(3) READING DATA

If you are a SAS user, it is easy to think of data as being in either of two formats: a SAS data set; not a SAS data set. As has been discussed in chapter two, a SAS data set is data in a proprietary format for use with SAS software. Most data you encounter will not be found in SAS data sets. Converting non-SAS data (referred to from now on as *raw data*) into a SAS data set is a fundamental task that is usually accomplished with a data step. SAS can handle raw data in just about any format. SAS is instructed as to how read raw data with an INPUT statement, and there are three types of raw data input: LIST, COLUMN, and FORMATTED. SAS is instructed as to where to find raw data with an INFILE statement.

Though raw data can be found in a variety of formats, one way to think about such data is that it is either organized or unorganized. Organized data are defined as data where the values of variables occur in the same location (columns) in every record of the raw data file. In a file of unorganized data, the values of variables need not be in the same location in every record. The following shows two small data file, each with 4 records and 3 variables: city, population in 2000, population in 1990. The raw data on the left are organized, unorganized on the right.

ORGANIZED	UNORGANIZED
COLUMN NUMBERS 123456789012345678901234 5	COLUMN NUMBERS 12345678901234567890123456 7
NEW YORK 80082787322564 LOS ANGELES36948203485398 BOSTON 589141 574283 ALBANY 95658 101082	NEW YORK,8003278,7322564 LOS ANGELES,3694820,3485398 BOSTON,589141,574283 ALBANY,95658,101082

In the organized data, the city name is always in columns 1 through 11, the 2000 population is always in columns 12 through 18, and the 1990 population is in columns 19 through 25. In the unorganized data, the column location of both the city and populations vary from record to record. Also notice that within the unorganized data, the values of the variables are separated by a comma, making it easy to determine where the value of one variable ends and another begins.

Up to this point, the location of all raw data used in data steps has been in files. Within the data steps, INFILE statements are used to tell SAS where to find the raw data file. For small amounts of data, it is also possible to place raw data within a data step in a location known as a DATALINES (or CARDS) file. This is sometimes referred to as instream data. Several of the examples in this chapter use raw data in a DATALINES file.

...LIST INPUT

LIST input is the only way to read data in which the values of variables do not always occur in the same location across observations, unorganized data. LIST input depends on there being a delimiter separating the values of all the variables within each observation. In the unorganized data shown above, that delimiter is a comma. There are many features associated with LIST input that allow you to read unorganized data. Having to learn the features of LIST input is the price you pay for having, for the want of a better term, sloppy data.

Delimiters (DSD and DLM Options)

The default delimiter for LIST input is a space, not a comma. If the delimiter is something other than a space, you must include information about the delimiter in an INFILE statement whether your data are located in an external file or in a DATALINES file. Some common delimiters are: space, comma, tab, slash (/).

...Example 3.1...

```
*LIST input with default space delimiter using instream data;
data presidents;
input ①
lastname  : $12.
chol
sbp
dbp
dob       : mmddyy. ②
dod       : mmddyy.
;

aod = (dod - dob) / 365.25; ③
label
chol = 'TOTAL CHOLESTEROL'
sbp  = 'SYSTOLIC BLOOD PRESSURE'
dbp  = 'DIASTOLIC BLOOD PRESSURE'
dob  = 'DATE OF BIRTH'
dod  = 'DATE OF DEATH'
aod  = 'AGE AT DEATH'
;
format dob dod date9. aod 4.1; ④
datalines; ⑤
ADAMS 230 160 90 04111767 02231848
FILLMORE 300 156 88 01071800 03081874
LINCOLN 222 144 82 02121809 04151865
JOHNSON 190 . 60 12291808 07311875
TRUMAN 150 140 70 05081884 12261972
WASHINGTON 110 150 75 02221732 12141799
CLINTON  200 140 085 08191946 .
;
run;
```

A new, temporary data set named PRESIDENTS is created from raw data that is part of the SAS job in a DATALINES file. The INPUT statement ① is written using suggestions made on page bottom of page 12 for reading delimited data:

a/ choose names for the variables in your raw data and write those names in a column
b/ to the right of each character variable, write a colon followed by the maximum number of spaces needed to accommodate the value of the variable, add a period after the number, add a $ prior to the number of columns
c/ nothing is written to the right of the numeric variables
d/ the description you write is preceded by the keyword INPUT and ends with a semi-colon

There is something new in this INPUT statement, date informats ②. Thus far, we have metioned character and numeric variables. Dates are a special type of numeric variable. In the DATALINES file, the 5th and 6th values on each line are dates written as a 2-digit month and day followed by a 4-digit year. You could read these values as character variables using $8. in place of MMDDYY. However, you would not be able to calculate an age at death ③ since you cannot use character variables in mathematical calculations. You could also read these values as numbers, allowing you to use them in calculations. That would make a calculation of age at death for the ADAMS produce the following result (with commas added to the larger numbers to make them easier to read) ...

```
aod = (dod - dob) / 365.25 = (2,231,848 - 4,111,767) / 365.25 = -5,146.93771
```

That result is obviously not what you wanted when you calculated an age at death. You can see that ADAMS was born in 1767 and died in 1848 so depending on the exact date in each year, he should be approximately 81 when he died. How can you get that answer from the given data? The easiest way to do that is to read the raw data values in a special way using a date informat. The text in INPUT statements that appears to the right of the variable names (also in examples 1.3, 1.4, 2.4) is referred to as an informat, or an instruction as to how to read raw data and convert it into the value of a variable in a data set. The informat MMDDYY. is an instruction to take the value of the raw data and convert it into the number of days since January 1, 1960. There are many different types of date informats and they are discussed in a later chapter. For now, all you have to remember is that dates are a special type of numeric data and that the values as raw data are converted to dates in in a data set using date informats. If you use the date informats as shown in the example, the calculation of age at death becomes ...

```
aod = (dod - dob) / 365.25 = (-40,854 - -70,391) / 365.25 = 29,357 / 365.25 = 80.8679
```

The numbers -40,854 and -70,391 are the number of days since January 1, 1960 for the date of death and date of birth respectively. A format is added to the age at death that will result in the value of the variable being displayed rounded to one decimal place ④. There is also a format to control the appearance of the two dates, DATE9. Since the actual values of the dates in the data set are those shown in the above equation, you need a format to convert those values into a form that you (and others) can read as dates. Just as there are many date informats (rules to read raw data values as dates), there are many date formats (rules to write the values of SAS variables as dates). More will said about date formats in a later chapter.

The raw data read by the INPUT statement are included in the data step as a DATALINES file ⑤. This is an easy method for using small amounts of raw data rather than placing them in a separate file. All the lines between the keyword DATALINES and the semi-colon after the data for CLINTON are considered as raw data to be read in the data step. Notice that there is no INFILE statement. If SAS encounters an INPUT statement in a data step without having already seen an INFILE statement, SAS will look for your data in a DATALINES file. If you have an INPUT statement without a previous INFILE statement or no subsequent DATALINES file, you will get an error message.

You should also notice some features of the raw data. First, values of all the variables are separated by a space. This allows the INPUT statement to know when the value of one variable ends and the next begins. Second, look in the JOHNSON and CLINTON records. In the JOHNSON record, the third entry on the line is a period indicating that the raw data value for SBP, the third variable in the INPUT statement, is missing. The same is true for the last entry in the CLINTON record where there is no value for DOD (no date of death since ex-president Clinton is still alive). These periods are VERY IMPORTANT when using list input and space-delimited raw data and we will look at what happens in list input if we leave out the periods in the JOHNSON and CLINTON records.

