4. Reading Data in CREATE

4.1 Overview

Specifying the reading of data from external files is the first task of the CREATE step, and the standard order, shown in Exhibit 1.1, places the input section first. Furthermore, Exhibit 1.4 presented an example of a CREATE step with only an input section, since it is the only essential section that cannot be replaced by defaults. To expand upon the observation of Chapter 1, however, there are several benefits of placing the descriptions of the recode, categorization, and output sections in Chapters 2 and 3 before this chapter on the input section:

- In many VPLX applications, a single input file may be produced for VPLX from another system, such as SAS or a database. One person may develop an input section, which may be used repeatedly by a series of VPLX applications. Indeed, the INCLUDE statement, to be described in Chapter xx, facilitates reuse of segments of VPLX code. Consequently, a VPLX novice may conduct multiple analyses without ever needing to modify an input section provided by someone else.

  The SIPP example of Exhibit 1.5 includes an INPUT statement to read all of the data from the extract. Consequently, this INPUT statement can remain the same for all subsequent uses of the extract in this documentation.

- In some applications, the input section may be automatically written by computer software to an external file. A CREATE step can simply INCLUDE this file without modification, leaving the user to develop other sections of the CREATE step for the application.

- The input section may incorporate any of the syntax of the recode section. In particular, Section 4.3 describes how IF-THEN-ELSE may be used in conjunction with reading linked files.

Section 4.2 describes the features of the INPUT statement. Previous examples have read only from the primary input file, specified as the IN= file in the CREATE statement. Section 4.3 introduces LINK, which is used to identify additional files as sources for input data. Section 4.4 describes a more advanced feature, the /INFLAG specification in the INPUT statement, which may be used to link files where only some of the records match. Section 4.5 discusses programming strategies for reading data into VPLX from multiple files in complex situations.

4.2 INPUT

4.2.1 General Description Several previous examples have employed the same INPUT statement,

```
input  rooms persons  weight cluster    ! input statement to read
        stratum /format (5F2.0) ;       ! data  ("input section")
```

A modified version illustrates the available features for the primary INPUT statement,

```
input  rooms persons  weight cluster stratum
       /format (5F2.0) /options nprint=1 nobs=6 ;
```

where the /FORMAT and /OPTION (or /OPTIONS) specifications may appear in either order. Section 4.3 describes how /KEY specifications may be added to subsequent INPUT statements for linked files. Section 4.4 covers the /INFLAG specification.

As evident from all previous examples, a list of one or more variables follows INPUT. Typically, the variables in the list have not been previously defined, and they become defined by inclusion on the list. With linked files, however, one or more of the variables may have already been defined *(1)*.

There are two possible options associated with INPUT, both for primary and linked files. If NPRINT is set to a positive integer, *n*, the first *n* observations will be printed. In Exhibit 1.5, NPRINT=1 requested the printing of the first observation. If NPRINT is set to a value equal to or greater than the number of observations, then all observations will be printed. A good practice is to examine some number of observations, NPRINT=1 or NPRINT=10, for example, for any new or modified INPUT statement or previously unread file. Many mistakes can be detected simply by examining a few records.

The NOBS (or OBS) option specifies a maximum number of observations to be read from the file. If the specified number of observations is reached, then the next attempt to read the file is treated as an end-of-file condition, even if more observations remain on the file. This feature permits testing of the code on a short section of the file. If NOBS specifies a number equal to or greater than the number of observations on the file, it has no effect.

4.2.2 Kinds of Data That VPLX Reads. In general, the /FORMAT feature of the INPUT statement and related uses of FORMAT elsewhere in VPLX are among the most Fortran-constrained features of the system. VPLX reads only one type of data, real (double precision) numbers. Integers may be read in this form, but alphabetic strings may not be.

Exhibit 4.1 presents 7 examples of data that VPLX can read through Fortran formats, followed by 2 examples that it cannot.

example	positions on record 1 1234567890	possible format
a.	1	f1.0
b.	1.	f2.0
c.	27	f2.0
d.	2.7	f3.1, f3.0 (decimal overrides)
e.	.27000D+02	d10.5
f.	.27000E+02	d10.5
g.		f1.0 (many others)
BUT NOT		
h.	.	
i.	Y	

Exhibit 4.1 Examples of types of data that VPLX can and cannot read. Position numbers 1-10 are indicated above the examples by the 2-line numbered key. In most cases, many other formats in addition to the ones shown may be used. Example *g* will be read as 0. Because of Fortran standards, VPLX cannot reliably read the solitary point illustrated by example *h* with floating-point edit descriptors in the Fortran format. Except for the special meaning of "D" or "E" as they appear in examples *e* and *f,* VPLX cannot read characters such as the one in example *i.*

VPLX is unable to read character strings of alphabetic characters, e.g., "Y" in example *i*, "CPS", *etc*. It is, however, able to interpret examples *e* and *f,* which represent numbers in scientific notation.

Fortran, and therefore VPLX, reads fields that are entirely blank, such as in example *g*, as "0" when reading real numbers. This is different from SAS, which generally treats blanks as missing.

SAS reads fields containing only "." as a missing value, as in example. Fortran generally does not tolerate these values, although the treatment varies from one Fortran environment to another. Consequently, fields of "." alone must be avoided as input to VPLX. (Of course, a decimal point included as part of a number, as in examples *b* and *d*, is perfectly acceptable to Fortran.)

4.2.3 /FORMAT The /FORMAT specification provides a Fortran format of the incoming variables. The format should describe the layout of the entire observation, corresponding to the list of variables on the INPUT statement, which may extend over more than one record. The primary edit descriptors are

C "F" and "D" format for input variables,

C "X" to denote skipped spaces,

C "/" to read the next record to continue processing the same observation, and

C "(" and ")" to bound the format and segments of description to be used multiple times.

