHEN (1) DO; /* 두 개 이상 실행문 있으므로 단순 DO 구문 */

z=x+y;

zz=z*z;

END;

WHEN (3) z=x*y;

OTHERWISE; /* a+1 이 1 또는 3 이 아니면 아무것도 실행안함 */

END; /* OTHERWISE 문장은 생략 가능하나 없을 경우에
SELECT 수식 계산값이 WHEN 조건 어디에도
부합되지 않으면 오류 발생하므로 주의 */

DATALINES;

0 1 2 1 2 3 2 3 4 5 4 3

RUN;

| OBS | a | x | y | z | zz |
|-----|---|---|---|----|----|
| 1 | 0 | 1 | 2 | 3 | 9 |
| 2 | 1 | 2 | 3 | . | . |
| 3 | 2 | 3 | 4 | 12 | . |
| 4 | 5 | 4 | 3 | . | . |

(예3) 단순 DO 안에 새로운 SELECT 문장 포함된 경우

SELECT (a);

WHEN (1) DO;

SELECT (b);

WHEN (-1) DO;

c=x**3;

d=y**3;

END;

WHEN (-2) c=x*x;

WHEN (-3) d=y*y;

OTHERWISE;

END;

END;

WHEN (2) c=x*y;

WHEN (3) d=x/y;

OTHERWISE;

END;

5.3.4 WHERE 문장

지정된 특별한 조건에 합치되는 관측들만 선별적으로 읽어 들일 때 사용한다.

WHERE *whereexpression*;

*whereexpression*에는 SAS 수식을 지정
산술연산자(+, -, *, / 등),
비교연산자(=, ^=, <, <=, >, >= 등),
논리연산자(&, |, ^),
특수비교연산자(IN) 사용 가능

[참고] PROC 단계에서도 사용 가능한 문장

[cf] IF 문장 (PROC 단계에서는 사용 불가)

(예) PROC PRINT DATA=saram;

WHERE sex="Male"; /* IF sex="Male"; 은 오류 */

RUN;

(예) DATA namja;

SET saram;

WHERE sex='Male';

RUN;

⇒ 자료 saram으로부터 변수 sex 값이 Male 인 관측들만 읽어 자료 namja
출력

⇔ DATA namja;

SET saram;

IF sex='Male';

RUN;

⇒ 위에서 WHERE 문장을 사용한 경우와 똑같은 결과를 도출하지만,
WHERE 문장을 사용하면 관측들이 DATA 단계로 들어오기 전에
조건에 맞는 관측들을 미리 고르지만,
부분집합화 IF를 사용하면 관측들이 일단 DATA 단계로 들어온 뒤에
조건에 맞는 관측들을 고른다는 차이점이 있다.

(WHERE 문장이 더 효율적)

(예) DATA metro;

SET saram;

WHERE city IN ('Seoul', 'Busan');

/* ⇔ IF city='Seoul' OR city='Busan'; */

/* ⇔ IF city IN ('Seoul', 'Busan'); */

RUN;

⇒ 자료 saram 에서 변수 city 값이 Seoul 이나 Busan 인 관측들만 읽음

5.3.5 GO TO 또는 GOTO 문장 / GOTO-RETURN

지정된 레이블(label)이 있는 곳으로 SAS 문장의 실행을 건너뛰게 한다.

GO TO *label* ;

[참고] - GOTO (또는 GO TO) 문장과 함께 RETURN 문장이 나오면 SAS 문장의 실행을 무조건 DATA 단계의 처음으로 보낸다.

- LINK 문장(교재 5.3.6절)도 GOTO 문장과 비슷한 역할을 하지만, RETURN 문장을 만나면 SAS 문장의 실행을 LINK 직후로 보낸다는 차이점이 있다. (RETURN 문장과 항상 함께 사용)

[참고] *label* 은 변수명 짓는 규칙과 동일

(예) DATA division;

INPUT x y @@;

IF y=0 THEN GOTO noway ; /* 레이블(label)은 noway */

z=x/y;

noway: /* 레이블에는 콜론(:)을 덧붙여 변수 이름과 구별 */

q=x+y;

DATALINES;

1 2 2 0 2 1

RUN;

⇒ y 값이 0 이라 나눗셈이 불가능할 때 레이블 noway 위치로 실행을 이동하여 오류를 방지하는 프로그램.

레이블에는 콜론(:)을 덧붙여 변수 이름과 구별.