Before taking out those periods, PROC CONTENTS is used to show the attributes of the variables in the data set MYSTUFF (shown on the right). The two dates are numeric variables and will be displayed using a date format. The age at death will be displayed rounded to one decimal place. Since we are allowing only 4 places to display the age at death, we are assuming that no president is 100+ years old when they die. The results of PROC PRINT used with the label option are shown on the top of the next page.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
7	aod	Num	8	4.1
2	chol	Num	8	
4	dbp	Num	8	
5	dob	Num	8	DATE9.
6	dod	Num	8	DATE9.
1	lastname	Char	12	
3	sbp	Num	8	

Obs	LASTNAME	TOTAL CHOLESTEROL	SYSTOLIC BLOOD PRESSURE	DIASTOLIC BLOOD PRESSURE	DATE OF BIRTH	DATE OF DEATH	AGE AT DEATH
1	ADAMS	230	160	90	11APR1767	23FEB1848	80.9
2	FILLMORE	300	156	88	07JAN1800	08MAR1874	74.2
3	LINCOLN	222	144	82	12FEB1809	15APR1865	56.2
4	JOHNSON	190	.	60	29DEC1808	31JUL1875	66.6
5	TRUMAN	150	140	70	08MAY1884	26DEC1972	88.6
6	WASHINGTON	110	150	75	22FEB1732	14DEC1799	67.8
7	CLINTON	200	140	85	19AUG1946	.	.

What happens if the periods indicating missing data are left out of the DATALINES file?

...Example 3.2...

```
data presidents;
input
lastname  : $12.
chol
sbp
dbp
dob       : mmddyy.
dod       : mmddyy.
;

aod = (dod - dob) / 365.25;
format dob dod date9. aod 4.1;
datalines;
ADAMS 230 160 90 04111767 02231848
FILLMORE 300 156 88 01071800 03081874
LINCOLN 222 144 82 02121809 04151865
JOHNSON 190 60 12291808 07311875 ①
TRUMAN 150 140 70 05081884 12261972
WASHINGTON 110 150 75 02221732 12141799
CLINTON  200 140 085 08191946 ②
;
run;
```

Before using PROC PRINT to display the data set and looking at the LOG file, look at the records for JOHNSON ① and CLINTON ②. For JOHNSON, there are values for only 5 variables but the INPUT statement instructs SAS to look for values of 6 variables in each record of the raw data file. The same is true for CLINTON, but now the missing value is at the end of the record rather than in the middle. Here are the results of PROC PRINT.

Obs	LASTNAME	CHOL	SBP	DBP	DOB	DOD	AOD
1	ADAMS	230	160	90	11APR1767	23FEB1848	80.9
2	FILLMORE	300	156	88	07JAN1800	08MAR1874	74.2
3	LINCOLN	222	144	82	12FEB1809	15APR1865	56.2
4	JOHNSON	190	60	12291808	31JUL1875	.	.
5	WASHINGTON	110	150	75	22FEB1732	14DEC1799	67.8

The first three observations in the data set match those of the previous example. Once we encounter the data for JOHNSON, there is a problem. The thirds value in the record for JOHNSON should be a missing

value for SBP. Since the INPUT statement has no way of knowing that the value is missing, the following sequence of events occurs.

a/ the value 60 is read as the value for systolic blood pressure (SBP) though it really is the value for diastolic blood pressure (DBP)
b/ the value 12291808 that should be the date of birth (DOB) is read as DBP
c/ the value 07311875 that should be the date of death (DOD) is read as DOB
d/ the INPUT statement still needs a value for DOD and there are no values left in the JOHNSON record
e/ SAS goes to the next record in the DATALINES file to find a value for DOD and reads the first entry in that record and that is the text TRUMAN
f/ since TRUMAN is text and DOD is a numeric variable, the value of DOD is missing
g/ the next use of the INPUT statement starts the record WASHINGTON record since the default behavior of the INPUT statement is to always start with the next record in the raw data file
h/ the values of all the variables for WASHINGTON are correct
i/ the next use of the INPUT statement tries to read the values of six variables in the CLINTON record, but there are only five values and no subsequent records in the raw data
j/ since SAS cannot find a value for each variable, ex-president CLINTON is left out of the data set

The three important things to remember from the above sequence are: every time the INPUT statement is used in a data step, SAS tries to read the next record in a raw data file; with list input, SAS must read a value for each variable listed in the INPUT statement and SAS will jump to the next record in the raw data if necessary to find those values; if a value is not found for each variable in an INPUT statement, that record is not added to the data set. The LOG file gives you information about some of the events listed in the sequence shown above.

```
NOTE: Invalid data for DOD in line 285 1-6.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+----0
285        TRUMAN 150 140 70 05081884 12261972
LASTNAME=JOHNSON CHOL=190 SBP=60 DBP=12291808 DOB=31JUL1875 DOD=. AOD=. _ERROR_=1 _N_=4
NOTE: LOST CARD.
288        ;
LASTNAME=CLINTON CHOL=200 SBP=140 DBP=85 DOB=19AUG1946 DOD=. AOD=. _ERROR_=1 _N_=6
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: Missing values were generated as a result of performing an operation on missing values.
      Each place is given by: (Number of times) at (Line):(Column).
      1 at 277:12
NOTE: The data set WORK.PRESIDENTS has 5 observations and 7 variables.
```

The note about invalid data for DOD refers to item 'f' in the sequence, using TRUMAN as a value for DOD. The LOST CARD note refers to items 'i' and 'j' in that there are not enough variable values in the raw data to complete INPUT statement for CLINTON, so that data is lost. The next note refers to items 'd' and 'e' in that the INPUT statement could not find values for all the variables in the JOHNSON record, so it went to a new line (TRUMAN) to find more data. The conclusion you should draw from this example is to remember to include periods for missing values when using list input and space-delimited raw data.

A more common delimiter is the comma. One reason for this is that you use Excel for raw data entry into a file and then save your data as comma-separated value (.CSV) file. Using a comma delimiter is an easy way to avoid the problems encountered in example 3.2 when some variables have missing values. You need not enter a period in the raw data for the missing value since SAS will interpret two consecutive commas in the middle of a record or a single comma at the end of a record as indicating missing data. This can be seen in the next example.

...Example 3.3...

```
data presidents;
infile datalines dsd;  ①
input
lastname  : $12.
chol
sbp
dbp
dob       : mmddyy.
dod       : mmddyy.
;

aod = (dod - dob) / 365.25;
format dob dod weekdate. aod 4.1;  ②
datalines;
ADAMS,230,160,90,04111767,02231848
FILLMORE,300,156,88,01071800,03081874
LINCOLN,222,144,82,02121809,04151865
JOHNSON,190,,60,12291808,07311875          ③
TRUMAN,150,140,70,05081884,12261972
WASHINGTON,110,150,75,02221732,12141799
CLINTON,200,140,085,08191946,              ④
;
run;
```

The default delimiter for list input is a space. Since the delimiter in the raw data is a comma, an INFILE statement is used with a DSD to tell SAS that the raw data file is comma-delimited. This was also done in example 1.4 when the INFILE statement pointed to an external file. In this example, the data are included in the data step and are in a DATALINES file. That is why the INFILE statement looks as it does. Just to show you another way of displaying dates, a new date format is used, weekdate ②. In the DATALINES file, you can see the comma delimiters. In the JOHNSON record, the two consecutive commas indicate missing data for SBP ③. In the CLINTON record, the comma at the end of the record indicates missing data for DOD ④ (without that comma, there would be no data for ex-president Clinton in the data set). Using PROC PRINT to display the data set produces the following. Notice the values of the dates produced by the weekdate format.

