Format 3

A Format 3 PERFORM statement executes a statement-list, iterated within the bounds of a PERFORM and END-PERFORM statement, once, or multiple times as designated by the FOREVER phrase or the TIMES phrase.

```
PERFORM [ { FOREVER    } ]
          {numeric-1 TIMES }
        [ statement-list ]
        [ END-PERFORM ]
```

Syntax

1. numeric-n is a numeric literal or data item.
2. statement-list is a list of imperative and/or conditional statements.

General Rules

1. The scope of the Format 3 PERFORM statement is all of the statements in statement-list, which are entered between the PERFORM and END-PERFORM statements.
2. The TIMES phrase indicates that the number of times that statements within the scope of the PERFORM statement are to be executed. After the statements within the scope of the PERFORM statement have been executed the designated number of TIMES, control returns to the next statement in the program after the PERFORM statement.
3. The FOREVER phrase indicates that the statements within the scope of the PERFORM statement should continue to execute continuously. A program may exit from a PERFORM FOREVER phrase with an EXIT PERFORM CYCLE statement.

Format 4

A Format 4 PERFORM statement executes a statement-list, iterated within the bounds of a PERFORM and END-PERFORM statement UNTIL a named condition tests true.

```
PERFORM
        [WITH TEST {BEFORE}]
                   {AFTER }
         VARYING { num-1 FROM num-2 by num-3 } ] UNTIL condition-1
        [ statement-list ]
        [ END-PERFORM ]
```

Syntax

1. num-n is a numeric literal or data item.
2. condition-1 is a conditional statement.
3. statement-list is a list of imperative and/or conditional statements.

General Rules

1. The scope of the Format 4 PERFORM statement is all of the statements in statement-list, which are entered between the PERFORM and END-PERFORM statements.
2. Condition-1 is the condition-test that determines whether or not to perform the statements within the scope of the PERFORM statement.
3. Condition-1 may contain the word EXIT. This is a PERFORM UNTIL EXIT statement, and the condition tests true when an EXIT PERFORM CYCLE statement is reached.
4. In a Format 4 PERFORM statement, the WITH TEST phrase is optional. In the absence of a WITH TEST phrase, WITH TEST BEFORE is assumed.
5. The WITH TEST BEFORE phrase indicates that the test of condition-1 takes place before executing the statements within the scope of the PERFORM statement. If condition-1 tests TRUE initially, the statements within the scope of the PERFORM statement are not executed, and control is passed to the next statement after the PERFORM statement.
6. The WITH TEST AFTER phrase indicates that the test of condition-1 takes place after executing the statements within the scope of the PERFORM statement. When the WITH TEST AFTER phrase is used, the statements within the scope of the PERFORM statement will be executed at least one time.
7. The VARYING phrase is used to increment, or decrement a numeric counter variable by a fixed amount. When using the VARYING phrase , condition-1 is set to a test of the numeric counter variable.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. PERFORM-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 CTR1    PIC 9(4).
       77 CTR2    PIC 9(4).
       77 CTR3    PIC 9(4).
       77 CTR4    PIC 9(4).
       77 COUNTER PIC 9(4).
       77 DUMMY   PIC X.
      *
       PROCEDURE DIVISION.
       MAIN.

           PERFORM PARA-1 WITH TEST BEFORE
             VARYING CTR1 FROM 1 BY 1 UNTIL CTR1 > 2.

           PERFORM PARA-1 THRU PARA-1-END WITH TEST AFTER
             VARYING CTR2 FROM 10 BY 10 UNTIL CTR2 > 20.

           MOVE 1 TO COUNTER.
           PERFORM FOREVER
             ADD 1 TO COUNTER
             IF COUNTER > 2
               EXIT PERFORM
```

```
            END-IF
          END-PERFORM.


          MOVE 1 TO COUNTER.
          PERFORM 1 TIMES
            ADD 1 TO COUNTER
            IF COUNTER > 2
              EXIT PERFORM
            END-IF
          END-PERFORM.

          MOVE 1 TO COUNTER.
          PERFORM WITH TEST BEFORE
            VARYING CTR3 FROM 1 BY 1 UNTIL CTR3 > 2
            ADD 1 TO COUNTER
            IF COUNTER > 2
              EXIT PERFORM CYCLE
            END-IF
          END-PERFORM.

          PERFORM WITH TEST AFTER
            VARYING CTR4 FROM 1 BY 1 UNTIL CTR4 > 2
            ADD 1 TO COUNTER
            IF COUNTER > 2
              EXIT PERFORM CYCLE
            END-IF
          END-PERFORM.


          PERFORM PARA-1 2 TIMES.

          PERFORM PARA-1 THRU PARA-1-END 2 TIMES.

          PERFORM PARA-2 THRU PARA-2-END FOREVER.
      *
       PARA-1.

          CONTINUE.
       PARA-1-END.
          EXIT.
       PARA-2.
          ADD 1 TO COUNTER.
          IF COUNTER >5
            DISPLAY "PERFORM-1 FINISHED!" LINE 15 COL 10
            ACCEPT DUMMY LINE 15 COL 30
            STOP RUN.
       PARA-2-END.
          EXIT.
```

5.3.31 PRAGMA Statement

The PRAGMA statement provides internal compiler control, for profiling. The first literal is the command sent to the compiler.

Format 1

```
PRAGMA  "PROFILING" { literal-1 } ...
```

Format 2

```
PRAGMA  "DUMP" { literal-1 } ...
```

Syntax

1. literal-n is a character string.

General Rules

1. The PRAGMA "PROFILING" statement causes programs compiled with –fprofiling to report time measurements from the current PRAGMA statement to the next PRAGMA , or to the end of the program.
2. The PRAGMA "PROFILING" statement should be entered in column 8, as in the Code Sample below.
3. The PRAGMA "DUMP" statement causes a profiling report to be generated when executed. Profiling reports produced by the PRAGMA "DUMP" statement will overwrite previous PRAGMA "DUMP" files with the same name.

PRAGMA "PROFILING" Code Sample

```
       PROCEDURE DIVISION.
       . . .
       PRAGMA "PROFILING" "STEP1".
       . . .
       PRAGMA "PROFILING" "STEP2".
```

PRAGMA "DUMP" Code Sample

```
       PROCEDURE DIVISION.
       . . .
       PRAGMA "DUMP" "REPORT".
       . . .
```

5.3.32 READ Statement

The READ statement retrieves records from files.

Format 1
A Format 1 Read is a Sequential READ, which retrieves the NEXT or PREVIOUS record, as determined by the current file pointer.

```
READ file-1     {NEXT     } RECORD INTO data-1
                {PREVIOUS}
                [{IGNORING LOCK     }]
                 {WITH LOCK         }
                 {WITH NO LOCK      }
                 {WITH IGNORE LOCK  }
                 {WITH WAIT         }
```