(그러나 GOTO 문장에 지정하는 레이블에는 콜론 덧붙이지 않음)

⇒

| OBS | x | y | z | q |
|-----|---|---|-----|---|
| 1 | 1 | 2 | 0.5 | 3 |
| 2 | 2 | 0 | . | 2 |
| 3 | 2 | 1 | 2.0 | 3 |

(예) DATA division;

INPUT x y @@;

IF y=0 THEN GOTO noway;

z=x/y;

RETURN; ***<< RETURN 추가됨 ;

noway:

q=x+y;

DATALINES;

1 2 2 0 2 1

RUN;

⇒ y 가 0 이 아닌 경우에는 DATA 단계의 모든 문장이 실행되지 않고 RETURN 문장이 사용되어 y 가 0 이 아니면, 변수 q 의 연산이 실행되지 않는다.

⇒

| OBS | x | y | z | q |
|-----|---|---|-----|---|
| 1 | 1 | 2 | 0.5 | . |
| 2 | 2 | 0 | . | 2 |
| 3 | 2 | 1 | 2.0 | . |

5.3.6 LINK-RETURN 문장

LINK 문장을 만나면 지정한 레이블이 있는 위치로 이동하여 일련의 SAS 문장을 실행한 후 RETURN을 만나면 다시 원래 LINK 문장 직후의 SAS 문장을 실행한다.

`LINK label ;`

- [참고] - LINK 문장은 반드시 하나 이상의 RETURN 문장을 동반한다.
- LINK-RETRUN 그룹 내부에 또 다른 LINK-RETRUN 그룹 포함 가능
 - LINK 문장과 더불어 사용된 RETURN 의 경우, LINK 문장 직후의 SAS 문장으로 실행이 이동되고, 기타 경우 RETURN 은 DATA 단계 처음으로 이동하여 새로운 DATA 단계 실행 시작

(예) DATA division;

INPUT x y @@;

IF y=0 THEN LINK modify ;

z=x/y;

c='OK';

RETURN;

modify:

x=x+1;

y=y+1;

RETURN; /* DATA 단계 맨 끝에 있는 경우에는 생략 가능 */

DATALINES;

1 2 2 0

RUN;

⇒ 첫 번째 관측은 y 가 0 이 아니므로, z=x/y=0.5, c='OK'.

두 번째 관측에서는 y=0 이므로 실행이 modify 로 옮겨져 x=x+1=3, y=y+1=1을 실행한 후 RETURN 문장을 만났으므로 LINK 문장 직후의 SAS 문장인 z=x/y; 로 돌아간다. 즉, z=x/y=3/1=3, c='OK' 다 된다. 그리고 RETURN 문장을 만났으므로 DATA 단계의 처음으로 되돌아간다.

⇒

| OBS | x | y | z | c |
|-----|---|---|-----|----|
| 1 | 1 | 2 | 0.5 | OK |
| 2 | 3 | 1 | 3.0 | OK |

5.3.7 RETURN 문장

SAS 문장들의 계속적인 실행을 중지하고 미리 지정한 위치로 실행 지점을 이동시킨다.

RETURN;

[참고] LINK 문장과 더불어 사용된 RETURN 이라면 LINK 문장 직후의 SAS 문장으로 실행이 이동되고,
기타 경우에는 DATA 단계의 처음으로 이동하여 새로운 DATA 단계의 실행이 시작된다.

(예) 분모 값이 0일 때 나눗셈을 하지 못하게 하는 프로그램

DATA division;

INPUT x y;

IF y=0 THEN RETURN;

z=x/y;

DATALINES;

1 2

2 0

2 1

RUN;

⇒

| OBS | x | y | z |
|-----|---|---|-----|
| 1 | 1 | 2 | 0.5 |
| 2 | 2 | 0 | . |
| 3 | 2 | 1 | 2.0 |

5.4 기타 문장들

5.4.1 할당(assignment) 문장

등호 오른쪽 수식을 계산하여 산출된 값을 등호 왼쪽 변수에 넣는 형태의 문장들

variable = *expression* ;

(예) a=3;

```
a=a+2;  /* 우변의 a+2를 계산하여 좌변의 a 에 할당 */
        /* 우변의 a 값과 좌변의 a 값은 서로 다름 */
```

5.4.2 BY 문장

SET 문장 또는 MERGE 문장과 함께 사용되며, 순서화된 변수들 기준으로 작업할 때 사용된다.