Obs	LASTNAME	CHOL	SBP	DBP	DOB	DOD	AOD
1	ADAMS	230	160	90	Saturday, April 11, 1767	Wednesday, February 23, 1848	80.9
2	FILLMORE	300	156	88	Tuesday, January 7, 1800	Sunday, March 8, 1874	74.2
3	LINCOLN	222	144	82	Sunday, February 12, 1809	Saturday, April 15, 1865	56.2
4	JOHNSON	190	.	60	Thursday, December 29, 1808	Saturday, July 31, 1875	66.6
5	TRUMAN	150	140	70	Thursday, May 8, 1884	Tuesday, December 26, 1972	88.6
6	WASHINGTON	110	150	75	Friday, February 22, 1732	Saturday, December 14, 1799	67.8
7	CLINTON	200	140	85	Monday, August 19, 1946	.	.

Some other common delimiters are the slash and the tab (if you use Excel to enter data, you have the option of saving your data as tab-delimited). If a delimiter other than a space or comma is used, you must use a DLM option in addition to the DSD option. The following statements show two alternate delimiters: a slash (/); a tab (expressed as a hexadecimal value 09, yes that is a bit 'geeky').

```
*tell SAS that a SLASH is used as the delimiter;
infile datalines dsd dlm="/";

*tell SAS that a TAB is used as the delimiter;
infile datalines dsd dlm="09"x;
```

Informat Statement
If you are using list input, rather than placing the informats after the names of variables in the INPUT statement, you also have the option of placing them in an INFORMAT statement. Example 3.3 could have been rewritten as follows.

...Example 3.4...
```
data presidents;
infile datalines dsd;

informat lastname $12. dob dod mmddyy.;  ①
input lastname chol sbp dbp dob dod;  ②

aod = (dod - dob) / 365.25;
format dob dod weekdate. aod 4.1;
datalines;
ADAMS,230,160,90,04111767,02231848
FILLMORE,300,156,88,01071800,03081874
LINCOLN,222,144,82,02121809,04151865
JOHNSON,190,,60,12291808,07311875
TRUMAN,150,140,70,05081884,12261972
WASHINGTON,110,150,75,02221732,12141799
CLINTON,200,140,085,08191946,
;
run;
```

An INFORMAT statement is used to associate informats with variables ①. The informats are instructions SAS will use in an INPUT statement in the same data step. The INPUT statement only has variable names, no informats ②. It might be a bit confusing in that you now a bit about ...

informat: a rule (instruction) for reading data, normally found in an INPUT statement
INFORMAT: a statement used in a data step that associates one or more informats with variables

The same name is used for both a concept (a rule for reading values of variables) and a statement used in a data step. If you want to avoid confusion, do not bother with INFORMAT statements. You never have to use one if you put the informats in an INPUT statement, not in an INFORMAT statement. Hopefully that is not too confusing. If it is, just pretend you never saw example 3.4.

What you should remember...
If you are using raw data to create a SAS data set and that raw data is unorganized (see page 26), you will have to use list input. List input depends on there being a delimiter between the values of variables in your data so an INPUT statement can tell where the value of one variable ends and the next begins. The default delimiter for list input is a space. You must use the DSD option on an INFILE statement to tell SAS your data contains a comma delimiter rather than a space. You can use delimiters other than a space or comma, but that requires you to add the DLM option to an INFILE statement to specify the delimiter. Rather than reading raw data from an external file, you can place raw data within a data step in a DATALINES file. Informats are used in an INPUT statement as instructions for SAS as to how to read values of variables from raw data. With list input, informats are needed for character variables and for numeric variables to be stored as dates (though, you might have guessed, you will learn more about exceptions to this in a later section).

...FORMATTED INPUT

When data are organized and not delimited, there are two alternatives to list input that are used to read raw data with an INPUT statement. The first of these is formatted input. Many of the examples with INPUT statements thus far have used formatted input in which a variable name is followed by an informat (remember, an informat is a rule or instruction for reading raw data).

...Example 3.5...

```
*FORMATTED input instream data;
data mystuff;
input
@01 lastname  $12.  ①
@13 chol        3.
@16 sbp         3.
@19 dbp         3.
@22 dob   mmddyy8.
@30 dod   mmddyy8.
;

aod = (dod - dob) / 365.25;
label
chol = 'TOTAL CHOLESTEROL'
sbp  = 'SYSTOLIC BLOOD PRESSURE'
dbp  = 'DIASTOLIC BLOOD PRESSURE'
dob  = 'DATE OF BIRTH'
dod  = 'DATE OF DEATH'
aod  = 'AGE AT DEATH'
;
format dob dod date9. aod 4.1;
datalines;
ADAMS       23016090 0411176702231848
FILLMORE    300156 880107180003081874
LINCOLN     222144 8202121809041518657
JOHNSON     190    601229180807311875
TRUMAN      150140 700508188412261972
WASHINGTON  11015075 0222173212141799
CLINTON     20014008508191946
;
run;
```

An INPUT statement is used to read values of variables from raw data in a DATALINES file ①. The location of the value of each variable is specified with a column pointer (an @ followed by a column number) that precedes variable names. Each variable name is followed by an informat. The informat converts raw data to the value of a variable with a type (character or numeric) determined by the informat. The informat also determines how many columns are read for the value of each variable. As stated earlier on page 10, a suggested process for writing a formatted INPUT statement is as follows. Notice that there are no periods in the raw data to indicate missing data. When you read values of data with formatted input, blanks are interpreted as missing data for both character and numeric variables. You do not have to use periods to indicate missing data, the blank spaces are enough.

a/ choose names for the variables in your raw data and write those names in a column
b/ to the left of the each variable name, write an @ followed by the first column in the raw data file where data for that variable is located
c/ to the right of the name of each character variable, write the number of columns occupied by the data for that variable, add a period after the number, add a $ prior to the number of columns
d/ to the right of each numeric variable, do the same thing you would do for a character variable, but leave out the $

A slight modification to this process is made in this example.

e/ if any of the numeric variables are dates, add a date informat to the prior to the number of columns added in part d

Formatted input is defined not by the use of column pointers (@-signs) for locations of your variable values, but by the informats that follow the variable names. You could leave out some or all of the column pointers and still read your data with formatted input.

...Example 3.6...

```
*FORMATTED input instream data - no column pointers;
data mystuff;
input
lastname  $12.  ①
chol        3.
sbp         3.
dbp         3.
dob   mmddyy8.
dod   mmddyy8.
;

aod = (dod - dob) / 365.25;
format dob dod date9. aod 4.1;

datalines;
ADAMS       23016090 0411176702231848
FILLMORE    300156 880107180003081874
LINCOLN     222144 820212180904151865
JOHNSON     190    601229180807311875
TRUMAN      150140 700508188412261972
WASHINGTON  11015075 0222173212141799
CLINTON     20014008508191946
;
run;
```

The INPUT statement in this example is similar to that used in example 3.6, however the column pointers are not used ①. The values of the variables created in this example will be identical to those in example 3.6. Understanding why this INPUT statement works correctly requires a little more knowledge about how an INPUT statement works. When a SAS encounters an INPUT statement within a data step, one record (one line of your data) is moved into memory. The remaining portion of the INPUT statement instructs SAS as to how to find values of variables within that record. The first record in the DATALINES file show above is 37 characters long. When that record is moved into memory, it looks as shown in the last row of the table below. The first instruction in the INPUT statement is an informat ($12.) instructing SAS to read 12 columns of data and store that information in the variable LASTNAME. Without a column pointer for the first variable, the default starting point is column 1 of the record the INPUT statement has placed in memory. After reading the first value, the column pointer is now at column 13 (start at 1, read 12 columns, stop at 13). Since there is no column pointer for the second variable and the informat (3.) tells SAS to read 3 columns, the value of the variable CHOL is read from columns 13 through 15.