```
                    {WITH KEPT LOCK   }
                  [ AT END statement-1 ]
                  [ NOT AT END statement-2 ]
                  [ END-READ ]
```

1. file-n is a file described in the File Section with an FD.
2. data-n is a data item.
3. key-name-n is a data element that has been named as a key for an indexed file.
4. statement-n is an imperative statement.

General Rules

1. file-1 must be OPEN Input or I-O before the READ statement is executed.
2. The Format 1 READ statement applies to files declared with ACCESS MODE IS SEQUENTIAL or ACCESS MODE IS DYNAMIC.
3. A successful READ statement retrieves a record from an indexed, relative, or sequential file, and stores the record in the data area described in the File Description (FD) in the File Section of the program.
4. The NEXT and PREVIOUS phrases indicate in which direction the READ is to move the file pointer in the sequential READ operation.
5. The file pointer is initially placed at the beginning of the file after a successful OPEN statement.
6. In files described with ORGANIZATION IS SEQUENTIAL, a READ NEXT statement after an OPEN statement retrieves the first record in the file.
7. In files described with ORGANIZATION IS SEQUENTIAL, the READ NEXT and READ PREVIOUS statements retrieve the next/previous physical record in the file after/before the current position of the file pointer.
8. In files described with ORGANIZATION IS RELATIVE, the READ NEXT statement retrieves the next physical record in the file after the position marked by the RELATIVE KEY, and increments the relative-key data field. The READ PREVIOUS statement retrieves the previous physical record in the file after the position marked by the RELATIVE KEY and decrements the relative-key data field.
9. In files described with ORGANIZATION IS INDEXED, the READ NEXT and READ PREVIOUS statements retrieve the next/previous physical record in the file after/before the current position of the file pointer.
10. The file pointer may be placed anywhere in the file with the START statement. For details, see START statement.
11. The INTO phrase causes a MOVE to data-1 after the data has been retrieved in the record area of the File Description (FD).
12. The IGNORING LOCK phrase ignores locks held by other programs.
13. IGNORING LOCK and WITH IGNORE LOCK are synonyms.
14. The WITH LOCK phrase causes the record to be locked. A locked record cannot be READ WITH LOCK or updated by another program.
15. WITH LOCK and WITH KEPT LOCK are synonyms.
16. The WITH NO LOCK phrase allows access to the record by other programs.
17. The WITH IGNORE LOCK phrase ignores locks held by other programs.
18. The WITH WAIT phrase causes the READ statement to wait for a lock held on record to be released.

19. The AT END condition is triggered when the end of the file is reached in a sequence of READ NEXT statements, or when the beginning of the file is reached in a sequence of READ PREVIOUS statements.
20. If the AT END condition is triggered, and the AT END phrase is included in the READ statement, the FILE STATUS variable is set to "10", the statement-list within the scope of the AT END phrase is executed, and control is then passed to the next statement after the READ.
21. The NOT AT END condition is triggered when the sequential READ is successful. When the NOT AT END phrase is included in the READ statement, and the READ is successful, the statement-list within the scope of the NOT AT END phrase is executed, and control is then passed to the next statement after the READ.

Format 2

A Format 2 READ is a KEY'ed READ, which READs on the relative key of a relative file, or one of the record keys of an indexed file.

```
READ file-1     RECORD INTO data-1
                [{IGNORING LOCK    }]
                 {WITH LOCK        }
                 {WITH NO LOCK     }
                 {WITH IGNORE LOCK }
                 {WITH WAIT        }
                 {WITH KEPT LOCK   }
                 [ KEY IS { data-2 } ]
                          { key-name-1 } ]
                 [ INVALID KEY statement-3 ]
                 [ NOT INVALID KEY statement-4 ]
                 [ END-READ ]
```

Syntax

1. file-n is a file described in the File Section with an FD.
2. data-n is a data item.
3. key-name-n is a data element that has been named as a key for an indexed file.
4. statement-n is an imperative statement.

General Rules

1. file-1 must be OPEN Input or I-O before the READ statement is executed.
2. The Format 2 READ statement applies to files declared with ACCESS MODE IS RANDOM or ACCESS MODE IS DYNAMIC.
3. The INTO phrase causes a MOVE to data-1 after the data has been retrieved in the record area of the File Description (FD).
4. The KEY clause is optional.
5. If the KEY clause is not present, then:
    a. In relative files, the value of the relative key field is used as the key
    b. In indexed files, the current value of the primary key stored in the record in the File

Sectio is used as the key.

6. The KEY clause, when used, references keys as follows:
   a. In relative files, the value of the data field referenced is used as the relative key.
   b. In indexed files, the data field referenced must identify either the primary key or one of the alternate record keys.
7. A successful READ statement retrieves a record from an indexed, or relative file, and stores the record in the data area described in the File Description (FD) in the File Section of the program.
8. When a READ statement is made using an alternate key that allows duplicates, the first record that matches the given key is retrieved.
9. The INTO phrase causes a MOVE to data-1 after the data has been retrieved in the record area of the File Description (FD).
10. The WITH LOCK phrase causes the record to be locked. A locked record cannot be READ WITH LOCK or updated by another program.
11. WITH LOCK and WITH KEPT LOCK are synonyms.
12. The IGNORING LOCK phrase ignores locks held by other programs.
13. IGNORING LOCK and WITH IGNORE LOCK are synonyms.
14. The WITH NO LOCK phrase allows access to the record by other programs.
15. The WITH WAIT phrase causes the READ statement to wait for a lock held on record to be released.
16. The INVALID KEY condition is triggered in a READ on a relative file when a RELATIVE KEY data field does not contain a positive integer value.
17. The INVALID KEY condition is triggered in a READ on an indexed file when the RECORD KEY is not found.
18. When the INVALID KEY condition is triggered, and the INVALID KEY clause is included in the READ statement, the FILE STATUS variable is set to “23”, the statement-list within the scope of the INVALID KEY phrase is executed, and control is then passed to the next statement after the READ.
19. The NOT INVALID KEY condition is triggered when the READ is successful. When the NOT INVALID KEY phrase is included in the READ statement, and the READ is successful, the statement-list within the scope of the NOT INVALID KEY phrase is executed, and control is then passed to the next statement after the READ.
20. The READ statement updates the FILE STATUS variable.
21. If a READ statement is unsuccessful, the file pointer is undefined.

Code Sample

```
...
       SELECT SEQ-FILE-1 ASSIGNTO"SEQ-FILE-1"
       ORGANIZATION IS SEQUENTIAL
       ACCESS IS SEQUENTIAL
       FILE STATUS IS SEQ-FILE-1-STAT.
...
   FD SEQ-FILE-1.
   01 SEQ-FILE-1-RECORD    PIC X(10).
...
   77 SEQ-FILE-1-STATPIC XX.
   01 WS-SEQ-FILEPIC X(10).
...
       READ SEQ-FILE-1 NEXT RECORD.
       READ SEQ-FILE-1 NEXT RECORD INTO WS-SEQ-FILE.
       READ SEQ-FILE-1 NEXT RECORD IGNORING LOCK.
       READ SEQ-FILE-1 NEXT RECORD WITH LOCK.
```

```
           READ SEQ-FILE-1 NEXT RECORD WITH NO LOCK.
           READ SEQ-FILE-1 NEXT RECORD WITH IGNORE LOCK.
           READ SEQ-FILE-1 NEXT RECORD WITH WAIT.
           READ SEQ-FILE-1 NEXT RECORD WITH KEPT LOCK.
           READ SEQ-FILE-1 NEXT RECORD
             AT END MOVE "10" TO SEQ-FILE-1-STAT
             NOT AT END
               CONTINUE
           END-READ.

           SELECT IDX-FILE-1 ASSIGN TO "IDX-FILE-1"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS IDX-KEY
           FILE STATUS IS IDX-FILE-1-STAT.


       FD IDX-FILE-1.
       01 IDX-FILE-1-RECORD.
         03 IDX-KEY PIC X(10).

       77 IDX-FILE-1-STAT PIC XX.
       01 WS-IDX-FILE PIC X(10).

     ...
           READ IDX-FILE-1 NEXT RECORD.
           READ IDX-FILE-1 NEXT RECORD INTO WS-IDX-FILE.
           READ IDX-FILE-1 PREVIOUS RECORD.
           READ IDX-FILE-1 IGNORING LOCK.
           READ IDX-FILE-1 WITH LOCK.
           READ IDX-FILE-1 WITH NOLOCK.
           READ IDX-FILE-1 WITH IGNORE LOCK.
           READ IDX-FILE-1 WITH WAIT.
           READ IDX-FILE-1 WITH KEPT LOCK.
           READ IDX-FILE-1
             INVALID KEY MOVE "23" TO IDX-FILE-1-STAT
             NOT INVALID KEY
               CONTINUE
           END-READ.
```

5.3.33 READY/RESET TRACE Statements

The READY/RESET TRACE statements enable paragraph tracing in a program compiled with the -fready-trace compiler flag.

General Format

```
READY TRACE
. . .
RESET TRACE
```

General Rules

1. The READY TRACE and RESET TRACE statements enable paragraph tracing and are otherwise ignored in the execution of the program.

2. Paragraphs/Sections entered in the interval between the READY TRACE and RESET TRACE statement are traced by writing output to the console in the form:

   PROGRAM-ID: [program-id]: [paragraph name]

   In the example below:
   PROGRAM-ID: READYTRACE: INITIALIZATIONS
   PROGRAM-ID: READYTRACE: MAIN-BODY

   Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. READYTRACE.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       PROCEDURE DIVISION.
       MAIN.
           READY TRACE.
           PERFORM INITIALIZATIONS.
           PERFORM MAIN-BODY.
           RESET TRACE.
           STOP RUN.
           ….
```

5.3.34 RELEASE Statement

The RELEASE statement makes records available for a SORT in an INPUT PROCEDURE.