BY [DESCENDING] *variables*;

variables 는 기준이 되는 변수(들) 지정

DESCENDING 옵션 지정하면 내림차순으로 처리됨 (디폴트는 오름차순)

[참고] DATA 단계에서 BY 문장을 사용하려면 SORT 절차를 이용하여 동일한 BY 변수를 기준으로 먼저 자료가 순서화되어 있어야 한다.

```
(예) PROC SORT DATA=a; BY DESCENDING x y; RUN;
      PROC SORT DATA=b; BY DESCENDING x y; RUN;
      DATA test;
        MERGE a b;
        BY DESCENDING x y;  /* 변수 x를 기준으로 내림차순으로, 변수 y를
                             기준으로 오름차순으로 처리 */
      RUN;
```

(예) 변수 x는 오름차순으로, 변수 y는 내림차순으로 순서화된 자료 처리하려면,
BY x DESCENDING y;

(예) 변수 x, y, 둘 다 내림차순으로 처리하려면,
BY DESCENDING x DESCENDING y;

5.4.3 RETAIN 문장

일반적으로 DATA 단계는 관측 단위로 실행되기 때문에 첫 번째 관측에 대한 실행 결과는 두 번째 관측에 대한 실행에 아무런 영향을 미치지 않는다. 그러나 때로는 선행 관측의 실행 결과에 따라 뒤따르는 관측의 실행 방향을 제어할 필요가 있는데, RETAIN 문장은 **방금 읽은 데이터 값을 계속 유지하여 다음 DATA 단계에서 이용하고 싶을 때** 사용하는 문장이다.

RETAIN *variables*;

variables 는 그 값을 유지하고 싶은 변수들 이름

- [참고] - 실행문이 아닌 선언문이므로 DATA 단계 어디에 삽입해도 된다.
- 바로 직전 DATA 단계에서 입력된 값만 유지하므로, 적어도 두 단계 이전의 변수 값을 기억하고 싶으면 LAG 함수를 사용한다.

(예) 동일한 관측이 겹쳐 입력된 자료에서 겹친 관측을 제거하는 프로그램
(변수 **name** 값이 같으면 동일 관측으로 간주하는 경우에 대한 예제)

```
DATA phone;
  INPUT name $ area $ number;
  DATALINES;
      Sung      02 4169679
      Moon      02 7976155
      Oh        0341 876322
      KETEL     02 3122868
      Moon      02 7976155
      Sung      02 4169679
  ;
RUN;
PROC SORT;
  BY name;
RUN; /* 위의 자료 phone 에는 동일한 관측이 존재하는데,
      RETAIN 문장을 사용하여 겹친 관측을 제거하려면 겹치는 관측들이
      이웃해 있어야 한다. 간편하게 SORT 프로시저를 활용하였다. */
DATA phone1;
  SET phone;
  IF oldname=name THEN DELETE;
  oldname=name;
  RETAIN oldname; /* 이 문장 없을 때와 비교 */
RUN;
```


⇒ SORT 프로시저를 이용하여 겹치는 동일한 관측을 서로 이웃하게
 순서화했다. 그 다음 DATA 단계에서는, oldname 과 name을 비교하여 이
 둘이 일치하면 현재 관측을 제거한다. 즉, 첫 번째 관측에서 oldname 은
 아직 정의되지 않은 상태이므로 oldname 값은 분실 데이터 값을 표시하는
 빈칸이 된다. Oldname 과 name 값이 서로 다르므로 그 다음 문장으로
 가서 oldname 에 방금 읽은 name 값 'KETEL'을 넣게 되고 이 값이 두
 번째 관측 입력때까지 유지된다. 두 번째 관측 입력할 때 oldname 값은
 'KETEL'이고 name 값은 'Moon' 이 된다.

⇒

| OBS | name | area | number | oldname |
|-----|-------|------|---------|---------|
| 1 | KETEL | 02 | 3122868 | KETEL |
| 2 | Moon | 02 | 7976155 | Moon |
| 3 | Oh | 0341 | 876322 | Oh |
| 4 | Sung | 02 | 4169679 | Sung |

[참고] NODUPKEY 옵션 사용하여 동일한 관측치 제거 가능

(예) DATA phone;

INPUT name \$ area \$ number;

DATALINES;

Sung 02 4169679

Moon 02 7976155

Oh 0341 876322

KETEL 02 3122868

Moon 02 7976155

Sung 02 4169679

RUN;

PROC SORT NODUPKEY;

BY name;

RUN;

PROC PRINT; RUN;

| OBS | name | area | number |
|-----|-------|------|---------|
| 1 | KETEL | 02 | 3122868 |
| 2 | Moon | 02 | 7976155 |
| 3 | Oh | 0341 | 876322 |
| 4 | Sung | 02 | 4169679 |

5.4.4 합산(sum) 문장

$variable + expression;$

variable 은 누적된 값이 들어가는 변수 이름
expression 은 적당한 수식

(예) DATA xequals0;

INPUT x @@;

IF x=0 THEN count+1 ; /* \Leftrightarrow count+(x=0) ; */

* (1) 초기값 부여하지 않으면 초기값은 0 ;

* (2) count = count + 1 ;

* (3) RETAIN (\Rightarrow 즉, 결과적으로 누적됨) ;