Each of these is described more below. Except to change "f", "d", and "x" to upper case, and to remove blanks, VPLX does not edit or scan the format. It relies on the Fortran implementation of the format to read the input file.

The format should specify exactly as many fields as the number of variables in the INPUT statement, even though Fortran does not require this restriction. If more variables are listed than the format specifies, Fortran begins to reuse part or all of the format, with usually incorrect results in VPLX applications. Section 4.2.5 discusses possible difficulties when the format specifies more fields than the length of the variable list.

VPLX will squeeze out extraneous blanks and remove comments from the format before storing it for use. There is a limit, depending on the installation, for the total length of the format after extraneous information is removed. For the PC version, this limit is 2560 non-blank characters *(2)*, which has not yet appeared to be a practical problem. VPLX will print an error message if the limit is exceeded.

4.2.4 Fortran Edit Descriptors. Fortran edit descriptors describe the contents of positions on one or more records. The primary forms for VPLX are:

F*w.d* - For a right-justified floating-point number. The width, *w*, is a positive integer (>0) specifying the total number of positions occupied by the number, and *d* is a non-negative integer ($0 \# d \# w$) specifying the number of places after the decimal. (This corresponds closely to *w.d* in SAS.) If a decimal point is present in the data, then it automatically overrides *d*, that is, the number will be correctly read regardless of *d*. Without a decimal, if *d*=0, then the number will be read as the real (floating point) representation of an integer.

For example, F1.0 may be used to read a number in the range 0-9 from a single position. In Exhibit 4.1, F1.0 translates example *a* into the number 1. F2.0 translates the two characters "27" in example *c* into the number 27, but F2.1 translates them into 2.7 and F2.2 into .27.

The decimal present in example *d* overrides any distinction between F3.0, F3.1, F3.2, or F3.3; thus, 2.7 is read in all four cases.

Although leading zeroes are completely unnecessary before a decimal point, leading zeroes after an implied decimal point help to assure that the value is correctly read. For example, the number 4 will be correctly read from " 4" by F3.0. F3.3 will safely read "004" as .004, but reading " 4" with F3.3 may produce results that vary by Fortran implementation, and it is essential to avoid this situation. (If no decimal points appear in the input data, a safe alternative in this case is to read the number with F3.0 and then divide it by 1000 in the recode section.)

Warning: Left-justified numbers, with trailing blanks in the range covered by *w*, pose potential problems for some Fortran implementations. For example, all Fortran implementations read " 4" with F3.0 as 4., but their reading of "4 " is less predictable. For safety, users should always right-justify variables. (Users who employ SAS to manipulate their data for input to VPLX can accidentally produce left-justified data by using character variables. The situation is safer with real (numeric) items, however, since SAS right-justifies real (numeric) data output with a SAS PUT statement from a DATA step.) Further comments follow in Sections 4.2.5 and 4.2.6.

D*w.d* - For right-justified scientific notation in Fortran. The number is expressed as a decimal fraction followed by a power of 10. The width, *w*, is a positive integer (>0) specifying the total number of positions occupied by the number, and *d* is a non-negative integer ($0 \# d < w$) specifying the number of places after the decimal for the fractional part. (Fortran and other systems that write data with this format include the decimal point of the fraction.) For example, " .200D+01" is read by D9.3 as 2, which is the product of the fraction, .2, and exponent, 10^1. Both examples *e* and *f* in Exhibit 4.1 are read with D10.5 as 27. This descriptor is infrequently used with survey data *(3)*.

*w*X - To skip *w* positions. This may be used to skip over blanks, data not referenced by INPUT, and alphabetic data that VPLX cannot read. For example

```
input  psu  segment / format (f3.0,3x,f4.0);
```

would skip over positions 4-6, which could contain "CPS" or other data that VPLX cannot read.

/ - To skip to the next record in the observation. For example:

1) If an observation is on a single record, "/" should not be used.

2) If the observation is on two records, "/" should be used once to indicate the point at which reading should begin from the second record. If the information is only on the second record, "/" should precede other descriptors to indicate that Fortran should go to the second record immediately. For example

```
input  psu  segment weight repw1 - repw48 /
    format  (/,f3.0,3x,f4.0,49f8.2) ;
```

skips the first of two records and goes immediately to the second. If the information is instead only on the first record, the format should end with ", /)".

```
input  psu  segment weight repw1 - repw48 /
    format  (f3.0,3x,f4.0,49f8.2,/) ;
```

to indicate that the second record should be skipped over entirely.

3) If the observation is on three records, two "/"'s should appear in the format, *etc*.

Both F*w.d* and D*w.d* may be preceded by a multiplier. For example, "3F2.0" in a format is equivalent to "F2.0,F2.0,F2.0". Besides saving space, use of multipliers often makes the format easier to check.

The format must start with "(" and end with ")", but internal sets of parentheses can be set up, typically preceded by a multiplier. This feature is illustrated in the following example.

```
create  in = temp$:[r_fay]w2vplx1.dat out = temp$:[r_fay]vplx1.vpl ;
     input   sexw2  agew2 degree  degreex tm8428 racew2 workact waitw2
       totinc earnings  repw1 - repw100 / option nprint=2
      /format  (7x,f1.0,f3.0,3x,f1.0, ! sexw2 8  agew2 9-11 degree 15
                1x,f1.0,f2.0,         ! degreex 17  tm8428 18-19
                2f1.0,f8.5,           ! racew2 20  workact 21  waitw2 22-29
                2f10.1,               ! totinc 30-39  earnings 40-49
              12(/,8f10.7),/,4f10.7); ! repw1 - repw100
```

Exhibit 4.2 In this example, the input statement defines 110 variables to be read from the input file, and the format statement that follows specifies their locations on a file that has 14 records per observation. The first variable, sexw2, is read from position 8 of the first record. The second, agew2, occupies positions 9-11. Because 3 more positions are then skipped, degree is read from position 15. The remaining variables up through earnings are read from the first record. The next part of the format gives a segment to be used 12 times. On each of the 12 uses, the first step is to go to the next record, so that, for example, repw1 is read from positions 1-10 of the second record. Note that, because the observation corresponds to 14 records, the format accounts for 13 "/"'s.