General Format

```
RELEASE sort-record-1 [ FROM identifier-1 ]
```

Syntax

1. sort-record-n is a record described in a Sort Description (SD) in the File Section.
2. identifier-n is a data element, literal, or data returned from a function call.

General Rules

3. The RELEASE statement can only be used in the INPUT PROCEDURE of a SORT statement.
4. The RELEASE statement MOVEs data from identifier-1 to a sort-record, which is declared in a Sort Description (SD) in the FILE SECTION of the program.
5. The RELEASE statement adds the data MOVE'd from identifier-1 to the SORT file.

   Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. RELEASE-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
      *SORT INPUT FILE
           SELECT MEMBER-FILE ASSIGN TO "MEMBERSHIP.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERSHIP-STAT.

      *SORT OUTPUT FILE
           SELECT MEMBER-LIST ASSIGN TO "MEMBERLIST.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERLIST-STAT.

      *SORTFILE (SD)
           SELECT MEMBER-SORT ASSIGN TO "SORT-WORK".

       DATA DIVISION.
       FILE SECTION.
       FD MEMBER-FILE.
       01 MEMBER-INFO PIC X(40).

       FD MEMBER-LIST.
       01 SORTED-MEMBER-INFOPIC X(40).

       SD MEMBER-SORT.
       01 SORT-DATA.
         05 MEMBER-FIRST-NAME PIC X(10).
         05 MEMBER-LAST-NAME  PIC X(20).
         05 YEAR-JOINED       PIC X(4).
         05 MEMBER-RANK       PIC X(6).

       WORKING-STORAGE SECTION.
       77 MEMBERLIST-STAT PIC XX.
         88 EOF-MEMBERLIST  VALUE "10".
       77 MEMBERSHIP-STAT PIC XX.
         88 EOF-MEMBERSHIP  VALUE "10".
       77 OUTPUT-SORT-AT-END  PIC X.
         88 EOF-SORT-FILE VALUE "Y".
       77 DUMMY               PIC X.

       PROCEDURE DIVISION.
       MEMBER-SORT-PROC.

           SORT MEMBER-SORT
             ON ASCENDING KEY MEMBER-LAST-NAME
             ON DESCENDING KEY MEMBER-RANK
             WITH DUPLICATES IN ORDER
             INPUT PROCEDURE INPUT-PROC
             OUTPUT PROCEDURE OUTPUT-PROC.

           DISPLAY "SORT-3 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.


       INPUT-PROC SECTION.
       INPUT-PROCESS.

           OPEN INPUT MEMBER-FILE.
           READ MEMBER-FILE NEXT RECORD.
```

```
           PERFORM UNTIL EOF-MEMBERLIST
             RELEASE SORT-DATA FROM MEMBER-INFO

             READ MEMBER-FILE NEXT RECORD
               AT END MOVE "10" TO MEMBERLIST-STAT
             END-READ
           END-PERFORM.

           CLOSE MEMBER-FILE.
           EXIT SECTION.

       OUTPUT-PROC SECTION.
       OUTPUT-PROCESS.
           OPEN OUTPUT MEMBER-LIST.
           PERFORM UNTIL EOF-SORT-FILE
             RETURN MEMBER-SORT
               AT END MOVE "Y" TO OUTPUT-SORT-AT-END
               NOT AT END
                 WRITE SORTED-MEMBER-INFO FROM SORT-DATA
             END-RETURN
           END-PERFORM.

           CLOSE MEMBER-LIST.
           EXIT SECTION.
```

5.3.35 RETURN Statement

The RETURN statement returns records from SORT or MERGE operations to the operating program from within an OUTPUT procedure.

General Format

```
RETURN sort-file-1 RECORD [ INTO data-1 ]
   AT END statement-1
   [NOT AT END statement-2 ]
   [END-RETURN ]
```

Syntax

1. sort-file-n is a file described in the File Section with an SD.
2. data-n is a data item.
3. statement-n is an imperative statement.

General Rules

1. The RETURN statement can only be used in the OUTPUT PROCEDURE of a SORT or MERGE statement.
2. The INTO phrase causes the data to be MOVE'd to data-1.
3. If there is no INTO phrase, the data is returned into the sort-record area in the sort file.
4. When the end of a sort file is reached, the AT END condition is triggered, otherwise the NOT AT END condition is triggered.

5. The RETURN statement does not update the File Status variable.

An excerpt from the sample program illustrates the use of the RETURN statement in an OUTPUT PROCEDURE. In the example below, the program RETURNs records from the sort file, writes them out to a new file.

```
            OPEN OUTPUT MEMBER-LIST.
            PERFORM UNTIL EOF-SORT-FILE
             RETURN MEMBER-SORT
               AT END MOVE "Y" TO OUTPUT-SORT-AT-END
               NOT AT END
                 WRITE SORTED-MEMBER-INFO FROM SORT-DATA
             END-RETURN
            END-PERFORM.
```

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. RETURN-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
      *SORT INPUT FILE
           SELECT MEMBER-FILE ASSIGN TO "MEMBERSHIP.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERSHIP-STAT.

      *SORT OUTPUT FILE
           SELECT MEMBER-LIST ASSIGN TO "MEMBERLIST.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERLIST-STAT.

      *SORTFILE (SD)
           SELECT MEMBER-SORT ASSIGN TO "SORT-WORK".

       DATA DIVISION.
       FILE SECTION.
       FD MEMBER-FILE.
       01 MEMBER-INFOPIC X(40).

       FD MEMBER-LIST.
       01 SORTED-MEMBER-INFO PIC X(40).

       SD MEMBER-SORT.
       01 SORT-DATA.
         05 MEMBER-FIRST-NAME PIC X(10).
         05 MEMBER-LAST-NAME  PIC X(20).
         05 YEAR-JOINED       PIC X(4).
         05 MEMBER-RANK       PIC X(6).

       WORKING-STORAGE SECTION.
       77 MEMBERLIST-STAT PIC XX.
         88 EOF-MEMBERLIST  VALUE "10".
       77 MEMBERSHIP-STAT PIC XX.
         88 EOF-MEMBERSHIP  VALUE "10".
       77 OUTPUT-SORT-AT-END  PIC X.
         88 EOF-SORT-FILEVALUE "Y".
```

```
       77 DUMMY                PIC X.

       PROCEDURE DIVISION.
       MEMBER-SORT-PROC.

           SORT MEMBER-SORT
             ON ASCENDING KEY MEMBER-LAST-NAME
             ON DESCENDING KEY MEMBER-RANK
             WITH DUPLICATES IN ORDER
             INPUT PROCEDURE INPUT-PROC
             OUTPUT PROCEDURE OUTPUT-PROC.

           DISPLAY "SORT-3 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.


       INPUT-PROC SECTION.
       INPUT-PROCESS.

           OPEN INPUT MEMBER-FILE.
           READ MEMBER-FILE NEXT RECORD.

           PERFORM UNTIL EOF-MEMBERLIST
             RELEASE SORT-DATA FROM MEMBER-INFO

             READ MEMBER-FILE NEXT RECORD
               AT END MOVE "10" TO MEMBERLIST-STAT
             END-READ
           END-PERFORM.

           CLOSE MEMBER-FILE.
           EXIT SECTION.

       OUTPUT-PROC SECTION.
       OUTPUT-PROCESS.
           OPEN OUTPUT MEMBER-LIST.
           PERFORM UNTIL EOF-SORT-FILE
             RETURN MEMBER-SORT
               AT END MOVE "Y" TO OUTPUT-SORT-AT-END
               NOT AT END
                 WRITE SORTED-MEMBER-INFO FROM SORT-DATA
             END-RETURN
           END-PERFORM.

           CLOSE MEMBER-LIST.
           EXIT SECTION.
```

5.3.36 REWRITE Statement

The REWRITE statement modifies a record in an indexed file, posting modifications to non-key fields.