/* (cf) 잘못된 방식

count=0; IF x=0 THEN count=count+1; RETAIN count; */

DATALINES;

0 4 0 9 0 5

RUN;

\Rightarrow

| OBS | x | count |
|-----|---|-------|
| 1 | 0 | 1 |
| 2 | 4 | 1 |
| 3 | 0 | 2 |
| 4 | 9 | 2 |
| 5 | 0 | 3 |
| 6 | 5 | 3 |

0 이 아닌 x를 포함하는 관측에서도 count 값은 유지
 (내부적으로 RETAIN 문장이 실행되었으므로)

(예) a=5;

a+2; /* 단, 위에서 초기값 부여하지 않으면 초기값은 0 */

PUT a;

\Rightarrow 7

(예) 감산 (빼기) 연산은?

interest+(-debt); /* interest-debt; 는 틀린 표현 */

(예) x+y=0;

\Leftrightarrow x+(y=0); * 즉, if y=0 then x+1;

\Rightarrow (y=0) 이 참이면 y=0 이라는 비교연산 결과는 1 이 되고, 거짓이면 0 이라는 의미이므로 옳은 형태의 문장이다.

(즉, y=0 이면 x 값 하나씩 증가하여 누적)

(예) IF age < 50 THEN count+1;

\Leftrightarrow count + (age < 50);

5.4.5 LABEL 문장

DATA 단계에서 레이블을 붙이는데 사용한다. 레이블은 변수 이름에 붙어 다니며 일단 변수에 레이블을 붙여놓으면 각종 PROC 단계에서 변수 이름 대신 레이블이 출력되도록 지정할 수 있다.

```
LABEL variable="label" variable="label" ..... ;
```

variable 은 레이블을 붙이고 싶은 변수 이름

label 은 문자열로, (빈 칸을 포함하여) 최대 256자(문자열)까지 사용 가능

[cf] PROC 단계에서도 사용 가능

(단, DATA 단계에서 사용해야만 데이터셋에 레이블이 붙어서 저장됨)

(예) DATA phone;

```
INPUT name $ area $ number $;
```

```
LABEL area="Area Code" number="Phone Number";
```

```
/* 변수 area 와 변수 number 에 각각 레이블 붙임 */
```

```
DATALINES;
```

```
Sung (02) 416-9679
```

```
Moon (032) 797-6155
```

```
RUN;
```

```
PROC PRINT LABEL; /* PRINT 절차에서 변수 대신 레이블을 출력시키려면  
선택옵션에서 LABEL을 명시 */
```

```
RUN;
```

⇒

| OBS | name | Area Code | Phone Number |
|-----|------|--------------|-----------------|
| 1 | Sung | (02) | 416-9679 |
| 2 | Moon | (032) | 797-6155 |

5.5 자동변수 (automatic variable)

: SAS 시스템에서 용도가 이미 지정된 특별한 변수들

5.5.1 _ERROR_

DATA 단계에서 발생하는 오류의 수준을 나타내는 자동변수로, 입력 자료에 오류가 있으면 1, 없으면 0 값을 가진다. _ERROR_ 값이 1인 오류가 있는 관측은 항상 Log 창에 출력된다.

(예) DATA;

```
INPUT number code;
```

```
DATALINES;
```

```
561105 1024511
```

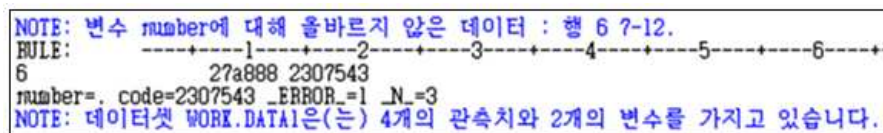
```
270911 2025649
```

```
27a888 2307543
```

```
456983 1098324
```

```
RUN;
```

⇒



NOTE: 변수 number에 대해 올바르지 않은 데이터 : 행 6 7-12.
RULE: -----1-----2-----3-----4-----5-----6-----
6
number= 27a888 2307543
number=, code=2307543 _ERROR_=1 _N_=3
NOTE: 데이터셋 WORK.DATA1은(는) 4개의 관측치와 2개의 변수를 가지고 있습니다.

[참고] _N_ 은 관측번호(?)를 나타내는 자동변수

(예) DATA에 오류가 있으면 실행을 중지하려면?

```
DATA;
```

```
INPUT number code;
```

```
IF _ERROR_=1 THEN STOP; /* ⇔ IF _ERROR_ THEN STOP; */
```

```
DATALINES;
```

```
1 3
```

```
2 x
```

```
RUN;
```

5.5.2 _N_

DATA 단계의 실행횟수 즉, 관측번호(?)를 나타내는 자동변수로, 프로그램에서 직접 이용 가능하다.