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	26	27	28	29	30	31	32	33	34	35	36	37
↓												↓			↓			↓			↓								↓									
A	D	A	M	S								2	3	0	1	6	0	9	0		0	4	1	1	1	7	6	7	0	2	2	3	1	8	4	8

The column pointer is now at column 16 and once again since there is no column pointer in the INPUT statement for the third variable, the value of SBP is read from columns 16 through 18 since the informat (3.) instructs SAS to read 3 columns of the record. This process of moving the column pointer and reading values of variables continues until the value of the last variable (DOD) in the INPUT statement is read from columns 30 through 37. The informat MMDDYY8. tells SAS not only to read the value of the variable from 8 columns, but also to convert the value found in the raw data to a numeric variable whose value is a number of days since January 1, 1960. After the INPUT statement is completed, the remaining statements in the data step are executed and an observation is added to the data set. This process continues until all the records (lines of data) are read from the DATALINES file. The process is the same when the raw data are located in an external file.

One advantage of using column pointers is that you need not read all the raw data in each record when creating a data set. Looking back at example 3.5, if you wanted to create data set with only the values for cholesterol and the two dates, you could write the INPUT statement as follows ...

```
input
@13 chol        3.
@22 dob   mmddyy8.
@30 dod   mmddyy8.
;
```

The INPUT statement would still move an entire record into memory, but it would then go directly to column 13 to find a value for the variable CHOL in columns 13 through 15. Next it would move to column 22 and then 30 to read values for the two dates, DOB and DOD. Skipping parts of your data is not possible without column pointers. A good habit to develop is to ***always*** use column pointers in formatted input.

There are two more features of using column pointers: they need not be in numeric order; they can be used to read the same raw data multiple times. Once again looking back at example 3.5, you could write the INPUT statement as follows ...

```
input
@22 dates     $16.
@22 dob   mmddyy8.
@30 dod   mmddyy8.
@01 name      $12.
;
```

Numeric variables read as dates can be assigned missing values for several reasons. One is that the value was missing. Another is that a portion of the date is missing, for example there is a month and year present but no day. Yet another is that the value in the raw data is an invalid date, for example June 31 or February 30. If you used the INPUT statement shown above with the character variable DATES, you could see why either DOB or DOD was assigned a missing value if you printed your data set ...

```
proc print data=mystuff;
where dob is missing or dod is missing;
run;
```

Why you might want to read the values of you variables not in the order that they appear in you raw data is up to you. One reason might be is that the order of the variables in a data set is determined by when SAS encounters the names of variables in a data step. If for some reason you wanted the dates to appear first in the data set (prior to NAME), you could use the INPUT statement with NAME occurring last.

What you should remember...
If you are using raw data to create a SAS data set and that raw data is organized (see page 26), you can use formatted input. Formatted input uses informats after variable names to both determine the type of variable being created from raw data and to also specify how many columns of data to read for the value of a variable. Column pointers (@-signs) can be used to specify where the value of a variable begins. Without a column pointer for the first variable in an INPUT statement, SAS will start to look for the value of that variable in column 1 of the raw data. You can use column pointers to skip portions of your raw data.

The material about reading the same raw data more than once or reading values out of order is useful, but not as important as the other material in this section.

...COLUMN INPUT

An alternative to formatted input for reading organized, non-delimited data is column input. Column input does not use informats or column pointers. Rather, it use a start and end column in an INPUT statement to tell SAS where the values of variables in your raw data begin and end.

...Example 3.7...

```
*COLUMN input;
data mystuff;
input
name $ 1-17   ①
chol   18-20  ②
sbp    21-23
dbp    24-25
;
datalines; ③
JOHN QUINCY ADAMS23016090
MILLARD FILLMORE 30015688
ABRAHAM LINCOLN  22214482
ANDREW JOHNSON   190   60
HARRY TRUMAN     15014070
GEORGE WASHINGTON11015075
BILL CLINTON     20014085
;
run;
```

There are no column pointers (@-signs) in the INPUT statement. For the character variable NAME, there is a $ followed by a column range telling SAS to find the value for that variable in columns 1 to 17 of the raw data ①. The next variable, CHOL, is numeric and there is no $, just another column range ②. In the DATALINES file, notice that the variables DOB and DOD have been left out of this example ③. ***There is no way to read a variable as a date using column input.*** Only LIST input with an informat or FORMATTED input can be used to read dates. That is all there is to column input ...

a/ choose names for the variables in your raw data and write those names in a column
b/ to the right of the name of each variable, write the range columns occupied by the data for that variable
c/ for character variables, add a $ between the variable name and the column range

What you should remember...
If you are using raw data to create a SAS data set and that raw data is organized (see page 26), you can use column. Column input uses column ranges to tell SAS where to find the values of variables in your raw data. All variables will be read as numeric unless you add a $ to the INPUT statement following the name of any character variable(s). ***There is no way to read a variable as a date using column input.*** Finally, if you do not want to remember column input, just use formatted input for organized data.

...INFORMATS

There are many SAS-supplied informats that allow you to read data that are stored in different ways. You can also write your own informats to read data (more about that later when we look at PROC FORMAT). In example 3.6, we are reading two different types of data, character and numeric. The informat $12. tells SAS that there are twelve columns of data to be stored as a character variable. The 3. informat tell SAS that you are reading data that are to be stored as numeric variables and that they are located in three columns. The MMDDYY8. used to read the dates is another informat.

...Character Data...

There are a few SAS-supplied informats that you will use repeatedly. The two informats commonly used to read characters data are $<w.> and $char<w.>, where the <w.> indicates a width that you provide (a minimum of 1 and a maximum, for now, of 32,000). In most instances, reading character data with either of these two formats has the same results. However, there are two instances that you should know about where they differ. If the value of a character variable has leading blanks, the $<w.> informat will get rid of them and left justify the data, while the $char<w.> will not modify the data, leaving the leading blanks. The other instance where these two formats differ is when you use a period to indicate missing character data. The $<w.> informat will read the period as indicating a missing value, while the $char<w.> informat will read the period as representing real data value.

...Example 3.8...

```
* difference between $ and $char informat;
data characters;
input  ①
@1 a $char10.
@1 b $10.
;
label
a = 'INFORMAT $char10.'
b = 'INFORMAT $10.'
;
datalines;  ②
MIKE ZDEB
.
     ZDEB
;
run;

title 'CHARACTER INFORMATS';
proc print data=characters label
run;
```

CHARACTER INFORMATS

Obs	INFORMAT $char10.	INFORMAT $10.
1	MIKE ZDEB	MIKE ZDEB
2	.	
3	ZDEB	ZDEB

The same data are read twice, in two different ways (two different informats) ①. The @1 tells SAS to start in column one of the raw data file in reading values of variables for each variable (the same data are read for variables A, B) and the two INFORMATS tell SAS to read ten columns. The DATALINES file shows values that will be treated differently by the two different character informats ②.

Notice in the printed results the variable A has values that look exactly the same as those that occur in the DATALINES section of example 3.8. The values for variable B are different. The period in the second line of the DATALINES file has been converted to a blank, and the value of the third observation has been left justified (leading blanks have been removed). Neither of these character informats is inherently the correct one to use. That depends on how you want your raw data treated as you assign values to variables in your data set.

...Numeric Data...

When reading numeric data, the <w>. format will suffice in most instances. However, if you have numeric data containing anything other than numbers, a decimal point, or a minus sign, you can't read that data with just the <w.> informat. You must use the comma<w.> informat.