General Format

```
REWRITE record-1 [ FROM identifier-1 ]
     [ WITH [NO] LOCK  ]
     [ INVALID KEY statement-1 ]
     [ NOT INVALID KEY statement-2 ]
```

```
    [ END-REWRITE ]
```

Syntax

1. record-n is the name of a record declared in the FILE Section.
2. identifier-n is a data element, literal, or data returned from a function call.
3. statement-n is an imperative statement.

General Rules

1. The REWRITE statement can only be issued on records in a file that is OPEN I-O.
2. The REWRITE statement allows non-key fields to be altered and re-written to a file that is OPEN I-O.
3. For files declared with ACCESS IS SEQUENTIAL, the only valid target of a REWRITE statement is the record most recently retrieved by a successful READ statement. The effect of the REWRITE statement is to replace the contents of the record with the new record.
4. For files declared with ACCESS IS RANDOM or ACCESS IS DYNAMIC, the target of a REWRITE statement is determined by the setting of the primary key in the case of indexed files, and the relative key in the case of relative files.
5. The REWRITE statement does not allow records to altered in length.
6. The FROM phrase causes a MOVE from identifier-1 to the record area of record-1 prior to the execution of the REWRITE statement.
7. The WITH [NO] LOCK phrase is optional.
8. The INVALID KEY/NOT INVALID KEY phrase requires that the file be of ORGANIZATION RELATIVE or ORGANIZATION INDEXED.
9. The INVALID KEY condition is triggered by any of the following:
    a. A change has been made to an indexed file's primary key.
    b. A changes has been made to an alternate key that does not allow duplicates, and the change has created a duplicate key condition.
    c. The size of the record has changed
    d. The INVALID KEY condition causes the REWRITE to fail. If an INVALID KEY phrase is used in the REWRITE statement, then the following statement-list is executed. If no INVALID KEY condition is used, then the condition may be detected in the DECLARATIVES. If there are no DECLARATIVES, then the program aborts.
10. The NOT INVALID KEY condition exists if the INVALID KEY condition is not triggered. If a NOT INVALID KEY phrase is used in the REWRITE statement, then the following statement-list is executed.
    a. The REWRITE statement updates the FILE STATUS variable.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. REWRITE-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
```

```
           SELECT RESWORDS ASSIGN TO "RESWORDS"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS RESERVED-WORD
           FILE STATUS IS RESWORDS-STAT.

       DATA DIVISION.
       FILE SECTION.
       FD RESWORDS.
       01 RESWORDS-RECORD.
         03 RESERVED-WORD         PIC X(30).
         03 COMMENT               PIC X(20).

       WORKING-STORAGE SECTION.
       77 RESWORDS-STAT PIC XX.
         88 END-OF-RESWORDS  VALUE "10".
       77 DUMMY                   PIC X.

       PROCEDURE DIVISION.
       MAIN.
           OPEN OUTPUT RESWORDS.
           MOVE "ACCEPT" TO RESERVED-WORD.
           MOVE "HELLO WORLD" TO COMMENT.
           WRITE RESWORDS-RECORD.
           CLOSE RESWORDS.

           OPEN I-O RESWORDS.
           MOVE"ACCEPT"TO RESERVED-WORD.
           READ RESWORDS.
           MOVE "CHANGING COMMENT" TO COMMENT.
           REWRITE RESWORDS-RECORD WITH LOCK
             INVALID KEY
               DISPLAY "REWRITE FAILED! " LINE 10 COL 10
               DISPLAY RESWORDS-STAT LINE 10 COL 26
             NOT INVALID KEY
               DISPLAY "REWRITE SUCCEEDS!" LINE 10 COL 10
           END-REWRITE.

           DISPLAY "REWRITE-1 FINISHED!" LINE 11 COL 10.
           ACCEPT DUMMY LINE 11 COL 35.

           CLOSE RESWORDS.
           STOP RUN.
```

5.3.37 ROLLBACK Statement

The ROLLBACK statement causes a FILE I-O operation to be cancelled.

ROLLBACK is only available when underlying file system supports it.

General Format

```
ROLLBACK.
```

General Rules

1. The ROLLBACK statement causes unwritten file buffers to not be written to the target file.
2. After cancelling the unwritten FILE I-O operations, the ROLLBACK statement releases record and file locks held by the target file.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. ROLLBACK-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT RESWORDS ASSIGN TO "RESWORDS"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS RESERVED-WORD
           FILE STATUS IS RESWORDS-STAT.

       DATA DIVISION.
       FILE SECTION.
       FD RESWORDS.
       01 RESWORDS-RECORD.
         03 RESERVED-WORD         PIC X(30).

       WORKING-STORAGE SECTION.
       77 RESWORDS-STATPIC XX.
         88 END-OF-RESWORDS  VALUE"10".
       77 DUMMY                  PIC X.

       PROCEDURE DIVISION.
       MAIN.
           OPEN OUTPUT RESWORDS.
           MOVE "ACCEPT" TO RESERVED-WORD.
           WRITE RESWORDS-RECORD.
           ROLLBACK.
           DISPLAY "ROLLBACK-1 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.

           CLOSE RESWORDS.
           STOP RUN.
```

5.3.38 SEARCH Statement

The SEARCH statement does a sequential or binary search of a table that is populated with data. The binary search (SEARCH ALL) statement requires that the data be pre-sorted.

Format 1

The Format 1 Search statement reads through a table sequentially until a condition tests TRUE, or until the end of the table is reached.

```
SEARCH search-table-1 [VARYING {index-1 }
    [ AT END statement-1 ]
    { WHEN condition-1 {statement-2   } } ...
                       {NEXT SENTENCE }
```

```
    [ END-SEARCH ]
```

Syntax

1. search-tbl-n is a data item with an occurs clause, and an index. When used with the SEARCH ALL, a key is also required.
2. index-n is a table index, and must be described with USAGE INDEX.
3. statement-n is an imperative statement.

General Rules

1. The VARYING clause in not necessary if the table being SEARCH'ed contains an INDEXED BY clause.
2. Index-1 defines the starting position in the table of the search. For a search of a table from beginning, to end, index-1 should be SET to 1 prior to the execution of the SEARCH statement, as follows:
   SET index-1 TO 1.
3. The WHEN condition-1 phrase is tested for each instance of the table during the SEARCH. When condition-1 tests TRUE, statement-2 is executed, after which the SEARCH is terminated, and control passes to the next statement after the SEARCH statement.
4. Where multiple WHEN condition phrases are listed consecutively, they will each be tested for each instance of the table during the SEARCH. When any WHEN condition phrase tests TRUE, the statement-list associated with it is executed, after which the SEARCH is terminated, and control passes to the next statement after the SEARCH statement.
5. When the condition-test is TRUE, the index stores the location in the table of the element that has been identified.
6. The AT END condition is triggered when the entire table is SEARCH'ed and no WHEN condition phrase tests true. When the AT END condition is triggered, the statement-list associated with the AT END phrase is executed, the SEARCH is terminated, and control passes to the next statement after the SEARCH statement.

Format 2

The Format 2 Search statement performs a binary search on a table that is sorted, and in which the sort-keys are described in the declaration of the table.

```
SEARCH ALL search-table-1
    [AT END statement-1 ]
    WHEN {data-1 {IS EQUAL TO } { identifier-3 }}
                 {IS =        } {literal-1      }
                 {arithmetic-expression} {condition-1}
    [AND {data-2 {IS EQUAL TO} { identifiier-4 } } ] ...
                 { IS =      } { literal-2       }
                 {arithmetic-expr   } { condition-name-2 |}
    {statement-2   }...
    {NEXT SENTENCE }
    [END-SEARCH ]
```

Syntax

1. search-tbl-n is a data item with an occurs clause, and an index. When used with the SEARCH ALL, a key is also required.
2. data-n is a data item.
3. identifier-n is a data element, literal, or data returned from a function call.
4. literal-n is a character string.
5. condition-n is a condition name described in a level 88 statement.
6. statement-n is an imperative statement.