(예) DATA list;

```
INPUT a b;
```

```

FILE PRINT;
IF _N_=1 THEN PUT @21 "a" @31 "b" @40 '_N_' / @19 25*"-";
PUT @20 a @30 b @41 _N_;
DATALINES;
      100 200
      23 345
      776 15

```

RUN;

⇒

| a | b | _N_ |
|-----|-----|-----|
| 100 | 200 | 1 |
| 23 | 345 | 2 |
| 776 | 15 | 3 |

<<== PUT된 내용

(예) DATA choose1;

```

SET list;
FILE PRINT;
IF _N_=1 THEN PUT @21 'a' @31 'b' @40 '_N_' / @19 25*'-' ;
PUT @20 a @30 b @41 _N_;  **<<< (1) ;
IF _N_=2 THEN DELETE;      **<<< (2) ;

```

RUN;

⇒

| a | b | _N_ |
|-----|-----|-----|
| 100 | 200 | 1 |
| 23 | 345 | 2 |
| 776 | 15 | 3 |

<<== PUT된 내용

[참고] DELETE 문장보다 PUT 문장이 앞서므로 위와 동일한 PUT 결과

PROC PRINT DATA=choose1;

RUN;

⇒

| OBS | a | b |
|-----|-----|-----|
| 1 | 100 | 200 |
| 2 | 776 | 15 |

[참고] 자동변수 _N_ 는 DATA 단계에서만 유효하고 PROC 단계에서는 사용할 수 없으므로, PRINT 절차를 사용하여 출력된 결과에는 _N_ 값이 존재하지 않는다.

(예) DATA choose2;

```

SET list;
FILE PRINT;
IF _N_=1 THEN PUT @21 'a' @31 'b' @40 '_N_' / @19 25*'-' ;

```

```
IF _N_=2 THEN DELETE;          **<<< (2) ;
PUT @20 a @30 b @41 _N_;      **<<< (1) 순서 이렇게 바꾸면 ;
RUN;
```

⇒

| a | b | _N_ |
|-----|-----|-----|
| 100 | 200 | 1 |
| 776 | 15 | 3 |

<<== PUT된 내용

[참고] PUT 문장보다 DELETE 문장이 앞에 나옴

_N_에는 입력되는 관측 번호가 부여되어 삭제되는 관측이 있더라도 그 값이 변하지 않는다. 두 번째로 인쇄된 관측의 _N_ 값이 3인 것은 원래 데이터에서 3번째 관측이었음을 의미한다.

[참고] 자동변수 _N_은 각 DATA 단계마다 독립적으로 부여되며 DATA 단계가 끝나더라도 _N_ 값이 SAS 자료로 출력되지는 않는다.

5.5.3 _ALL_

DATA 단계에서는 반드시 PUT 문장과 함께 사용된다.

(cf) PROC PRINT DATA=one;

```
VAR _ALL_ ; /* One 이라는 데이터셋에 있는 모든 변수들 (자동변수 제외) */
RUN;        /* ( _ERROR_ 와 _N_ 제외 ) */
```

PUT _ALL_ ;

자동변수 _ERROR_와 _N_을 포함하여 현재 DATA 단계에서 정의된 모든 변수들의 값을 출력한다.

(예) DATA;

```
INPUT a b;
FILE PRINT;
PUT _ALL_;
DATALINES;
    100 200
    12d 300
    300 234
```

RUN;

⇒

| |
|-----------------------------|
| a=100 b=200 _ERROR_=0 _N_=1 |
| a=, b=300 _ERROR_=1 _N_=2 |
| a=300 b=234 _ERROR_=0 _N_=3 |

<<== PUT된 내용

5.6 분실값/결측값의 처리

통계 조사나 실험 등을 하다보면 어쩔 수 없이 분실 데이터 값이 나오게 마련인데, 이같이 빠진 데이터 값을 SAS 시스템에서는 분실값(missing value) 또는 결측값이라 한다.