Heavily commenting a format, as in the preceding example, facilitates checking and documentation. A similar style appeared in Exhibit 1.5. Since VPLX stores the format without the comments and extraneous blanks, no additional storage resources are required by this useful practice.

4.2.5 Verifying INPUT and /FORMAT. Errors in the format or mismatches in length or order between the list of input variables and the format are perhaps the most frequent source of user error during the CREATE step. Unlike spelling and syntactic errors, which VPLX generally spots immediately, errors in the format do not surface until VPLX attempts to use it to read the input file. It is good practice to devote careful attention to the format and input list initially and then to change them as little as possible. It is usually better to read in all of the variables from the file and simply

omit unnecessary ones from the BLOCK specifications than to modify a working INPUT and FORMAT pair in order to avoid reading extraneous information. (This is an instance where “if it's not broken, don't fix it” applies.)

The format should specify exactly the number of variables to be read. If the variable list is longer than the format specifies, then Fortran will reuse the format at a point depending on the imbedding of parentheses *(4)*. This automatic feature of Fortran almost always leads to misreading some of the data. No Fortran error condition is generated, however, unless the application of the format leads to an attempt to read inappropriate data because of imbedded blanks or extra decimal points, as in attempting to read “3 0” or “.3.” with F3.0. Thus, review of printed observations for unexpected values for any of the input variables, but particularly those at the end of the list, is necessary to detect format specifications shorter than the input list. Using NPRINT=2 or more, as in Exhibit 4.2, lets the user check that both the first and the second records are correctly read. The effect of many format errors show up by the first two records.

Fortran also permits a format to specify more fields than variables. There are two reasons to avoid this option, however. The first is that even after Fortran has read all of the variables, it continues to scan the format and implement the “/” specification to read the next record until it encounters the first unnecessary F or D field specification. Thus, the outcome when “/” specifications are present may be different than anticipated. The second reason is that difficulties have resulted in applications from Sun Microsystem’s FORTRAN 77 implementation that were not reproducible when the problem was moved to a different Fortran environment. In fact, one or more errors frequently occurred at unpredictable points after the first record, often well into a large data file. Because this sort of error was initially difficult to isolate, the experience has led to the recommendation of always matching the number of fields specified by the format to the length of the variable list.

Printing the first 10 observations or so often reveals errors in the input list or format specification, and even NPRINT = 1 or 2 is often enough. In addition to providing possible evidence of a mismatch between the length of the input list and the fields specified by the format, reading of the incorrect positions on the record may also be evident.

Besides checking some individual records with NPRINT, a second general strategy to verify the reading of data is to DISPLAY the totals for each of the variables and to check them against another source. Marginal totals may be published or easily produced in another system, such as SAS.

The Fortran F*w.d* specification with *d*>0 allows an implied decimal point to be omitted from an input file. On output, however, Fortran always inserts the decimal point with F*w.d*. Consequently, as a general rule, it is better to use F*w*.0 instead of F*w.d* in the /FORMAT for VPLX, for the following reasons:

C Any data written from Fortran in F*w.d* may be safely read by VPLX with F*w*.0, since the decimal point in the data will be correctly interpreted.

C This strategy may be better for data written by a non-Fortran system. In one instance *(5)*, a non-Fortran system writing data in the form equivalent to F*w.d* wrote some values as right-justified integers without the decimal point. These data were misinterpreted with a /FORMAT specification of F*w.d.* This problem, which was subtle, was only discovered by comparing totals. This problem was corrected by changing the /FORMAT to F*w*.0.

4.2.6 Writing Data from SAS Some features of the SAS system pose potential difficulties when the data are written out to a character (ASCII) file using PUT within the DATA step. The features affect not only VPLX but other Fortran programs.

SAS files are typically considerably smaller when numeric variables of one or a few digits are stored as character data, which has encouraged the practice of doing so. Unfortunately, this approach requires additional programming care. The author has observed a number of processing errors and difficulties to result from this one feature.

Both Fortran and SAS output numeric data right justified with leading blanks. The most familiar format form for character data in SAS left-justifies character data. For example,

```
input  psu $ 1-5 ;
```

would left-justify " 1001" to the character string "1001 ". If the field was then written to another file,

```
put  psu $ 1-5 ;
```

the output remains left-justified. As noted earlier, this justification may cause a Fortran error if the field is read with F5.0. This problem may be avoided by consistently using the $char*w.* format instead. In this instance,

```
input @1 psu $char5.;
put @1 psu $char5. ;
```

The $char*w.* format must be used in both input and put statements to achieve this; in particular, if $char*w.* is not used in the input statement, its use in the put statement only will not achieve the required result. Alternatively, character data may be converted to numeric in order to write it,

```
input psu $ 1-5 ;
psucode = input(psu,5.);
put psucode 1-5 ;
```

4.3 LINK

4.3.1 General LINK and LINK_MISSING are available to specify files from which the CREATE step is to read to obtain additional variables. The first LINK statement, if present, may not appear until the first INPUT statement has specified the reading of the primary IN = file in the CREATE statement. The form of the statement is,

```
link filename ;
```

Each LINK statement must be followed by a corresponding INPUT statement describing how to read records from the file, and this INPUT statement must appear before any subsequent LINK or LINK_MISSING statements. LINK_MISSING is described in Section xx of Volume III as an advanced feature *(6)*.

There are different ways in which data from LINKed files may be associated with the observations read from the primary (IN=) file. These different options are not reflected in the LINK statement itself but by the corresponding INPUT statement and its placement. The separate options are discussed in the remainder of this section.