General Rules

1. index-1 does not need to be SET before the execution of a SEARCH ALL statement.
2. The contents of a table that is the target of a SEARCH ALL statement must be pre-sorted. The manner in which the SEARCH ALL statement determines whether or not there is a match is to internally SET the index to a value that represents the midpoint of the table, and test the truth of the condition clause. After the condition test, the SEARCH ALL statement limits the rest of its SEARCH to either the data elements above, or below the midpoint in the table. In this manner, it performs a binary search, and determines whether or not there is a match to the condition clause.
3. The Format 2 SEARCH ALL statement only permits a single WHEN phrase.
4. In order to get predictable results, the WHEN phrase must identify a unique table entry.
5. A condition test may be made up of multiple condition clauses that are connected with AND or OR .
6. When the condition-test is TRUE, the index stores the location in the table of the element that has been identified.
7. The VARYING syntax is not relevant, and not allowed on a SEARCH ALL statement.
8. The AT END condition is triggered when the entire table is SEARCH'ed and no WHEN condition phrase tests true. When the AT END condition is triggered, the statement-list associated with the AT END phrase is executed, the SEARCH is terminated, and control passes to the next statement after the SEARCH statement.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SEARCH-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 ALLSTAR-TABLE.
         05 STAR-1  PIC X(30) VALUE "01HANK    AARON     BRAVES    ".
         05 STAR-2  PIC X(30) VALUE "02YOGI    BERRA     YANKEES   ".
         05 STAR-3  PIC X(30) VALUE "03RODNEY  CAREW     TWINS     ".
         05 STAR-4  PIC X(30) VALUE "04JOE     DIMAGGIO  YANKEES   ".
         05 STAR-5  PIC X(30) VALUE "05DENNIS  ECKERSLEY AS        ".
         05 STAR-6  PIC X(30) VALUE "06CARLTON FISK      RED SOX   ".
         05 STAR-7  PIC X(30) VALUE "07LOU     GEHRIG    YANKEES   ".
         05 STAR-8  PIC X(30) VALUE "08ROGERS  HORNSBY   CARDINALS ".
         05 STAR-9  PIC X(30) VALUE "09REGGIE  JACKSON   YANKEES   ".
         05 STAR-10 PIC X(30) VALUE "10SANDY   KOUFAX    DODGERS   ".
         05 STAR-11 PIC X(30) VALUE "11TOMMY   LASORDA   DODGERS   ".
         05 STAR-12 PIC X(30) VALUE "12EDDIE   MURRAY    ORIOLES   ".
         05 STAR-13 PIC X(30) VALUE "13PHIL    NIEKRO    BRAVES    ".
```

```
           05 STAR-14 PIC X(30) VALUE "14JIM     PALMER    ORIOLES   ".
           05 STAR-15 PIC X(30) VALUE "15BABE    RUTH      YANKEES   ".
           05 STAR-16 PIC X(30) VALUE "16TOM     SEAVER    METS      ".
           05 STAR-17 PIC X(30) VALUE "17BILL    VEECK     WHITE SOX ".
           05 STAR-18 PIC X(30) VALUE "19EARL    WEAVER    ORIOLES   ".
           05 STAR-19 PIC X(30) VALUE "19BILLY   WILLIAMS  CUBS      ".
           05 STAR-20 PIC X(30) VALUE "20CARL    YAZ       RED SOX   ".
       *
        01 ALLSTAR-TBL REDEFINES ALLSTAR-TABLE.
         05 STAR-TABLE OCCURS 20 TIMES
              ASCENDING KEY IS A-LAST-NAME
              INDEXED BY INDEX-1.
           10 A-NUMBER               PIC 99.
           10 A-FIRST-NAME           PIC X(8).
           10 A-LAST-NAME            PIC X(10).
           10 A-TEAM                 PIC X(10).

        77 DUMMY   PIC X.
        PROCEDURE DIVISION.
        MAIN.
       *  SEARCH VARYING REQUIRES THE INDEX BE SET BEFORE BEGINNING
            SET INDEX-1 TO1.

            SEARCH STAR-TABLE VARYING INDEX-1
              AT END
                DISPLAY "END OF SEARCH!" LINE 11 COL 10
                  WHEN A-LAST-NAME(INDEX-1) = "KOUFAX"
                    DISPLAY A-FIRST-NAME(INDEX-1) LINE10COL10
            END-SEARCH.

       *   SEARCH ALL REQUIRES THE TABLE BE SORTED ON A KEY FIELD
       *   NAMED IN THE DECLARATION.
       *   INDEX-1 IS AUTOMATICALLY INITIALIZED BY SEARCH ALL

            SEARCH ALL STAR-TABLE
              AT END
                DISPLAY "END OF SEARCH ALL!" LINE 13 COL 10
                WHEN A-LAST-NAME(INDEX-1) = "RUTH"
                  DISPLAY A-FIRST-NAME(INDEX-1) LINE 12 COL 10
             END-SEARCH.

             DISPLAY "SEARCH-1 FINISHED!" LINE 15 COL 10.
             ACCEPT DUMMY LINE 15 COL 30.
             STOP RUN.
```

5.3.39 SET Statement

The SET statement can be used to set data types and environment variables to prescribed values.

Format 1

The Format 1 SET statement sets the value of an environment variable. If the environment variable does not exist, it is created, and assigned the given value.

```
SET ENVIRONMENT { literal-1 TO identifier-1 } ...
```

Syntax

1. literal-n is a character string.
2. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. In the Format 1 SET statement, literal-1 is the name of an environment variable that is set to the value of identifier-1.
2. The value of the environment variable can be retrieved by the ACCEPT... FROM ENVIRONMENT statement by the COBOL-IT program , and any subprograms called by the COBOL-IT program.

Code Sample

```
       77 COBOL-TYPE    PIC X(10) VALUESPACES.
       ...
           SET ENVIRONMENT "COBOL" TO "COBOL-IT".
           ACCEPT COBOL-TYPE FROM ENVIRONMENT "COBOL".
           DISPLAY COBOL-TYPE LINE 13 COL 10.
       ...
```

Format 2

The Format 2 SET statement sets a data item to a given value.

```
SET {data-1} ... TO identifier-1.
```

Syntax

1. data-n is a data item.
2. identifier-n is a data element, literal, or data returned from a function call.

General Rules

1. The Format 2 Set statement has the effect of a MOVE statement, assigning a given value to a data item.

Code Sample

```
       77 NUMERIC-1    PIC 99 VALUE 0.
       77 ALPHA-1      PIC X(5) VALUE SPACES.
       ...
           SET NUMERIC-1 TO 10.
           DISPLAY NUMERIC-1 LINE 5 COL 10.
           SET ALPHA-1 TO "HELLO".
           DISPLAY ALPHA-1 LINE 5 COL 20.
       ...
```

Format 3

The Format 3 SET statement assigns data item with USAGE POINTER to the ADDRESS of a variable.

```
SET {data-ptr-1}... TO { ADDRESS OF identifier-1 }
                       { NULL                    }
```

Syntax

1. data-ptr-n is a data item declared with USAGE POINTER.
2. identifier-n is a data element, literal, or data returned from a function call.

General Rules

There are no General Rules.

Note: The Format 3 SET statement can be used in conjunction with the Format 4 SET statement to enable the use of a Linkage item in a sub-program which has not been part of a CALL USING statement.

For an example, see the Code Sample included with the Format 4 Set Statement.

Format 4

The Format 3 SET statement assigns data item with USAGE POINTER to the ADDRESS of a variable.

```
SET {ADDRESS OF linkage-data-item-1}... TO data-ptr-1
```

Syntax

1. linkage-data-item-n is a data item declared in the LINKAGE SECTION.

General Rules

There are no General Rules.

Note: The Format 3 SET statement can be used in conjunction with the Format 4 SET statement to enable the use of a Linkage item in a sub-program which has not been part of a CALL USING statement.

Referencing a Linkage item that had no address associated with it would normally abort a runtime session. In the sample below, the linkage item acquires a valid address, and can be referenced without a problem.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SUBPGM.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 COBOL-TYPE   PIC X(8) VALUE "COBOL-IT".
       77 DATA-PTR-1   USAGE POINTER.
       77 DATA-PTR-2   USAGE POINTER.
       LINKAGE SECTION.
       01 LINK-1  PIC X(8).
       PROCEDURE DIVISION.
       MAIN.
           SET DATA-PTR-1 TO ADDRESS OF COBOL-TYPE.
           DISPLAY "DATA-PTR SET TO ADDRESS" LINE 6 COL 10.
           SET DATA-PTR-2 TO NULL.
           DISPLAY "DATA-PTR SET TO NULL" LINE 7 COL 10.
           SET ADDRESS OF LINK-1 TO DATA-PTR-1.
           DISPLAY "LINK-1: " LINE 8 COL 10.
           DISPLAY LINK-1 LINE 8 COL 20.
           GOBACK.
```

Format 5

```
SET {data-1} ... {UP   } BY integer-1
                 {DOWN}
```

Syntax

1. data-n is a data element that is an integer.
2. integer-n is a data element, literal or data returned from a function call that is an integer.