결측값을 비결측값과 구분하여 입력하는 방식은 두 가지가 있는데,

- 입력 방식이 목록(자유)입력이라면, 결측값으로 마침표(.)를 반드시 사용
- 열입력 또는 포맷입력 방식이면, 빈 칸 또는 마침표

(예)

| name | sex |
|----------|-----|
| Nolboo | 0 |
| Sabangee | |
| | 1 |

<목록/자유입력 방식>

```
DATA;
```

```
INPUT name $ sex; /* sex=0, if 남자  
                  1, if 여자 */
```

```
DATALINES;
```

```
Nolboo 0
```

```
Sabangee .
```

```
. 1
```

```
RUN;
```

⇒

| OBS | name | sex |
|-----|----------|-----|
| 1 | Nolboo | 0 |
| 2 | Sabangee | . |
| 3 | | 1 |

[참고] 출력에서 결측값이 문자일 때는 빈 칸, 숫자일 때는 마침표가 출력

(cf) INFILE DATALINES MISSOEVER ;

지정한 변수에 대응하는 값이 존재하지 않는 경우에 입력 포인터가 다음 줄로 내려가는 것을 방지 (노트 p.19 (예) 참고)

<열입력 방식>

```
DATA;
```

```
INPUT name $ 1-8 sex 10; /* INPUT name $8. sex 2. */
```

```
DATALINES;
```

```
Nolboo 0
```

Sabangee .

. 1

RUN; /* 열입력 또는 포맷입력 방식의 경우에는 마침표(.) 또는 빈칸 ok */

⇒

| OBS | name | sex |
|-----|----------|-----|
| 1 | Nolboo | 0 |
| 2 | Sabangee | . |
| 3 | | 1 |

DATA;

INPUT name \$ 1-8 sex 10; /* INPUT name \$8. sex 2. */

DATALINES;

Nolboo 0

Sabangee

1

RUN; /* 열입력 또는 포맷입력 방식의 경우에는 마침표(.) 또는 빈칸 ok */

⇒

| OBS | name | sex |
|-----|----------|-----|
| 1 | Nolboo | 0 |
| 2 | Sabangee | . |
| 3 | | 1 |

(예) DATA 단계에서 결측 관측을 제거하려면?

IF string=" " THEN DELETE; /* string 은 문자변수 */

IF num=. THEN DELETE; /* num 은 숫자변수 */

(예) 사회과학 분야 통계조사에서는 무응답으로 처리된 각 문항에 관례적으로 9 또는 99 등의 특별한 숫자를 부여하는데 이를 결측치로 처리하려면?

IF response=9 THEN response=. ; /* response 는 숫자변수 */

IF response=99 THEN response=. ; /* response 는 숫자변수 */

(예) 특정한 문자열 또는 특정 범위를 벗어나는 변수값을 결측치로 처리하려면?

IF string="I am null" THEN string=""; /* string 은 문자변수 */

IF num>100 THEN num=. ; /* num 은 숫자변수 */

[참고] 문자변수 결측값은 빈칸, 숫자변수 결측값은 마침표(.)로 출력되지만, 숫자변수도 빈칸으로 출력하려면?

OPTIONS MISSING=' ';

5.7 데이터 입력 응용사례

[예1] 일원분산분석(one-way analysis of variance)

한 과수원에서 5가지 사과 품종(A, B, C, D, E)을 비교한다. 각 품종별로 7그루의 12년생 사과나무를 임의표집하여 수확량(단위: Kg)을 산출하였다.

| A | B | C | D | E |
|----|----|----|----|----|
| 13 | 27 | 40 | 17 | 36 |
| 19 | 31 | 44 | 28 | 32 |
| 39 | 36 | 41 | 41 | 34 |
| 38 | 29 | 37 | 45 | 29 |
| 22 | 45 | 36 | 15 | 25 |
| 25 | 32 | 38 | 13 | 31 |
| 10 | 44 | 35 | 20 | 30 |

DATA apple;

INPUT variety \$ yield; /* variety=품종, yield=수확량 */

DATALINES;

A 13

A 19

A 39

.

.

E 25

E 31

E 30

RUN;

⇔ DATA apple;

INPUT variety \$ yield @@;

DATALINES;

A 13 B 27 C 40 D 17 E 36

A 19 B 31 C 44 D 28 E 32

A 39 B 36 C 41 D 41 E 34

A 38 B 29 C 37 D 45 E 29

A 22 B 45 C 36 D 15 E 25

A 25 B 32 C 38 D 13 E 31

A 10 B 44 C 35 D 20 E 30

RUN;

⇔ DATA apple; /* DO 그룹 이용한 자료 입력 */

DO variety='A', 'B', 'C', 'D', 'E';

```

INPUT yield @@;
OUTPUT;
END;
DATALINES;
    13 27 40 17 36
    19 31 44 28 32
    39 36 41 41 34
    38 29 37 45 29
    22 45 36 15 25
    25 32 38 13 31
    10 44 35 20 30
RUN;

```

[예2] 라틴정방설계(latin square design)

정유회사에서 4 종류의 휘발유 A, B, C, D 에 대한 자동차의 연비(리터당 주행 가능 Km)를 비교하려 한다. 그러나 연비는 실험에 동원된 운전 기사와 자동차 모델에 따라 차이가 날 수 있으므로 이를 배제하기 위하여 운전기사와 자동차 모델을 블록변수(block variable)로 잡아 라틴정방실험을 설계하였고 수집된 데이터는 다음과 같다.