4.3.2 One-to-one matching with keyed LINK Linking files by matching keys provides a measure of assurance that the correct data from the LINKed file are being associated with each observation. The following example illustrates keyed link to a file that results in a one-to-one match.

```
create  in = i1-1.dat  out = vplx1.vpl ; ! Beginning of CREATE step

Note: Fortran unit 19 assigned to i4-1.dat

  input  rooms persons  weight cluster   ! input statement to read
          stratum /format (5F2.0) ;      ! data  ("section #1")

  link  i4-1.dat;                        ! linked file
  input stratum cluster tenure           ! variables to read from file
      / format (3f2.0)
      / key stratum cluster ;            ! key variables already defined

/* the file i1-1.dat contains
 5 7 1 1 1
 6 8 1 2 1
 5 2 1 3 2
 4 1 1 4 2
 8 4 1 5 3
 8 2 1 6 3
 the file i4-1.dat contains
 1 1 2
 1 2 2
 2 3 1
 2 4 2
 3 5 1
 3 6 1
```

```
*/

  print cluster tenure ;
  class tenure (2/1,3) 'Renter' 'Owner or other';

    Stratified jackknife replication assumed

    Size of block   1  =              6

    Total size of tally matrix =      6

**** End of CREATE specification/beginning of execution
PRINT request    1
     cluster             1.0000          tenure              2.0000
PRINT request    1
     cluster             2.0000          tenure              2.0000
PRINT request    1
     cluster             3.0000          tenure              1.0000
PRINT request    1
     cluster             4.0000          tenure              2.0000
PRINT request    1
     cluster             5.0000          tenure              1.0000
PRINT request    1
     cluster             6.0000          tenure              1.0000

    End of primary input file after obs #       6

    End on unit 19 after obs #                  6

         3 strata observed on incoming file
```

Exhibit 4.3 An extract from i4-1.lis. Italics identify information provided by VPLX, including the file assignment to a unit number. After the first INPUT statement, which specifies how the primary file is to be read, a second file is linked. The subsequent INPUT statement reads 3 variables from the file, including 2 that are already defined and used as the key. Both files are in sort by the key variables. For each case on the primary file, there is a matching secondary case. VPLX reports reading 6 observations each from the primary and linked files, indicative of one-to-one matching.

The example illustrates several general features:

- For keyed linking, the key variables must already be defined before the input statement containing the /KEY specification. The key variables may have been read, although in some cases it may be necessary to compute them.

- The key variables in the primary and linked files must be in sort order. They must uniquely identify each record in the linked file, that is, there cannot be records with duplicate keys in the linked file.

C Other variables included on the INPUT statement become defined and may be used in the same manner as variables read from the primary input file.

The example does not illustrate another general rule for keyed linkage: the LINKED file may contain additional observations whose keys do not match the keys for the observations established by the primary file. These nonmatching records are ignored, except that they are included in the total number of observations read from the linked file when it is reported by VPLX. On the other hand, failure to obtain a matching record from the LINKed file is treated as a fatal error, unless an /INFLAG specification, to be explained in Section 4.4, is present in the INPUT statement.

4.3.3 Conditional matching with keyed LINK An INPUT statement for a LINKED file may be placed within an IF block, so that records will be read from the LINKED file only under the specified condition. Variables are assigned the default value of 0 for other observations, except for existing variables that have been previously defined. The following example builds upon the one shown in Exhibit 4.3.

```
create  in = i1-1.dat  out = vplx1.vpl ; ! Beginning of CREATE step

Note: Fortran unit 19 assigned to i4-1.dat

Note: Fortran unit 18 assigned to i4-2.dat

  input  rooms persons  weight cluster   ! input statement to read
          stratum /format (5F2.0) ;      ! data  ("section #1")

  link  i4-1.dat;                        ! linked file
  input stratum cluster tenure           ! variables to read from file
      / format (3f2.0)
      / key stratum cluster ;            ! key variables already defined

  if tenure == 1 then;
    link i4-2.dat;
    input cluster assessment mortgage    ! input will be executed
      / format (f2.0,2f7.0) /key cluster;! conditionally
  end if;

/* the file i4-2.dat contains
 3 100000  75000
 5 150000 100000
 6 100000      0
*/

  print cluster tenure assessment mortgage  ;
  block assessment mortgage / select if tenure .in. {1};

    Stratified jackknife replication assumed

    Size of block   1  =              3

    Total size of tally matrix =      3
```

```
**** End of CREATE specification/beginning of execution
PRINT request    1
     cluster             1.0000          tenure              2.0000
     assessment           .0000          mortgage             .0000
PRINT request    1
     cluster             2.0000          tenure              2.0000
     assessment           .0000          mortgage             .0000
PRINT request    1
     cluster             3.0000          tenure              1.0000
     assessment     100000.0000          mortgage         75000.0000
PRINT request    1
     cluster             4.0000          tenure              2.0000
     assessment           .0000          mortgage             .0000
PRINT request    1
     cluster             5.0000          tenure              1.0000
     assessment     150000.0000          mortgage        100000.0000
PRINT request    1
     cluster             6.0000          tenure              1.0000
     assessment     100000.0000          mortgage             .0000

    End of primary input file after obs #       6

    End on unit 19 after obs #                  3

         3 strata observed on incoming file
```

Exhibit 4.4 An extract from i4-2.lis. The program reads from i4-2.dat only if tenure is 1. If linkage had been attempted unconditionally instead, then an error would have occurred for the first observation when no matching observation from i4-2.dat is available.