...Example 3.9...

```
*COMMA informat;
data numbers;
input  ①
@1 a $char8.
@1 b 8.
@1 c comma8.
;
label
a = 'INFORMAT $char8.'
b = 'INFORMAT 8.'
c = 'INFORMAT comma8.'
;
datalines; ②
1000
 1000
  1000
   1000

1,000
$1000
1000.
-1000
(1000)
10-00
10-0-0
10 00
1  00
;
run;

title 'COMMA INFORMAT';
proc print data=numbers label;
run;
```

NUMERIC INFORMATS

Obs	INFORMAT $char8.	INFORMAT 8.	INFORMAT comma8.
1	1000	1000	1000
2	1000	1000	1000
3	1000	1000	1000
4	1000	1000	1000
5	1,000	.	1000
6	$1000	.	1000
7	1000.	1000	1000
8	-1000	-1000	-1000
9	(1000)	.	-1000
10	10-00	.	1000
11	10-0-0	.	1000
12	10 00	.	1000
13	1 000	.	1000

The same data are read three times, in three different ways (three different informats) ① . The @1 tells SAS to start in column one of the data file in reading data for each variable (the same data are read for variables A, B, and C) and the various INFORMATS tell SAS to read eight columns. The data comprise a combination of different ways of representing the number one-thousand ②.

This comma informat has some "smarts" in that it knows that you want the values being read to be stored as numeric data and will strip away (or convert) anything other than numbers prior to storing the values. Any characters other than numbers, decimal points, or minus signs cause the numeric informat to result in a missing value. What about observations 12 and 13? Is the conversion correct (variable c) or should there be zeroes, not blanks, in the values? Notice that SAS does not care where in the specified column range the number is located: left-justified, right-justified, centered, whatever. SAS will find the number anyplace within the range specified and store the number as it appears in your data.

One last item about numeric informats contrasts actual and implied decimal points in your data. There are occasions when numeric values in your data will contain a decimal point. However, it is not uncommon to have an implied decimal point. One example of this is values a numeric data that represent dollars and cents. A value of 100012 stored as raw data might actually represent $1000.12. If your task is to read that data and calculate a mean value, unless you read the data correctly (insert a decimal point with an informat), your calculated values will be incorrect.

...Example 3.10...

```
data decimals;
input ①
@1 a $char10.
@1 b 10.
@1 c 10.2
@1 d 10.4
;
label
a = 'INFORMAT $char8.'
b = 'INFORMAT 10.'
c = 'INFORMAT 10.2'
d = 'INFORMAT 10.4'
;
datalines; ②
100012
1000.12
1000.123
;
run;

title 'REAL AND IMPLIED DECIMAL POINTS';
proc print data=decimals;
run;

title 'REAL AND IMPLIED DECIMAL POINTS - SHOW ALL THE DECIMAL PLACES';
proc print data=decimals; ③
format b c d 15.5;
run;
```

The same data are read in four different ways (four different informats) ①. The @1 tells SAS to start in column one of the data file in reading data for each variable (the same data are read for variables A, B, C, and D) and the various INFORMATS tell SAS to read ten columns. The data contain a mix of implied and real decimal places ②. A FORMAT is used to display each number with five decimal places ③.

REAL AND IMPLIED DECIMAL POINTS

Obs	INFORMAT $char8.	INFORMAT 10.	INFORMAT 10.2	INFORMAT 10.4
1	100012	100012.00	1000.12	10.00
2	1000.12	1000.12	1000.12	1000.12
3	1000.123	1000.12	1000.12	1000.12

REAL AND IMPLIED DECIMAL POINTS - SHOW ALL THE DECIMAL PLACES

Obs	INFORMAT $char8.	INFORMAT 10.	INFORMAT 10.2	INFORMAT 10.4
1	100012	100012.00000	1000.12000	10.00120
2	1000.12	1000.12000	1000.12000	1000.12000
3	1000.123	1000.12300	1000.12300	1000.12300

Notice that the INFORMAT 10. left the data just as it appeared in the DATALINES file. The 10.2 INFORMAT added two decimal places to the number that had no decimal places (10012), but had no effect on the numbers that already had a real decimal point. The same behavior is true for the 10.4 INFORMAT. If you print that data, you might think that SAS has dropped some of the decimal values. However, PROC PRINT makes some decisions as to how you might want the output to look. You can override the appearance of the numeric data in the output by adding a FORMAT statement as was done in the second PROC PRINT, using the format 15.5 to display the numeric variables.

What you should remember...
Informats are used to convert values of variables in raw data into values of variables in a SAS data set. The character informat $char<w>. creates a character variable and makes no changes in the value, while $<w>. removes leading blanks. The only allowable characters in standard numeric raw data are: numbers, + and - signs, decimal points. If any other characters occur (for example, a comma or dollar sign), you must use a comma<w>. informat to read the value of the variable. Decimal points already in raw data will cannot be moved with an informat. Numeric raw data with an implied decimal point can have a decimal point added with an informat.

...VARIATIONS ON THE INPUT STATEMENT
In the examples up to now, all the raw data to be converted into a SAS data set have been presented in the same manner. There has been one record per observation, and all the data for variables to be read by an input statement have been located with one record. Each record in the raw data files have always contained the same amount of information, i.e. data for each of the variables mentioned in the input statement. There are methods of reading raw data files that do not have these characteristics. There are also some shortcuts to reading raw data that occurs in a standard form, one record per person but with repeated values of similar variables

...Informat Lists...
There are cases when you have data where the same format applies to a number of different variables that happen to fall one-after-another in a record in your raw data. For example, what if you have a patient record with an ID, up to five diagnoses, and a length of stay. You decide to use formatted input to read the data.

...Example 3.11...
```
data hosp;
input ①
@01 pat_id   $9.
@10 dx1      $5.
@15 dx2      $5.
@20 dx3      $5.
@25 dx4      $5.
@30 dx5      $5.
@35 los      4.
;
datalines;
1303476542848 2913                 0010
000231234292822961030301309213150200 35
123456789V31                       0002
;
run;
```

Formatted input is used to read the raw data ①. The @-sign is use to direct SAS to a specific column in the raw data. An informat is specified for each variable, six character variables and one numeric. There is an easier way to read these data, with an informat list.

...Example 3.12..

```
data hosp;
input ①
@01 pat_id     $9.
@10 (dx1-dx5) ($5.) ②
@35 los         4.
;
datalines;
1303476542848 2913                0010
000231234292822961030301309213150200 35
123456789V31                      0002
;
run;
```

Formatted input is used to read the raw data ①. An informat list is used to read the diagnoses ②. Notice that if variables have a common prefix (in this case DX) and a numeric suffix (in this case 1 though 5), you can use a shortcut to refer to all the variables DX1 through DX5, and that is DX1-DX5. The informat $5. applies to each of the five diagnoses DX1 through DX5. You can modify the input statement in example 10 by putting all the variables and all the informats in parentheses.

```
input (pat_id dx1-dx5 los) ($9. 5*$5. 4.);
```

In an input statement, if one or more variables are listed within a set of parentheses, SAS expects another set of parentheses that contain the informats used to read the data. There should be an informat for each variable. In the input statement shown above, the $9. informat is used to read the patient id. The 5*$5. informat tells SAS to repeat using the $5. informat five times, once for each diagnosis. The numeric informat 4. is used to read length of stay.

What if you were only interested in reading the first three characters of each diagnosis (major diagnostic subgroups rather than all the detail conveyed by the full five digits)? No problem since in addition to the absolute column pointer (the @ sign), SAS also has a relative column pointer (the + sign). The relative column pointer tells SAS to move a specified number of columns from its current position on the line of data currently being read.

...Example 3.13..

```
data hosp;
input
@01 pat_id     $9.
@10 (dx1-dx5) ($5. +2)  ①
@35 los         4.
;
datalines;
1303476542848 2913                0010
000231234292822961030301309213150200 35
123456789V31                      0002
;
run;
```

Each diagnosis would be stored with a length of three rather than five. Informat lists are one of many instances of writing SAS code that can be considered a "shortcut", i.e. there are other ways to accomplish the same task with SAS code, but the informat list requires fewer keystrokes. One of the tasks in learning how to use SAS (or any topic) is to determine when the "shortcut" might be complicated enough to make the "more keystroke" solution the preferred path.