| 연비 데이터 | | 자동차 모델 | | | | | | | |
|--------|---|--------|------|---|------|---|------|---|------|
| | | 1 | | 2 | | 3 | | 4 | |
| 운전기사 | 1 | D | 15.5 | B | 33.9 | C | 13.2 | A | 29.1 |
| | 2 | B | 16.3 | C | 26.6 | A | 19.4 | D | 22.8 |
| | 3 | C | 10.8 | A | 31.1 | D | 17.1 | B | 30.3 |
| | 4 | A | 14.7 | D | 34.0 | B | 19.7 | C | 21.6 |

```

DATA gasoline; /* 두 개의 DO 그룹 이용 */
  DO driver=1 TO 4; * driver는 운전기사;
    DO car=1 TO 4; * car는 자동차모델;
      INPUT gas $ km @@; * gas는 휘발유 종류;
      OUTPUT; * km은 연비;
    END;
  END;
END;
DATALINES;
  D 15.5 B 33.9 C 13.2 A 29.1
  B 16.3 C 26.6 A 19.4 D 22.8
  C 10.8 A 31.3 D 17.1 B 30.3
  A 14.7 D 34.0 B 19.7 C 21.6
RUN;

```

[예3] 임의화블록설계(randomized block design)

한 품종의 콩에 대한 3 가지 살충제의 효과를 비교 분석하여 한다. 토질에 따라 차이가 있을 수 있으므로 동일한 크기의 서로 다른 토질의 4 종류의 땅을 선택한 후 각각을 3등분하여 100 알 씩의 콩을 심었고 그 후 살충제를 뿌리고 **싹이 나온 콩의 개수**를 세었다. 기타 다른 조건은 모두 같았다. 여기서 토질이 다른 네 종류의 땅이 블록에 해당한다.

| 살충제 | 블록(토질) | | | |
|-----|--------|----|----|----|
| | 1 | 2 | 3 | 4 |
| 1 | 56 | 49 | 65 | 60 |
| 2 | 84 | 78 | 94 | 93 |
| 3 | 80 | 72 | 83 | 85 |

```
DATA soybean; /* 두 개의 DO 그룹 이용 */
  DO insecticide=1 TO 3;          * insecticide는 살충제;
    DO block=1 TO 4;              * block은 토질;
      INPUT seedings @@;          * seedings는 싹 튼 콩 개수;
      OUTPUT;
    END;
  END;
DATALINES;
  56 49 65 60
  84 78 94 93
  80 72 83 85
RUN;
```

[예4] 회귀분석(regression analysis)

광고비가 매출액에 끼치는 경향을 분석하기 위하여 로그를 위한 매출액을 광고비의 함수로 놓고 회귀분석을 하려한다. 모두 10 개 회사에 관한 자료가 있다.

| 회사 | 매출액 | 광고비 |
|----|------|-----|
| 1 | 2.5 | 1.0 |
| 2 | 2.6 | 1.6 |
| 3 | 2.7 | 2.5 |
| 4 | 5.0 | 3.0 |
| 5 | 5.3 | 4.0 |
| 6 | 9.1 | 4.6 |
| 7 | 14.8 | 5.0 |
| 8 | 17.5 | 5.7 |
| 9 | 23.0 | 6.0 |
| 10 | 28.0 | 7.0 |

```

DATA linear;
  INPUT sales ad;      * sales는 매출액, ad는 광고비;
  logsales=LOG(sales);
  DATALINES;
    2.5 1.0
    2.6 1.6
    2.7 2.5
    5.0 3.0
    5.3 4.0
    9.1 4.6
    14.8 5.0
    17.5 5.7
    23.0 6.0
    28.0 7.0
  RUN;

```

[예5] 회귀분석(regression analysis) - 반복측정

보관 기일에 따라 어떤 제품에 포함된 염소(chlorine)의 잔류량 퍼센티지를 측정한 자료다. 이 자료에 회귀모형을 적합하여 경과 시간에 대한 잔류 염소량의 감소 추세를 설명하려 한다. 자료에서 **경과 시간에 대한 잔류 염소량의 측정 회수가 일정치 않음**에 유의하라.