A related example illustrates additional features of conditional input under a keyed link.

```
create  in = i1-1.dat  out = vplx1.vpl ; ! Beginning of CREATE step

Note: Fortran unit 19 assigned to i4-1.dat

Note: Fortran unit 18 assigned to i4-3.dat

  input  rooms persons  weight cluster   ! input statement to read
          stratum /format (5F2.0) ;      ! data  ("section #1")

  link  i4-1.dat;                        ! linked file
  input stratum cluster tenure           ! variables to read from file
      / format (3f2.0)
      / key stratum cluster ;            ! key variables already defined

  if tenure == 1 then;
    link i4-3.dat;
    input cluster assessment mortgage    ! input will be executed
      / format (f2.0,2f7.0) /key cluster;! conditionally
  end if;

/* the file i4-3.dat contains
 2  88000      0
 3 100000  75000
 5 150000 100000
 6 100000      0
```

```
*/

  print cluster tenure assessment mortgage  ;
  block assessment mortgage / select if tenure .in. {1};

    Stratified jackknife replication assumed

    Size of block   1  =             3

    Total size of tally matrix =      3

**** End of CREATE specification/beginning of execution
PRINT request    1
     cluster             1.0000          tenure              2.0000
     assessment           .0000          mortgage             .0000
PRINT request    1
     cluster             2.0000          tenure              2.0000
     assessment           .0000          mortgage             .0000
PRINT request    1
     cluster             3.0000          tenure              1.0000
     assessment     100000.0000          mortgage         75000.0000
PRINT request    1
     cluster             4.0000          tenure              2.0000
     assessment           .0000          mortgage             .0000
PRINT request    1
     cluster             5.0000          tenure              1.0000
     assessment     150000.0000          mortgage        100000.0000
PRINT request    1
     cluster             6.0000          tenure              1.0000
     assessment     100000.0000          mortgage             .0000
```

Exhibit 4.5 An extract from i4-3.lis. As in Exhibit 4.4, the program reads from i4-3.dat only if tenure is 1. Because tenure==2 for the second observation, the data present in i4-3.dat is not read at that point. When tenure == 1 for the third observation, the first record of i4-3.dat is skipped and the second observation, which has a matching key, is correctly linked. Note that this circumstance is not regarded as an error. A subsequent DISPLAY step, not shown, produces the same results as in Exhibit 4.4.

4.3.4 Many-to-one matching with keyed LINK If more than one observation read from the primary file results in the same matching key, then a many-to-one match occurs. There is no syntactic distinction required between many-to-one matching and one-to-one matching. In other words, the same VPLX commands will result in either one-to-one or many-to-one matching depending on the files. Although the following example does not illustrate this, many-to-one matching may occur conditionally, that is, the INPUT statement may occur within an IF block.

```
create  in = i1-1.dat  out = vplx1.vpl ; ! Beginning of CREATE step

Note: Fortran unit 19 assigned to i4-4.dat

  input  rooms persons  weight cluster   ! input statement to read
          stratum /format (5F2.0) ;      ! data  ("section #1")

  link  i4-4.dat;                        ! linked file
```

I.4.14

```
  input stratum weight                    ! many-to-one match
      / format(f2.0,f8.2)/key stratum ;   ! by stratum
  weight = 2*weight;

/* the file i4-4.dat is
 1 1000.00
 2 1500.00
 3 1000.00
*/

  print cluster weight  ;

    Stratified jackknife replication assumed

    Size of block   1  =              3

    Total size of tally matrix =      3

**** End of CREATE specification/beginning of execution
PRINT request    1
     cluster             1.0000          weight           2000.0000
PRINT request    1
     cluster             2.0000          weight           2000.0000
PRINT request    1
     cluster             3.0000          weight           3000.0000
PRINT request    1
     cluster             4.0000          weight           3000.0000
PRINT request    1
     cluster             5.0000          weight           2000.0000
PRINT request    1
     cluster             6.0000          weight           2000.0000

    End of primary input file after obs #       6

    End on unit 19 after obs #                  3

         3 strata observed on incoming file
```

Exhibit 4.6 An extract from i4-4.lis. A file of stratum level weights is linked. In this case, the weight on the second file replaces the value from the first. Each observation receives the weight carried on the second file, which is then doubled for purposes of illustration. For each observation, the input statement provides the values as they were read from the linked file, regardless of recoding that occurred with previous observations. Consequently, the final value of weight is the same for each observation within a stratum.

4.3.5 LINK without /KEY If no /KEY specification is included in the INPUT statement, then an observation is read from the INPUT file each time the INPUT statement is executed. No check is performed to confirm that the linked record matches. An error occurs, however, if the linked file ends prematurely.

In some applications, it may be simpler to perform this form of linkage rather than to establish keys. If, however, for whatever reason, two files become sorted into different orders prior to their input

to VPLX, a keyed link allows VPLX to detect this error, while an unkeyed link generally does not. The following example illustrates an unkeyed link with the intended result.

```
create  in = i1-1.dat  out = vplx1.vpl ; ! Beginning of CREATE step

Note: Fortran unit 19 assigned to i4-1.dat

  input  rooms persons  weight cluster   ! input statement to read
          stratum /format (5F2.0) ;      ! data  ("section #1")

  link  i4-1.dat;                        ! linked file
  input stratum cluster tenure           ! variables to read from file
      / format (3f2.0);

  print cluster tenure ;
  class tenure (2/1,3) 'Renter' 'Owner or other';


    Stratified jackknife replication assumed

    Size of block   1  =              6

    Total size of tally matrix =      6

**** End of CREATE specification/beginning of execution
PRINT request    1
     cluster             1.0000          tenure              2.0000
   ...
```

Exhibit 4.7 An extract from i4-5.lis. The example follows the one in Exhibit 4.3 closely. Without a /KEY specification in the INPUT statement for the linked file, records are read without checking. The remaining results, including the remaining output from the PRINT statements and the DISPLAY step, are the same as in Exhibit 4.3 and i4-1.lis.