...Data for One Observation Stored in More than One Record (Slash and Number Sign)...
If the raw data to be read by an INPUT statement are present on more than one record, the input statement must be instructed to move to a new record to complete reading the data for all the variables mentioned in the INPUT statement. There are two ways of instructing an input statement to move to a new record.

...Example 3.14...
```
data new;
input
id             $ 1-5
gender         $ 6    /  ①
(grade1-grade5) (3.);
datalines;  ②
12345M
100 98 78 99100
45678F
 75 64 76 80100
99999M
100100 99100100
;
run;
```

A slash (/) is used in the input statement to instruct SAS to move to a new line in the raw data file ①. Column input is used to read the first two variables, while an informat list is used to read variables grade1 through grade5. The data intended for each observation are spread across two records in the raw data file. Two different types of input, column and formatted, are used in this example to show you that it is possible to mix input methods within one INPUT statement.

The other method of moving to a new line of data is to use a number sign (#), also known as a line pointer. The input statement in example 12 would be changed to look as follows ...

```
input
id      $ 1-5
gender  $ 6
#2
(grade1-grade5) (3.)
;
```

Rather than using a slash, the #2 instructs SAS to move to a second record in the raw data file. If data were present on three records, another slash followed by more variable names could be used. A number sign followed by a new record number, #3, could also be used.

The slash can also be used to skip records in a raw data file. The next example shows an instream file containing a county number, a gender, and populations for residents in six different age groups. Only the raw data for males are to be read into data set pop_m (gender is 'M').

...Example 3.15...
```
data pop_m;
input county $3. +1 pop1-pop6 / ; ①
datalines; ②
001M   1862   2015   2000   2050   2162   9943
001F   1820   1821   1954   2024   2129   9682
003M    344    342    396    392    430   2011
003F    331    322    350    364    386   1901
005M  11615  11587  11800  11475  11566  51125
005F  11234  11132  11208  11091  11458  48926
;
run;
```

The INPUT statement has a few parts: formatted input is used to read the value of the variable COUNTY; a relative pointer (+1) is used to move to column 5; list input is used to read the values of POP1 through POP6; a slash is used to skip to the next record ①. The first time the INPUT statement is used, it reads record 1 in the raw data. After all the data are read, the slash causes SAS to skip to record 2. The next time the data step uses the INPUT statement, it will go to the next record in the raw data, record 3, another set of data for males. The process continues until there are no more records to be read.

...Different Data Across Records (@ Sign)...
An input statement instructs SAS to read data in a raw data file in a specific manner. If you have one input statement in a data step, you assume that the data for each observation are present in the same manner across all records in your raw data file. If the location of data changes across records, you may need more than one input statement. In the following example, the data for males and females have been entered in different orders.

...Example 3.16...

```
data m_f;
input
@01 gender $1.
@02 age     3.
@  ①
;

if gender eq 'M' then input (ldl hdl) (3.); ②
else                  input (hdl ldl) (3.);

pct_hdl = 100 * hdl / (ldl + hdl); ③
datalines;
M025120076
F033080200
M050070080
;
run;
```

Values for the variables GENDER and AGE are read and an @ sign is used at the end of the input statement to tell SAS to remain on the current line of raw data when the next input statement is encountered ①. The normal behavior of SAS is to move to a new line of raw data each time an input statement is encountered. After the value of AGE is read, the pointer shown in earlier examples is in column 6 of the raw data. The gender read in the first input statement is used to tell SAS how to read the remaining data in the file of raw data ②. For males, LDL occurs prior to HDL. For females, HDL occurs prior to LDL. In either case, the INPUT statement starts in column 4 and uses an informat list to read the values of three variables. This is the first time in these notes that an IF-THEN-ELSE statement has been used. There will be more about this type of statement later in the notes. An equation is used to calculate the percentage of total cholesterol that is HDL ③.

...Data for More than One Observation in One Record (Double At Sign)...
Raw data files sometimes have data for more than on observation present in a single record. The normal behavior of the INPUT statement is to move to a new line of raw data. Also, the normal behavior of a data step is to release any line of raw data held by an @ sign (as was done in example 3.16) once the end of the data step is reached. In the next example, there are data for multiple observations within each line of raw data.

...Example 3.17...

```
data many_obs;
input id
age hr @@;  ①
datalines;  ②
12345 34 76 23456 10 55
9999 100 76
987 5 80 12121 54 80
13131 65 65
;
run;
```

Two @ signs are used to give SAS two instructions ①. The first is to leave the line pointer in the record just read when the next input statement is encountered. The second instruction is to hold the location even when the end of the data step is encountered. The raw data file has data for individual observations stored in various manners within and records ②. Some of the records contain data for more than one person.

In the last examples, the location of data changed across records. It is also possible to have different data in records in on file. In either case, you can read the file properly if you understand the role of slashes, number signs, line pointers, the @-sign, and the @@-sign.

What you should remember...
Informat lists can be used to reduce the effort in writing an INPUT statement (that is the only real purpose of an informat list). There are many options you can use within an INPUT statement that allow you to read raw data that is in a non-standard format. You should remember the role that the following play in an INPUT statement: slash (/), and if you remember that you do not have to remember what the # does; one @ sign; two @ signs.

..."LOOSE (SEMI-IMPORTANT) ENDS"

There are a few more topics you should know about. You will notice that this section does not end with what you should remember. For this course, you need to remember when the LRECL option is needed. The other topics (TRUNCOVER, FILEREF, AMPERSAND modifier, LENGTH) are important, but optional for your SAS knowledge right now.

...Missing Data, Short Records and the TRUNCOVER Option...
In example 3.1, a period is used to tell SAS that there was no data for variable SBP in the fourth record (JOHNSON) and no data for DOD in the last record (CLINTON). Example 3.2 shows what happens when those periods are removed. The default behavior of SAS is to try to find values for all the variables in the INPUT statement. If not enough values are found in one record of raw data, the INPUT statement moves to the next record. That default type of behavior has a name, FLOWOVER. You can use an option on the INFILE statement to change this default behavior.

...Example 3.18...

```
data presidents;
infile datalines truncover; ①
input
lastname  : $12.
chol
sbp
dbp
dob       : mmddyy.
dod       : mmddyy.
;

aod = (dod - dob) / 365.25;
format dob dod date9. aod 4.1;
```

```
datalines;
ADAMS 230 160 90 04111767 02231848
FILLMORE 300 156 88 01071800 03081874
LINCOLN 222 144 82 02121809 04151865
JOHNSON 190 60 12291808 07311875      ②
TRUMAN 150 140 70 05081884 12261972
WASHINGTON 110 150 75 02221732 12141799
CLINTON  200 140 085 08191946    ③
;
run;
```

The TRUNCOVER option is added to the INFILE statement, replacing the default behavior of FLOWOVER ①. In this example, the TRUNCOVER option assigns missing values to any variables that do not have values when the end of a record is encountered. The INPUT statement tries to read the values of six variables from each record in the DATALINES file. The records for JOHNSON ② and CLINTON ③ only contain five values. If you print the data set after the completion of the data step, it wil look as follows.

NO PERIODS AS SPACE HOLDERS, TRUNCOVER OPTION USED

Obs	lastname	chol	sbp	dbp	dob	dod	aod
1	ADAMS	230	160	90	11APR1767	23FEB1848	80.9
2	FILLMORE	300	156	88	07JAN1800	08MAR1874	74.2
3	LINCOLN	222	144	82	12FEB1809	15APR1865	56.2
4	JOHNSON	190	60	12291808	31JUL1875	.	.
5	TRUMAN	150	140	70	08MAY1884	26DEC1972	88.6
6	WASHINGTON	110	150	75	22FEB1732	14DEC1799	67.8
7	CLINTON	200	140	85	19AUG1946	.	.