General Rules

1. The Format 5 SET statement has the effect of an ADD or SUBTRACT statement, incrementing or decrementing a numeric data item with an integer value be a given integer value.
2. The Format 5 SET statement may be used with indexes, as a means of directing table handling.

Code Sample

```
       77 NUMERIC-1  PIC 99 VALUE 20.
       ...
```

```
           SET NUMERIC-1 UP BY 10.
           DISPLAY NUMERIC-1 LINE 6 COL 10.
           SET NUMERIC-1 DOWN BY 5.
           DISPLAY NUMERIC-1 LINE 7 COL 10.
       ...
```

Format 6

```
SET { {condition-1} ... TO {TRUE } } ...
                           {FALSE}
```

Syntax

1. condition-n is a condition name that is described in a level 88 statement.

General Rules

1. The Format 6 SET statement has the effect of MOVE statement, MOVEing a value to the parent data item to create a TRUE condition when the condition is SET to TRUE, and MOVEing a value to the parent data item to create a FALSE condition when the condition is SET TO FALSE.
2. IF the level-88 condition contains a WHEN SET TO FALSE value, then the effect of the SET [condition-1] TO FALSE is to MOVE that value to the parent data item.
3. If an attempt is made to set a level-88 condition to FALSE, and that condition does not have an explicit WHEN SET TO FALSE clause, the compiler will generate an error.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SET3.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 DECISION-FLAG PIC 9 VALUE 0.
         88 DECISION-YES   VALUES 1 THRU 5.
            WHEN SET TO FALSE 6.
       77 WS-DUMMY   PIC X.
       PROCEDURE DIVISION.
       MAIN.
           SET DECISION-YES TO TRUE.
           DISPLAY DECISION-FLAG LINE 8 COL 10.
           SET DECISION-YES TO FALSE.
           DISPLAY DECISION-FLAG LINE 9 COL 10.
           DISPLAY "ALL DONE!" LINE 10 COL 10.
           ACCEPT WS-DUMMY LINE 10 COL 30.
           STOP RUN.
```

Format 7

```
SET { {mnemonic-name-1} … TO (ON ) } …
                             (OFF)
```

Syntax

1. mnemonic-name is a user-defined word.

General Rules

1. The Format 7 SET statement SETs Switches that are described in SPECIAL-NAMES to ON or OFF. If the switch is described with an ON STATUS and OFF STATUS, these conditions can programmatically be checked for truth.

Code Sample

```
...
 SPECIAL-NAMES.
       SWITCH 1IS SWITCH-1
       ON STATUS IS SET-CHECK
       OFF STATUS IS SKIP-CHECK.
...

     SET SWITCH-1 TO ON.
...
     IF SET-CHECK
       DISPLAY "SWITCH 1 SET" LINE 4 COL 10
     END-IF.
...
```

Code Sample

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SET-1.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SPECIAL-NAMES.
     SWITCH 1IS SWITCH-1
     ON STATUS IS SET-CHECK
     OFF STATUS IS SKIP-CHECK.

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 77 NUMERIC-1 PIC 99 VALUE 0.
 77 DECISION-FLAG PIC 9 VALUE 0.
   88 DECISION-YES VALUE 1
      WHEN SET TO FALSE 0.
 77 COBOL-TYPE PIC X(10) VALUE SPACES.
```

```
       77 DATA-PTR-1 USAGE POINTER.
       77 DATA-PTR-2 USAGE POINTER.
       01 DUMMY  PIC X.
       LINKAGE SECTION.
       01 LINKAGE-1        PIC X(10).
       PROCEDURE DIVISION.
       MAIN.
      * FORMAT 1
           SET ENVIRONMENT "COBOL" TO "COBOL-IT".
           ACCEPT COBOL-TYPE FROM ENVIRONMENT "COBOL".
           DISPLAY COBOL-TYPE LINE 4 COL 10.
      *
      * FORMAT 2
      *
           SET NUMERIC-1 TO 10.
           DISPLAY NUMERIC-1 LINE 5 COL 10.
      *
      * FORMAT 3
      *
           SET DATA-PTR-1 TO ADDRESS OF COBOL-TYPE.
           DISPLAY "DATA-PTR SET TO ADDRESS" LINE 6 COL 10.
           SET DATA-PTR-2 TO NULL.
           DISPLAY "DATA-PTR SET TO NULL" LINE 7 COL 10.
      *
      * FORMAT 4
      *
           SET ADDRESS OF LINKAGE-1 TO DATA-PTR-1.
           DISPLAY "ADDRESS OF LINKAGE ITEM SET" LINE 8 COL 10.
      *
      * FORMAT 5
      *
           SET NUMERIC-1 UP BY 10.
           DISPLAY NUMERIC-1 LINE 9 COL 10.
           SET NUMERIC-1 DOWN BY 5.
           DISPLAY NUMERIC-1 LINE 10 COL 10.
      *
      * FORMAT 6
      *
           SET DECISION-YES TO TRUE.
           DISPLAY DECISION-FLAG LINE 8 COL 10.
           SET DECISION-YES TO FALSE.
           DISPLAY DECISION-FLAG LINE 9 COL 10.
      *
      * FORMAT 7
      *
           SET SWITCH-1 TO ON.
           IF SET-CHECK
             DISPLAY "SWITCH 1 SET" LINE 11 COL 10
           END-IF.
      *
           DISPLAY "SET-1 FINISHED!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.

           STOP RUN.
```

5.3.40 SORT Statement

The SORT statement reorders the records in a file, or a table .

Format 1

The Format 1 SORT statement reorders the records in a file.

```
SORT sort-file-1
       { ON {ASCENDING } KEY {sort-key-1} } ...
         {DESCENDING}
       [ WITH DUPLICATES IN ORDER ]
       [ COLLATING SEQUENCE IS alphabet-name ]
       { USING {file-1} ...   }
       { GIVING {file-2} ... }
       { INPUT PROCEDURE IS proc-name  }
       { OUTPUT PROCEDURE IS proc-name }
```

Syntax

1. sort-file-n is a file described in the File Section with an SD.
2. sort-key-n is a data item that has been declared as a key to a sort table, or sort file.
3. alphabet-name is a user-defined word.
4. proc-name is the name of a paragraph or section in the program.
5. file-n is a file described in the File Section with an FD.

General Rules

1. Sort-file-1 must be described with an SD in the FILE SECTION.
2. The USING and GIVING files must be described with ORGANIZATION LINE SEQUENTIAL, or ORGANIZATION BINARY SEQENTIAL, and must be described with an FD in the FILE SECTION.
3. The order in which the KEY clauses are listed indicates the order in which the sort keys are applied. The first KEY listed is the primary key, the second key listed is the second key, etc...
4. The ASCENDING KEY clause causes values to sort from lowest to highest.
5. The DESCENDING KEY clause causes values to sort from highest to lowest.
6. The COLLATING SEQUENCE clause names the alphabet which is used for purposes of ordering KEY data in the SORT. Usage of the COLLATING SEQUENCE clause in a SORT statement overrides the naming of a PROGRAM COLLATING SEQUENCE in the object-computer paragraph.
7. WITH DUPLICATES IN ORDER is treated as commentary.
8. The SORT statement writes duplicates in the order in which they are encountered.
9. The file listed in the USING clause may not be OPEN when the SORT statement executes. The SORT statement OPENs the file.
10. The files listed in the GIVING clause may not be OPEN when the MERGE statement executes. The SORT statement OPENs the file.
11. The OUTPUT PROCEDURE is executed after all of the records have been SORTed into the temporary sort-file.
12. Inside the OUTPUT PROCEDURE, the RETURN statement is used to transfer records from the sort-file to the output file.
13. When the OUTPUT PROCEDURE is finished, the SORT statement closes all files.
14. The SORT statement updates file status variable that is defined for sort-file after performing

OPEN, READ, WRITE, and CLOSE operations. File Status errors may be detected in DECLARATIVES.