4.4 The /INFLAG specification

The /INFLAG specification in the INPUT statement associated with a linked file names a variable to indicate whether a record has been successfully linked (=1) or not (=0). For example,

```
link   testfile1.dat ;
input  rooms persons  weight cluster stratum
       /format (5F2.0) /options nprint=1 nobs=6
       /key stratum cluster /inflag flag1 ;
```

illustrates all available features of an INPUT statement for a linked file. When an observation is matched on the basis of stratum and cluster to testfile1.dat, flag1 is set to 1. If the observation is not linked, then all variables except the key variables, that is, rooms, persons, weight, and flag1 in this example, are all set to 0. It is then possible to specify, on the basis of flag1, some additional action, such as reading the data from another file or excluding the observation from a block. Without /INFLAG, VPLX would terminate the step in error.

I.4.16

The following example illustrates use of /INFLAG with a keyed link.

```
create  in = i1-1.dat  out = vplx1.vpl ; ! Beginning of CREATE step

Note: Fortran unit 19 assigned to i4-1.dat

Note: Fortran unit 18 assigned to i4-2.dat

Note: Fortran unit  9 assigned to i4-6.dat

  input  rooms persons  weight cluster   ! input statement to read
          stratum /format (5F2.0) ;      ! data  ("section #1")

  link  i4-1.dat;                        ! linked file
  input stratum cluster tenure           ! variables to read from file
      / format (3f2.0)
      / key stratum cluster ;            ! key variables already defined
  link i4-2.dat;
  input cluster assessment mortgage      ! input will be executed
      / format (f2.0,2f7.0) /key cluster ! unconditionally
      / inflag flag4_2 ;                 ! flag for i4-2.dat
/* the file i4-2.dat contains
 3 100000  75000
 5 150000 100000
 6 100000      0
*/
  if .not. flag4_2 then ;                !  or, "if flag4_2 == 0 then ;"
     link i4-6.dat ;
     input cluster rent
           / format (f2.0,f5.0)
           / key cluster
           / inflag flag4_6 ;            ! flag for i4-6.dat
/* the file i4-6.dat contains
 1  675
 4  490
*/
     if flag4_6 == 0 then;               ! print for observations
        print cluster ;                  ! not linked to either file
     end if;
  end if;

  print cluster tenure assessment mortgage rent  ;
  block assessment mortgage / select if tenure .in. {1};
  block rent / select if flag4_6 .in. {1} ;

    Stratified jackknife replication assumed

    Size of block   1  =             3

    Size of block   2  =             2

    Total size of tally matrix =     5

**** End of CREATE specification/beginning of execution
PRINT request    2
     cluster             1.0000          tenure             2.0000
     assessment           .0000          mortgage            .0000
     rent              675.0000
```

```
PRINT request    1
     cluster             2.0000
PRINT request    2
     cluster             2.0000          tenure              2.0000
     assessment           .0000          mortgage             .0000
     rent                 .0000
PRINT request    2
     cluster             3.0000          tenure              1.0000
     assessment     100000.0000          mortgage         75000.0000
     rent                 .0000
PRINT request    2
     cluster             4.0000          tenure              2.0000
     assessment           .0000          mortgage             .0000
     rent              490.0000
PRINT request    2
     cluster             5.0000          tenure              1.0000
     assessment     150000.0000          mortgage        100000.0000
     rent                 .0000
PRINT request    2
     cluster             6.0000          tenure              1.0000
     assessment     100000.0000          mortgage             .0000
     rent                 .0000

    End of primary input file after obs #       6

    End on unit 19 after obs #                  6

    End on unit 18 after obs #                  3

    End on unit  9 after obs #                  2

         3 strata observed on incoming file


display

list assessment mortgage rent

                                                  Estimate      Standard error

assessment              :  MEAN                 116666.6667          17921.5110

mortgage                :  MEAN                  58333.3333          33978.1384

rent                    :  MEAN                    582.5000             97.5036
```

Exhibit 4.8 An extract from i4-6.lis. The example follows the one in Exhibit 4.3 closely. The /INFLAG specifications allow files to be read without matching records. Observation 2 remains unmatched, which is indicated by the resulting print request 1 output.

When /INFLAG is used without /KEY, the /INFLAG variable is set to 1 until an end of file occurs on the linked file, in which case it is set to 0.

The /INFLAG feature is new to version 1997.12. Previously, the only means to achieve a similar effect was to employ LINK_MISSING in place of LINK. Generally, /INFLAG allows greater programming flexibility and avoids complex features of *real with missing* variables.

4.5 Programming Strategies for Complex Data

4.5.1 The Role of the Primary Input File The various forms of linking permit the combining of data from different files in a variety of situations. Nonetheless, the primary (IN=) input file has the unique function of establishing the potential observations for analysis. Although observations can be dropped from the output VPLX file with /SELECT in the BLOCK statements, observations must appear in the primary file in order to be included in the analysis.

This section discusses three familiar situations in which the requirement that the primary file includes each observation necessitates special programming approaches. In each case, there are typically two general strategies. The first is to use a single CREATE step and to design the primary file to include each observation. The second is to divide the problem into separate pieces with more than one CREATE step and then to integrate the resulting VPLX files with other parts of the VPLX system, specifically, either the VPLXAPPEND feature of the TRANSFORM step (Section xx) or the MERGE step (Section xx). The situations to be considered are:

- Hierarchical files, such as demographic surveys with household- and person-level data,
- Longitudinal files, where most observations may have data from two or more time periods, although some observations may be present at only one time period,
- A time series of cross-sectional files, where one may be interested in aggregate statistics for two or more time periods without requiring observation-level linking. For example, the Current Population Survey (CPS) measures labor force participation and unemployment monthly with a rotating panel design, but most analyses of this survey do not require person-level linkage across months.

In each case, both strategies will be discussed.