If you compare the above table to the one at the bottom of page 29 (after example 3.2), you can see that the TRUNCOVER option did fix some of the problems, but not all of them. There now is one observation for each record in the DATALINES file. CLINTON has a missing value for both DOD and AOD. However, the values of the variables for JOHNSON are incorrect. The missing data for CLINTON (DOD) occurs at the end of the record, while that for JOHNSON occurs in the middle of the record. For CLINTON, TRUNCOVER resulted in correct values for all variables. For Johnson, the values for all variables in the INPUT statement starting with SBP are incorrect. The morale to this example is that while TRUNCOVER may appear to correct some problems with your raw data, it should be used with caution.

There is one other problem that the TRUNCOVER option is meant to handle, short records. Look at the listings of the two raw data files shown on the right. They both contain the same values. However, the file shown on the top comprises variable length records (records that are only as long as the data that they contain) while the one on the bottom comprises fixed length records (all records are 37 characters long regardless of their content).

```
ADAMS      23016090 0411176702231848
BUSH
FILLMORE   300156 880107180003081874
LINCOLN    222144
JOHNSON    190    601229180807311875
TRUMAN     150140 7005081884
WASHINGTON 11015075 0222173212141799
CLINTON    20014008508191946
```

```
ADAMS      23016090 0411176702231848
BUSH
FILLMORE   300156 880107180003081874
LINCOLN    222144
JOHNSON    190    601229180807311875
TRUMAN     150140 7005081884
WASHINGTON 11015075 0222173212141799
CLINTON    20014008508191946
```

If you looked at both of these files without the highlighting, you could not tell the that they were different. However, if you try to read them with formatted input, they produce different results.

...Example 3.19...

```
data p_variable;
infile 'k:\epi514\data\president_variable.txt' truncover; ①
input
@01 lastname  $12.
@13 chol        3.
@16 sbp         3.
@19 dbp         3.
@22 dob   mmddyy8.
@30 dod   mmddyy8.
;

aod = (dod - dob) / 365.25;
format dob dod date9. aod 4.1;
run;
```

The TRUNCOVER option is added to the INFILE statement ①. The INPUT statement is expecting a record that is 37 characters long. If it encounters a short record, as it would with a file with records that have a variable length, the default FLOWOVER behavior would cause SAS to move to a new record to find more data to satisfy the requirement of assigning a value to each variable in the INPUT statement. The TRUNCOVER option tells SAS to assign missing values to all variables when a short record is encountered rather than to move to a new record to find more data. Without the TRUNCOVER option, the data set looks as shown on the top right. With the TRUNCOVER option, the data set looks as shown on the lower right.

PRESIDENTS - READ VARIABLE LENGTH RECORDS - NO TRUNCOVER OPTION

Obs	lastname	chol	sbp	dbp	dob	dod	aod
1	ADAMS	230	160	90	11APR1767	23FEB1848	80.9
2	FILLMORE	300	156	88	07JAN1800	08MAR1874	74.2
3	LINCOLN	222	144	.	29DEC1808	31JUL1875	66.6
4	TRUMAN	150	140	70	08MAY1884	.	.

PRESIDENTS - READ VARIABLE LENGTH RECORDS WITH TRUNCOVER OPTION

Obs	lastname	chol	sbp	dbp	dob	dod	aod
1	ADAMS	230	160	90	11APR1767	23FEB1848	80.9
2	BUSH	.	.	.	.	.	.
3	FILLMORE	300	156	88	07JAN1800	08MAR1874	74.2
4	LINCOLN	222	144	.	.	.	.
5	JOHNSON	190	.	60	29DEC1808	31JUL1875	66.6
6	TRUMAN	150	140	70	08MAY1884	.	.
7	WASHINGTON	110	150	75	22FEB1732	14DEC1799	67.8
8	CLINTON	200	140	85	19AUG1946	.	.

...Long Records and the LRECL Option...

When using SAS on a PC, the default longest record that you can read with an INPUT statement is 256 characters. If your file of raw data contains records that are longer thatn 256 characters, you must use an LRECL option on the INFILE statement and specify the record length

...Example 3.20...

```
data mystuff;
infile 'k:\epi514\data\amc99.txt' lrecl=445; ①
input @349 los   4.;
label los = 'LENGTH OF STAY';
run;
```

The LRECL option is used to tell SAS to expect records that are 445 characters long ①. The LOG file at the top of the next page shows that SAS read the file correctly.

```
577  data mystuff;
578  infile 'k:\epi514\data\amc99.txt' lrecl=445;
579  input @349 los   4.;
580  label los = 'LENGTH OF STAY';
581  run;

NOTE: The infile 'k:\epi514\data\amc99.txt' is:
      File Name=k:\epi514\data\amc99.txt,
      RECFM=V,LRECL=445

NOTE: 941 records were read from the infile 'k:\epi514\data\amc99.txt'.
      The minimum record length was 445.
      The maximum record length was 445.
NOTE: The data set WORK.MYSTUFF has 941 observations and 1 variables.
```

What if the LRECL option had been left off of the INFILE statement? Here is the LOG file.

```
585  data mystuff;
586  infile 'k:\epi514\data\amc99.txt';
587  input @349 los   4.;
588  label los = 'LENGTH OF STAY';
589  run;

NOTE: The infile 'k:\epi514\data\amc99.txt' is:
      File Name=k:\epi514\data\amc99.txt,
      RECFM=V,LRECL=256

NOTE: 941 records were read from the infile 'k:\epi514\data\amc99.txt'.
      The minimum record length was 256.
      The maximum record length was 256.
      One or more lines were truncated.
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.MYSTUFF has 470 observations and 1 variables.
```

Unless you read the LOG carefully, you may not notice that while the data step read 941 records from the file of raw data, there are only 470 observations in the data set. Compare that to the LOG at the top ofthe page where you can see that the data set has 941 records. What happened? Look at the second LOG and you can see that SAS thinks that the longest record in your raw data is only 256 characters long, the default value for the maximum length. SAS used the default maximum record length of 256 characters when reading the file. Since the INPUT statement specifies to read data at column 349, SAS went to the next record to find the data. This is a good lesson that shows how important it is to look for messages in the SAS LOG. There are no ERRORS and you have a SAS data set. However, it's not the data set you really wanted.

The LRECL option just used above statement tells SAS to expect record that contain 445 characters. If you are not sure of the record length, but know that it is longer than 256 characters, you can add a PAD option to the INFILE statement ...

```
infile "k:\epi514\data\amc99.dat' lrecl=500 pad;
```

The above statement results in the LOG file shown on the right. The data are read correctly. The PAD option tells SAS to expect records up to length specified with LRECL. If any records in the input data are shorter than the specified value, the PAD option fills the record with blanks.

```
790  data mystuff;
791  infile 'k:\epi514\data\amc99.txt' lrecl=500 pad;
792  input @349 los   4.;
793  label los = 'LENGTH OF STAY';
794  run;

NOTE: The infile 'k:\epi514\data\amc99.txt' is:
      File Name=k:\epi514\data\amc99.txt,
      RECFM=V,LRECL=500

NOTE: 941 records were read from the infile 'k:\epi514\data\amc99.txt'.
      The minimum record length was 445.
      The maximum record length was 445.
NOTE: The data set WORK.MYSTUFF has 941 observations and 1 variables.
```

...Filename Statement (Fileref)...
An alternative to putting the name of the file directly in the INFILE statement is to use a combination of FILENAME and INFILE statements.