Format 2

The Format 2 SORT statement reorders the elements in a table.

```
SORT sort-tbl
       [ ON ASCENDING/DESCENDING KEY data-name-1 ... ]
       [ WITH DUPLICATES IN ORDER ]
       [ COLLATING SEQUENCE IS alphabet-name ]
```

Syntax

1. sort-tbl is a data- item with an occurs clause.
2. data-name-n is a data-item defined in the sort-tbl, subordinate to the OCCURS.
3. alphabet-name is a user-defined word.

General Rules

1. The WITH DUPLICATES IN ORDER phrase is treated as commentary.
2. The COLLATING SEQUENCE phrase is treated as commentary.
3. The Format 2 SORT statement does not require a sort file definition (SD) in the File Section.
4. The effect of the Format 2 SORT statement is a re-ordering of the elements in sort-tbl, as specified by the ASCENDING/DESCENDING KEY phrase(s).
5. The internal COBOL SORT checks for the setting of TMPDIR as the location for the storage of temporary SORT files. If TMPDIR is not set, then the behavior is OS-Dependent. In some operating systems, this can limit the size of files that can be SORTed by the internal COBOL SORT.

Code Sample

SORT USING/GIVING

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SORT-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
      *INPUT FILE
           SELECT MEMBER-FILE ASSIGN TO "MEMBERSHIP.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERSHIP-STAT.

      *OUTPUT FILE
```

```
           SELECT MEMBER-LIST ASSIGN TO "MEMBERLIST.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERLIST-STAT.

      *SORTFILE (SD)
           SELECT MEMBER-SORT ASSIGN TO "SORT-WORK".

       DATA DIVISION.
       FILE SECTION.
       FD MEMBER-FILE.
       01 MEMBER-INFOPIC X(40).

       FD MEMBER-LIST.
       01 SORTED-MEMBER-INFOPIC X(40).

       SD MEMBER-SORT.
       01 SORT-DATA.
        05 MEMBER-FIRST-NAME PIC X(10).
        05 MEMBER-LAST-NAME  PIC X(20).
        05 YEAR-JOINED       PIC X(4).
        05 MEMBER-RANK       PIC X(6).

       WORKING-STORAGE SECTION.
       77 MEMBERLIST-STAT PIC XX.
       77 MEMBERSHIP-STATPIC XX.

       PROCEDUREDIVISION.
       PRODUCT-LIST-SORT.
           SORT MEMBER-SORT
             ON ASCENDING KEY MEMBER-LAST-NAME
             ON DESCENDING KEY MEMBER-RANK
             WITH DUPLICATES IN ORDER
             USING  MEMBER-FILE
             GIVING MEMBER-LIST.
```

SORT Table

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SORT-2.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       77 MEMBERLIST-STAT PIC XX.
       77 MEMBERSHIP-STAT PIC XX.
       77 LINE-NUMBER     PIC 99 VALUE 10.
       77 CTR             PIC 99 VALUE 0.
       77 DUMMY           PIC X.

       01 MEMBER-TABLE.
         05 MEMBER-1 PIC X(20) VALUE "SAMUEL   JOHNSON    ".
         05 MEMBER-2 PIC X(20) VALUE "THOMAS   JEFFERSON  ".
         05 MEMBER-3 PIC X(20) VALUE "GEORGE   WASHINGTON ".
         05 MEMBER-4 PIC X(20) VALUE "SAMUEL   ADAMS      ".
         05 MEMBER-5 PIC X(20) VALUE "ABRAHAM  LINCOLN    ".
         05 MEMBER-6 PIC X(20) VALUE "WOODROW  WILSON     ".
         05 MEMBER-7 PIC X(20) VALUE "FRANKLIN ROOSEVELT  ".
         05 MEMBER-8 PIC X(20) VALUE "DWIGHT   EISENHOWER ".
       01 MEMBER-TABLE-RED REDEFINES MEMBER-TABLE.
         05 PATRIOT-TBL OCCURS 8 TIMES.
           10 FIRST-NAME PIC X(10).
```

```
           10 LAST-NAME  PIC X(10).
       *
        PROCEDURE DIVISION.
        MEMBER-SORT.

           SORT PATRIOT-TBL
             ON ASCENDING KEY FIRST-NAME.

           PERFORM VARYING CTR FROM 1 BY 1 UNTIL CTR > 8
             DISPLAY FIRST-NAME(CTR) LINE LINE-NUMBER COL 10
             DISPLAY LAST-NAME(CTR)  LINE LINE-NUMBER COL 20
             ADD 1 TO LINE-NUMBER
           END-PERFORM.


           DISPLAY "SORT-2 FINISHED!" LINE 20 COL 10.
           ACCEPT DUMMY LINE 20 COL 30.
           STOP RUN.
```

SORT INPUT/OUTPUT Procedures

```
        IDENTIFICATION DIVISION.
        PROGRAM-ID. SORT-3.
        ENVIRONMENT DIVISION.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
       *INPUT FILE
           SELECT MEMBER-FILE ASSIGN TO "MEMBERSHIP.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERSHIP-STAT.

       *OUTPUT FILE
           SELECT MEMBER-LIST ASSIGN TO "MEMBERLIST.TXT"
           ORGANIZATION IS LINE SEQUENTIAL
           ACCESS IS SEQUENTIAL
           FILE STATUS IS MEMBERLIST-STAT.

       *SORTFILE (SD)
           SELECT MEMBER-SORT ASSIGN TO "SORT-WORK".

        DATA DIVISION.
        FILE SECTION.
        FD MEMBER-FILE.
        01 MEMBER-INFO         PIC X(40).

        FD MEMBER-LIST.
        01 SORTED-MEMBER-INFO  PIC X(40).

        SD MEMBER-SORT.
        01 SORT-DATA.
          05 MEMBER-FIRST-NAME PIC X(10).
          05 MEMBER-LAST-NAME  PIC X(20).
          05 YEAR-JOINED       PIC X(4).
          05 MEMBER-RANK       PIC X(6).

        WORKING-STORAGESECTION.
        77 MEMBERLIST-STAT PIC XX.
          88 EOF-MEMBERLIST  VALUE "10".
        77 MEMBERSHIP-STAT PIC XX.
          88 EOF-MEMBERSHIP  VALUE "10".
        77 OUTPUT-SORT-AT-END  PIC X.
```

```
          88 EOF-SORT-FILE VALUE "Y".
       77 DUMMY                PIC X.

       PROCEDURE DIVISION.
       MEMBER-SORT-PROC.

           SORT MEMBER-SORT
             ON ASCENDING KEY MEMBER-LAST-NAME
             ON DESCENDING KEY MEMBER-RANK
             WITH DUPLICATES IN ORDER
             INPUT PROCEDURE INPUT-PROC
             OUTPUT PROCEDURE OUTPUT-PROC.

           DISPLAY "SORT-3 FINISHED!" LINE 10 COL 10.
           ACCEPT DUMMY LINE 10 COL 30.
           STOP RUN.

       INPUT-PROC SECTION.
       INPUT-PROCESS.
           OPEN INPUT MEMBER-FILE.
           READ MEMBER-FILE NEXT RECORD.

           PERFORM UNTIL EOF-MEMBERLIST
             RELEASE SORT-DATA FROM MEMBER-INFO

             READ MEMBER-FILE NEXT RECORD
               AT END MOVE "10" TO MEMBERLIST-STAT
             END-READ
           END-PERFORM.

           CLOSE MEMBER-FILE.
           EXIT SECTION.

       OUTPUT-PROC SECTION.
       OUTPUT-PROCESS.
           OPEN OUTPUT MEMBER-LIST.
           PERFORM UNTIL EOF-SORT-FILE
             RETURN MEMBER-SORT
               AT END MOVE "Y" TO OUTPUT-SORT-AT-END
               NOT AT END
                 WRITE SORTED-MEMBER-INFO FROM SORT-DATA
             END-RETURN
           END-PERFORM.

           CLOSE MEMBER-LIST.
           EXIT SECTION.
```

5.3.41 START Statement

The START statement positions the file pointer of an indexed or relative file for a READ NEXT or READ PREVIOUS statement based on a condition that relates to the value of a KEY of the file.