4.5.2 Hierarchical Files The situation of persons within housing units illustrates issues that arise in processing hierarchical data. If the analysis is only of occupied housing units, then a primary file of person-level data may be linked with a household identifier as a key to a household-level file with geographic and other data. Characteristics of households may be estimated by establishing blocks using a /SELECT identifying the first person in the household, perhaps by using the IDCHANGE function (Section 2.7.2). (It is important, of course, to avoid including household-level characteristics for every person.) In some applications, a primary person, not necessarily the first in the household, may provide the household weight, so that blocks of household-level data should be formed using /SELECT to identify primary persons.

The situation is more complex if vacant units are to be included as well. In SAS and other languages, it is natural to establish a housing unit file including the vacants, with occupied housing units linked to a person-level file. This design does not conform to the requirement that each observation be represented on the primary file, however, since vacant units would not have corresponding persons in the person-level file.

To handle this situation in a single CREATE step, it is necessary to prepare an augmented person-level file by adding a person-level observation for each vacant unit. Naturally, it is necessary to clearly distinguish the added records for vacant units from real person-level data. A specific variable on the person-level file may be created to identify the added records pointing to vacant units. Alternatively, a person-level identifier, such as person number, may be set to 0 to identify these observations. Again, separate blocks should be established to distinguish variables defined at the levels of person and housing unit.

```
create  in = person1.dat           ! augmented person file with
                                   ! added records for vacants
        out = vplx1.vpl ;

     input hhid personid .... ;    ! person records contain both
                                   ! household and person ids

     person1 = idchange (hhid) ;   ! identify first person in each hhld

     link hhld1.dat ;              ! household file with data for both
                                   ! occupied and vacants
     input hhid ...   / key hhid ; ! match on hhid

     block ....                    ! block for housing data
       /select if person1 .in. {1};

     block ...                     ! block for person data
       /select if                  ! excluding dummy records from
          personid .in. {1-high};  ! person1.dat with personid == 0
```

Exhibit 4.9 Outline of a VPLX CREATE step to link housing and person-level data. In this example, an augmented person1.dat contains dummy records with personid == 0 to link to vacant units on the housing file. Blocks containing person data must explicitly exclude the dummy records.

The alternative in this situation is to employ two CREATE steps. One uses the person-level file as the primary file and links to the housing file to define person-level variables. The second uses the housing unit file as the primary file and obtains housing-level data for both occupied and vacant units. The resulting two VPLX files may be then linked through VPLXAPPEND in the TRANSFORM step, providing a VPLX file with results equivalent to the strategy based on a single CREATE step.

I.4.20

```
create  in = person.dat            ! person file with actual persons only
        out = vplx1.vpl ;

     input hhid personid .... ;    ! person records contain household id
                                   ! and, optionally, a personid

     link hhld1.dat ;              ! housing unit file with data for both
                                   ! occupied and vacants
     input hhid ...   / key hhid ; ! match on hhid - vacant hhlds will
                                   ! be skipped over
     block ...                     ! block for person data

create  in = hhld1.dat             ! housing unit file with data for both
                                   ! occupied and vacants
        out = vplx2.vpl ;

     input hhid ... ;

     block ...      ;              ! block for housing data

transform  in = vplx1.vpl          ! transform to create combined
           out = vplx3.vpl ;       ! file – vplx3.vpl

vplxappend vplx2.vpl
```

Exhibit 4.10 Outline of VPLX steps to read person and housing unit data. The first CREATE step creates person-level variables only, linking in housing unit records to obtain any data available at the housing unit level that might be necessary, such as geographic variables. The second CREATE step defines housing variables. Variables from the two files may be combined in a subsequent TRANSFORM step.

4.5.3 Longitudinal Files The particular strength of longitudinal surveys is to provide information on change at the observation level. In some applications, such as the SIPP example introduced in Chapter 1, the work of linking information together has already be done, and a single file is available as the primary file. In other cases, information may be available on separate cross-sectional files, and the user must decide whether to link the information in VPLX or to build a single longitudinal file before submitting the data to VPLX. The SIPP illustrates these circumstances, because cross-sectional files are typically released before longitudinal files *(7)*.

If the longitudinal interest is only in persons included in the first time period, then the file for the first period may serve as the primary input file. Files for subsequent periods may then be linked, perhaps using /INFLAG variables to identify attrition or persons who fall out of scope.

If the analytic situation is more complex, for example, if analytic interest includes persons who enter the universe subsequent to the first period, then it may be advantageous to build a master file of survey identifiers. The master file may be used as a primary file and characteristic data read from linked period files as appropriate.

```
create  in = master.dat            ! master file of all possible observations
        out = vplx1.vpl ;

     input hhid personid ;         ! ids of all possible observations

     link period1.dat;             ! data from period 1
     input hhid personid age1 ...
        /key hhid personid
        /inflag period1 ;          ! flag records if data present

     link period2.dat;             ! data from period 2
     input hhid personid age2 ...
        /key hhid personid
        /inflag period2 ;

     ...

     if period1 == 1 .and.         ! define variables requiring data
        period2 == 1 then ;        ! in first two periods

     ...

     end if;

     ...                           ! coding to define other variables
                                   ! such as observations in period 1 only

     block ....                    ! block statement for variables
        / select if period1        ! requiring data in first two
           .in. {1}                ! periods
        / select if period2
           .in. {1}

    ...                            ! block statements for other combinations
                                   ! such as observations in period 1 only
```

Exhibit 4.11 Outline of a VPLX CREATE step to read longitudinal data from multiple files. The master file must include the identifiers of all possible observations. The example illustrates how variables requiring data in the first two periods may be defined. This programming strategy may be modified for any other combination, such as observed in the first period, observed in the first period only, observed in the first three periods, *etc.*, and separate blocks created for each required combination.