...Example 3.21...
```
filename mydata 'k:\epi514\data\amc99.txt';  ①

data mystuff;
infile mydata lrecl=445;  ②
input @349 los   4.;
label los = 'LENGTH OF STAY';
run;
```

The FILENAME statement establishes a fileref (file reference), in this example it is 'mydata' ①. From then on, anytime SAS sees the fileref, it knows that it refers to the external file associated with the fileref in the FILENAME statement, AMC99.TXT. The INFILE statement uses the fileref instead of the full name of the external file and SAS knows to look for data in the external file AMC99.TXT.

The rules for the naming filerefs are the same as for naming SAS variables and data sets. Notice that the FILENAME statement is not located within a data or PROC step. It is a GLOBAL statement. There is no particular advantage in using a FILENAME statement in the last example, in fact it violates the rule of "...the less typing, the better..." in that there are more keystrokes needed.

There are instances where you might want to set up filerefs at the top of your SAS job rather than embed the full file names within the data step. One such instance might be if you are writing a SAS job that you will use with numerous different data files. It is easier to alter one or more FILENAME statements at the top of the SAS job than it is to hunt through the SAS code for all the INFILE statements and change the names of the external files.

Another advantage of the FILENAME statement is that it allows you to read multiple files within a data step. You can only specify one filename in an INFILE statement, but you can specify more than one external file within a FILENAME statement. For example, if you had three files containing hospital-based data and wanted to use an INFILE statement within a data step to tell SAS to read all three of them, you could use the following approach.

...Example 3.22...
```
filename all3 ('k:\epi514\data\amc97.txt' 'k:\epi514\data\amc98.txt' 'k:\epi514\data\amc99.txt'); ①

data amc97_99;
infile all3 lrecl=500 pad; ②
input
@027 dischyr   $4.
@349 los        4.
;
run;
```

```
NOTE: The infile ALL3 is:
      File Name=k:\epi514\data\amc97.txt,
      File List=('k:\epi514\data\amc97.txt' 'k:\epi514\data\amc98.txt' 'k:\epi514\data\amc99.txt'),
      RECFM=V,LRECL=500

NOTE: The infile ALL3 is:
      File Name=k:\epi514\data\amc98.txt,
      File List=('k:\epi514\data\amc97.txt' 'k:\epi514\data\amc98.txt' 'k:\epi514\data\amc99.txt'),
      RECFM=V,LRECL=500

NOTE: The infile ALL3 is:
      File Name=k:\epi514\data\amc99.txt,
      File List=('k:\epi514\data\amc97.txt' 'k:\epi514\data\amc98.txt' 'k:\epi514\data\amc99.txt'),
      RECFM=V,LRECL=500

NOTE: 1330 records were read from the infile ALL3.
      The minimum record length was 445.
      The maximum record length was 445.
NOTE: 1320 records were read from the infile ALL3.
      The minimum record length was 445.
      The maximum record length was 445.
NOTE: 941 records were read from the infile ALL3.
      The minimum record length was 445.
      The maximum record length was 445.
NOTE: The data set WORK.AMC97_99 has 3591 observations and 2 variables.
```

The FILENAME statement creates the fileref 'all3' and that fileref refers to three raw data files ①. An INFILE statement uses the newly cr**ea**ted fileref ②. The LOG on the right shows that all three files were read in the data step. There are other ways to read multiple raw data files with one data step. The method shown here is the least complicated.

...List Input Modifiers (Colon and Ampersand)...
There are a number of modifiers that can be used with LIST input. The colon modifier has already been used in a number of examples ... from example 3.1 ...

```
input
lastname  : $12.
chol
sbp
dbp
dob       : mmddyy.
dod       : mmddyy.
;
```

to tell SAS to use an informat when reading the values of variables. In example 3.1, the informat $12. results in LASTNAME being a character variable with a length of 12. The informat MMDDYY. is used twice and variables DOB and DOD are numeric variables stored as dates. It is very important to know the the difference between the following statements ...

```
input lastname : $12.;
```

and

```
input lastname  $12.;
```

The first version, with the colon, tells SAS to read an observation starting in column one up to the occurrence of the first delimiter (space, comma, tab, whatever). The $12. after the colon tells SAS that the data are character and that the variable LASTNAME will have a length of 12. The second version without the colon also tells SAS to start reading an observation in column one. However, it also tells SAS to read the first twelve characters in that observation, regardless of the occurrence of any delimiter.

Another useful modifier with list input is the ampersand (&). The & tells SAS that at least two spaces are needed to separate the values of variables.

...Example 3.23...

```
*LIST input with an ampersand (&) modifier;
data mystuff;
input
lastname & : $20.  ①
chol
sbp
dbp
(dob dod) (: mmddyy.) ②
;
aod = (dod - dob) / 365.25;
format dob dod date9. aod 4.1;
datalines;
JOHN QUINCY ADAMS   230 160 90 04111767 02231848
MILLARD FILLMORE  300 156 88 01071800 03081874
ABRAHAM LINCOLN   222 144 82 02121809 04151865
ANDREW JOHNSON  190 .  60 12291808 07311875
HARRY TRUMAN  150 140 70 05081884 12261972
GEORGE WASHINGTON  110 150 75 02221732 12141799
BILL CLINTON   200 140 085 08191946 .
;
run;
```

The ampersand (&) tells SAS to ignore any single spaces embedded within the value of the variable name and to look for at least two spaces before assigning a value to that variable ①. The colon followed by $20. tells SAS to allow up to 20 spaces to store the name. The variables DOB and DOD are treated as a dates by using a colon followed by MMDDYY ②. Since both dates share the same informat, the variable names are put in parentheses and they are followed by the informat in parentheses. This is an informat list as has been seen previously, but the colon (:) is needed within the parentheses to tell SAS that LIST input is being used.

...Length Statement...

The combination of the colon modifier and the informat in the various example of LIST input tell SAS that character variables were not the standard length, the content of variables was longer than the default of eight characters. There is yet another way to associate a length with a variable, a LENGTH statement.

...Example 3.24...

```
*use a LENGTH statement instead of informats in the INPUT statement;
data mystuff;
length lastname $12; ①
input
lastname ②
chol
sbp
dbp
(dob dod) (: mmddyy8.)
;
datalines;
ADAMS    230 160 90 04111767 02231848
FILLMORE  300 156 88 01071800 03081874
LINCOLN   222 144 82 02121809 04151865
JOHNSON  190 .  60 12291808 07311875
TRUMAN  150 140 70 05081884 12261972
WASHINGTON  110 150 75 02221732 12141799
CLINTON   200 140 085 08191946 .
run;
```

The LENGTH statement tells SAS that lastname is a character variable whose content can be up to twelve characters ①. No informat is needed to tell SAS that lastname is a character variable (that has been taken care of in the LENGTH statement) ②. If you use PROC CONTENTS to look at the attributes of the variables in the data set, you will see the output shown on the right.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
2	chol	Num	8
4	dbp	Num	8
5	dob	Num	8
6	dod	Num	8
1	lastname	Char	12
3	sbp	Num	8

Up to now, the output from PROC CONTENTS has always shown the length of numeric variables as 8. You can also use a LENGTH statement to change the length of numeric variables, but can be a bit risky unless you are well aware of the implications of making such a change.

Length in Bytes	Significant Digits Retained	Largest Integer Represented Exactly
3	3	8,192
4	6	2,097,152
5	8	536,870,912
6	11	137,438,953,472
7	13	35,184,372,088,832
8	15	9,007,199,254,740,992

The first rule in changing the length of numeric variables is to *only change the lengths of integers*. The second rule is to use the table on the right to determine the largest integer (last column) that can be stored given the various lengths (first column). The third rule is *do not change the length of numeric variables unless you have space issues in storing data sets.*