| 경과시간(단위: 주) | 염소 잔류량(단위: %) | | | |
|-------------|---------------|------|------|------|
| 8 | 0.49 | 0.49 | | |
| 12 | 0.46 | 0.46 | 0.45 | 0.43 |
| 16 | 0.44 | 0.43 | 0.43 | |
| 20 | 0.42 | 0.42 | 0.43 | |
| 24 | 0.42 | 0.40 | 0.40 | |
| 28 | 0.41 | 0.40 | | |
| 32 | 0.41 | 0.40 | | |
| 36 | 0.41 | 0.38 | | |
| 40 | 0.39 | | | |

```

DATA chlorine; /* DO UNTIL 그룹 이용 */
  INFILE DATALINES MISOVER;
  INPUT elapsed chlorpct @; /* elapsed=경과시간, chlorpct=염소잔류량 */
  OUTPUT;
  DO UNTIL(chlorpct=. ); ** <==> DO WHILE(chlorpct ^= . );
    INPUT chlorpct @;
    IF chlorpct=. THEN RETURN;
  OUTPUT;

```

END;

DATALINES;

| | | | | |
|----|------|------|------|------|
| 8 | 0.49 | 0.49 | | |
| 12 | 0.46 | 0.46 | 0.45 | 0.43 |
| 16 | 0.44 | 0.43 | 0.43 | |
| 20 | 0.42 | 0.42 | 0.43 | |
| 24 | 0.42 | 0.40 | 0.40 | |
| 28 | 0.41 | 0.40 | | |
| 32 | 0.41 | 0.40 | | |
| 36 | 0.41 | 0.38 | | |
| 40 | 0.39 | | | |

RUN;

5.8 큰 자료의 분석에 대하여 <<----- Skip

설문지 조사를 포함하여 여론조사 등에서 얻어지는 자료는 상당히 크다. 예를 들어, 임의표집한 1,000명에 대한 50 문항을 질문하는 설문지 자료가 있다고 가정해보자. 이같이 상당히 큰 자료를 SAS로 분석할 때 매번 원시자료를 읽고 분석하는 것은 극히 비효율적인 자료처리 방법이다. 이런 경우에는 일단 수집된 자료를 영구 SAS 자료로 만들어놓고 분석프로그램은 따로 실행시켜야 한다.

<큰 자료분석의 처리 단계>

- ① 문서편집기를 사용하여 원시자료를 외부 데이터 파일로 만든다.
- ② DATA 단계에서 외부 파일을 읽어 영구 SAS 자료로 출력한다.
- ③ 분석 프로그램을 작성하여 실행한다.

단계 ① 에서 만든 데이터 파일의 예 (quest1.dat)

```
101 501411304211104231000000000333423223 100000000000000100333
767222312276326112222221322112222217613222113222511122222
332111112210460521581100700423, 413341501010023434444123
102 712561502214102311000000000313231113 1000000000000100000755
77722276227532724222222157766222221173222223222522131222
11221111110410441561200700721, 014441712110032344454155
:
454 57237140221310323100000000032253132 0000000000100000100544
757222542274437142222222557333222221171222244422245112222
11321211111055 1451200802326, 011121562010021111211122
501 492213403232100321000000000322133313 000000000000000100222
7342224122711131122222211151222222113222211322211131222
1111 1112210510601571000600823, 513331491010033322242222
```

단계 ② 작업에서 영구 SAS 자료 만들기 (quest1)

```
LIBNAME MyFolder ".": /* 현재 폴더를 MyFolder 라는 라이브러리 이름으로 지정 */
DATA MyFolder.quest1: /* LIBNAME (교재 7.7절 참고) */
  INFILE "quest1.dat";
  INPUT id 3,
         @6 age 2, sex 1,
         (a1-a6) (1.) a7 2, (a8-a11) (1.) a12 2,
         (a13-a15) (1.) (aq1-aq3) (3.) (b1-b16) (1.)
         (bq1-bq14) (1.)
         #2 @6 (c1-c30) (1.) (cs1-cs30) (1.)
         #3 @7 (d1-d14) (1.) (ds1-ds7) (1.)
         (e1-e5) (3.) eq1 2, eq2 4,
         (e6-e16) (1.) (f1-f10) (1.);
RUN;
```

```
LIBNAME MyFolder ". ";
DATA MyFolder.mod11;
    SET quest1; /* SET 문장을 통해 저장된 영구 자료 불러서 사용 */
    ;
RUN;
```

단계 ③ 분석 단계 (예)

```
LIBNAME MyFolder ". "; /* LIBNAME 문장을 먼저 지정하며 영구 자료의 위치를 명시 */

PROC CORR DATA=MyFolder.quest1 NOSIMPLE;
    VAR a1-a6;
RUN;

PROC FREQ DATA=MyFolder.quest1;
    TABLES b1*b2;
RUN;

PROC UNIVARIATE DATA=MyFolder.quest1 PLOT;
    VAR c1-c10 eq1 eq2;
RUN;

PROC PLOT DATA=MyFolder.quest1;
    PLOT a7*a12 / BOX;
RUN;
```