An alternative may be to perform a series of CREATE steps, each of which uses a period file as the primary file. This process could begin by using the first period file as the primary file. The next CREATE step could use the second period file as the primary file, link back to the first period file to identify, with /INFLAG, persons making their first appearance in the second period, and forming blocks with characteristics of such persons *(8)*. Depending on the application, it may be more effective to VPLXAPPEND the resulting VPLX files or to MERGE them.

```
create  in = period1.dat;             ! data from period 1
        out = vplx1.vpl ;
     input hhid personid age1 ... ;   ! data from period 1

     link period2.dat;                ! data from period 2
     input hhid personid age2 ...
        /key hhid personid
        /inflag period2 ;

     ...

     if period2 == 1 then ;         ! define variables requiring data
                                    ! in first two periods

     ...

     end if;

     block ....                     ! block statement for variables
        / select if period2         ! requiring data in first two
           .in. {1}                 ! periods

    ...     other CREATE steps to handle other conditions

    transform in = vplx1.vpl        ! step to combine different
              out = vplxcomb.vpl ; ! files

    vplxappend vplx2.vpl

    vplxappend vplx3.vpl

    ...
```

Exhibit 4.12 Outline of a VPLX application to read longitudinal data from multiple files. The first CREATE step illustrates how variables requiring data in the first two periods may be defined. Other CREATE steps can be added for any other combination, such as observed in the first period, observed in the first period only, observed in the first three periods, *etc.*, and separate VPLX files created for each required combination. A final TRANSFORM step may be used to integrate the VPLX files together.

4.5.4 Time Series of Cross-Sectional Files The central focus of the CPS and some other demographic surveys, as well as that of almost all economic surveys, is in aggregate trends over time, and the linkage of information at the observation level is typically not required. In this instance, it is possible to consider forming a master file to govern the linkage of information in a single CREATE step. If a replication option that determines the number of replicates after inspection of the input file, such as the jackknife or stratified jackknife, is used, then this strategy may be necessary to assure a consistent assignment of replicates *(9)*.

```
create  in = master.dat            ! master file of all possible observations
        out = vplx1.vpl ;

     input hhid personid ;         ! ids of all possible observations

     link period1.dat;             ! data from period 1
     input hhid personid age1 ...
        /key hhid personid
        /inflag period1 ;          ! flag records if data present

     link period2.dat;             ! data from period 2
     input hhid personid age2 ...
        /key hhid personid
        /inflag period2 ;

     ...



     block ....                    ! block statement for 1st period
        / select if period1
           .in. {1}
     block ....                    ! block statement for 2nd period
        / select if period2
           .in. {1}
```

Exhibit 4.13 Outline of a VPLX CREATE step to read cross-sectional data from multiple files. The master file must include the identifiers of all possible observations. A separate block is established for each period. The MERGE step provides a more convenient alternative, however.

For this application, however, it is usually likely to be far more effective to process data for each time period separately, yielding a series of VPLX files, which then may be merged. Section xx describes this strategy in more detail.

NOTES

1. Until recently, input lists could only be comprised of new variables. This restriction resulted in circuitous programming accommodations and is now removed.

2. This limit is on the number of characters of the specified format itself, not the total number of characters in the input records that the format describes. Depending on the application, a 50-character format may describe thousands of character positions on the IN= data file. If the constraint proves a hindrance, however, it is possible to increase the maximum length by changing a parameter in the Fortran source.

3. Specifically, data sets at the observation level rarely use this feature. The format is frequently useful, however, for derived statistics. For example, VPLX outputs estimated covariance matrices using this format.

4. As previously described, parenthetical expressions may be preceded by a multiplier, such as 13(/,10f10.4) within a format specification. After the format has been fully used once, Fortran returns to the last parenthetical expression without a multiplier. The initial parenthesis beginning a format is included in this rule, so that Fortran returns to the beginning of the format if there are no other parenthetical expressions or if each parenthetical expression (except the entire format) is preceded by a multiplier, as in the format (4f3.0,13(/,10f10.4)).

   Because these rules are complex and not fully stated here, it is recommended that this feature of Fortran not be used in conjunction with VPLX.

5. The author's recollection was that SAS wrote the file, perhaps in a format such as 10.4. When the decimal fraction rounded to .0000, only the integer part, without the decimal point, was written to the file. In other words, instead of, for example, " 1917.0000", the value was expressed as " 1917". VPLX consequently read this number as 10^{-4} times its intended value. The problem was isolated with some effort and corrected by using F10.0, but the case was regrettably not carefully documented at the time. The author has been unable to reproduce a subsequent example from SAS, so that another system, such as a data base, may have caused the situation. In any case, the principle of using F*w.*0 as a defensive measure is still applicable.

6. When LINK_MISSING appears, all variables except KEY variables on the INPUT list become real with missing variables, rather than real when LINK is used. Real with missing variables are treated as an advanced feature because they have more complex properties than real variables and because there are generally VPLX programming strategies to accomplish the same results with real variables. In particular, the recent addition of the /INFLAG specification enables keyed linking with LINK in situations that previously required LINK_MISSING.

7. Cross-sectional files from the first few waves of a SIPP panel may become available far in advance of the longitudinal file. The longitudinal file includes enhancements not available from the cross-sectional files, including reediting and imputation of characteristics longitudinally and provision of longitudinal weights. Nonetheless, it is the author's understanding that a number of users may analyze the cross-sectional files longitudinally on the basis of timeliness.

8. In other words, the function of the second CREATE step is to pick up data for persons absent from the first period. When the second period file is used as the primary file, observations linking back to the first period file (indicated when the /INFLAG variable is set to 1) should be dropped, and only those not matching retained. Similar strategies can then be applied for the third period, *etc.*

9. For instance, VPLX established the number of replicates in the examples of this chapter on the basis of what was observed on the input file. If separate CREATE steps are run on each period of data separately, then the number of replicates may differ from one period to the next. Volume II, Section xx, discusses this situation in more detail.