General Format

```
START file-name-1
  [KEY IS { EQUAL TO }   keyname-1 ]
          [ { =      }   ]
```

```
[ { GREATER THAN  } ]
[ { >             } ]
[ { NOT LESS THAN } ]
        [ { NOT <         } ]
[ { GREATER THAN OR EQUAL TO } ]
[ { >=            } ]
[ { LESS THAN     } ]
[ { <             } ]
[ { NOT GREATER THAN } ]
[ { NOT >           } ]
[ { <=              } ]
[ { LESS THAN OR EQUAL TO } ]
[ INVALID KEY statement-1 ]
[ NOT INVALID KEY statement-2 ]
[ END-START ]
```

Syntax

1. file-name-n is an indexed or relative file described in the FILE Section.
2. keyname-1 is a literal or data element whose value creates the condition test for the placing of the file pointer by the START statement.
3. statement-n is an imperative statement.

General Rules

1. file-name-1 must be OPEN INPUT or OPEN I-O when the START statement executes.
2. If file-name-1 is a relative file, the key of reference is the data item named by the RELATIVE KEY phrase in the SELECT clause of the file.
3. If file-name-1 is an indexed file, the key of reference may be either the data item(s) referenced in the RECORD KEY phrase, or the data item(s) referenced in an ALTERNATE RECORD KEY phrase.
4. If keyname-1 is the data name referenced by the RECORD KEY phrase, then the START/READ NEXT sequence will be made on the primary key.
5. If keyname-1 is the data name referenced by the ALTERNATE RECORD KEY phrase, then the START/READ NEXT sequence will be made on the alternate record key.
6. A successful START statement positions the file pointer according to the following rules:
    a. Where the condition operand is one of the following: EQUAL (“ =”), GREATER THAN (“>”), GREATER THAN OR EQUAL (“>=”), the file pointer is placed at the first record that satisfies the KEY clause specification.
    b. Where the condition operand is one of the following: LESS THAN (“<”), LESS THEN OR EQUAL (“<=”), the file pointer is placed at the last record that satisfies the KEY clause specification.
7. The INVALID KEY condition exists if the START statement is unable to place the record pointer based on the condition described.
8. When the INVALID KEY condition exists, one of the following occurs:
9. If there is an INVALID KEY phrase in the START statement, the associated statement-list is executed, and control then passes to the next statement in the program.
10. If there is no INVALID KEY phrase in the START statement, the FILE STATUS variable of

the file is updated with an error code, and control is either transferred to the DECLARATIVES, if they have been set up for this case, or the program aborts.

11. The NOT INVALID KEY condition exists if the START statement is able to plsce the record pointer based on the condition described.

Code Sample

```
...
       SELECT CUSTFILE ASSIGN TO "CUSTOMER"
       ORGANIZATION IS INDEXED
       ACCESS IS DYNAMIC
       RECORD KEY IS CUSTOMER-ID
       FILE STATUS IS CUSTOMER-STAT.
...
  FD CUSTFILE.
  01 CUSTOMER-RECORD.
    03 CUSTOMER-ID     PIC 9(5).
    03 CUSTOMER-NAME   PIC X(20).
    03 CUSTOMER-ADDR   PIC X(20).
    03 CUSTOMER-CITY   PIC X(10).
    03 CUSTOMER-STATE  PIC XX.
    03 CUSTOMER-PHONE  PIC X(10).
...
  77 CUSTOMER-STAT PIC XX.
...
       MOVE 11111 TO CUSTOMER-ID.

       START CUSTFILE KEY EQUAL TO CUSTOMER-ID.

       START CUSTFILE KEY = CUSTOMER-ID.

       START CUSTFILE KEY GREATER THAN CUSTOMER-ID.

       START CUSTFILE KEY > CUSTOMER-ID.

       START CUSTFILE KEY NOT LESS THAN CUSTOMER-ID.

       START CUSTFILE KEY NOT < CUSTOMER-ID.

       START CUSTFILE KEY GREATER THAN OR EQUAL TO CUSTOMER-ID.

       START CUSTFILE KEY >= CUSTOMER-ID.

       START CUSTFILE KEY LESS THAN CUSTOMER-ID.

       START CUSTFILE KEY < CUSTOMER-ID.

       START CUSTFILE KEY NOT GREATER THAN CUSTOMER-ID.

       START CUSTFILE KEY NOT > CUSTOMER-ID.

       START CUSTFILE KEY <= CUSTOMER-ID.

       START CUSTFILE KEY LESS THAN OR EQUAL TO CUSTOMER-ID.

       START CUSTFILE KEY LESS THAN OR EQUAL TO CUSTOMER-ID

          INVALID KEY DISPLAY "INVALID KEY!" LINE 17 COL 10
          NOT INVALID KEY
            DISPLAY "NOT INVALID KEY!" LINE 17 COL 10
```

```
           END-START.
```

5.3.42 STOP Statement

The STOP statement terminates the current program.

Format 1

```
STOP RUN [ {RETURNING} numeric-1 ]
           {GIVING }
```

Syntax

1. numeric-n is a literal or data item that is numeric.

General Rules

1. A Format 1 STOP RUN statement terminates the program, optionally returning a return code to the operating system. Any OPEN files are CLOSEd.
2. In the Format 1 STOP RUN statement, the RETURNING/GIVING clause returns the value of numeric-1 to the operating system through the special return-code register.
3. RETURNING and GIVING are synonyms.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  STOP-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 NUMERIC-1   PIC 9 VALUE 1.
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *FORMAT 1
      *STOP RUN [ {RETURNING} NUMERIC ]
      *           {GIVING }
            EVALUATE DUMMY
              WHEN "A"
                STOP RUN
              WHEN "B"
                STOP RUN RETURNING NUMERIC-1
              WHEN "C"
                STOP RUN GIVING NUMERIC-1
              WHEN "D"
                STOP RUN RETURNING 1
              WHEN "E"
                STOP RUN GIVING 1
              WHEN OTHER
                CONTINUE
            END-EVALUATE.

            DISPLAY "STOP-1 FINISHED!" LINE 15 COL 10.
            ACCEPT DUMMY LINE 15 COL 30.
            STOP RUN.
```

5.3.43 STRING Statement

The STRING statement concatenates separate literals and data items into a single data string.

General Format

```
STRING { {src-string-1} ...   [DELIMITED BY {delimiter-1}] } ...
                                           {SIZE     }
       INTO target-string-1
       [ WITH POINTER pointer-variable-1   ]
       [ ON OVERFLOW statement-1     ]
       [ NOT ON OVERFLOW statement-2 ]
       [ END-STRING ]
```

Syntax

1. Src-string-n is a alphanumeric data element, literal, or data returned from a function call.
2. pointer-variable-n is a numeric data element with a positive integer value.
3. delimiter-n is a character string.
4. target-string-1 is an alphanumeric data item.
5. statement-n is an imperative statement.

General Rules

1. The STRING statement concatenates consecutive src-string-n, and stores the result in the targe-string data item that follows the INTO phrase.
2. There is no limit to the number of src-string-n identifiers that may be listed.
3. The DELIMITED BY clause provides a way to identify a substring of a src-identifier, for purposes of the STRING operation.
4. The DELIMITED BY delimiter-1 clause causes the STRING operation to stop copying data from the src-string when the delimiter-1 is encountered in the src-string. Delimiter-1 is not copied to the target-string.
5. The DELIMITED BY SIZE clause causes the entire src-string to be copied into the target string.
6. If there is no DELIMITED BY phrase, then DELIMITED BY SIZE is assumed.
7. The WITH POINTER clause provides a way to set a position in the target-string where the STRING statement will copy the selected source using a pointer-variable.
8. The WITH POINTER clause causes an explicit pointer-variable to be referenced by the STRING operation when it begins to copy data from a src-string to a target-string. The value of the pointer-variable is used as the position in the target-string at which to copy the data.
9. When the WITH POINTER clause is used, pointer-variable-n must be set to a positive integer value before the STRING statement is executed. Most commonly, the pointer-variable is initialized to 1, indicating that the STRING statement should begin copying into the target-string at the first byte of the target-string.
10. The pointer-variable is automatically incremented by 1 for each character that is copied into

the target-string.

11. If there is no WITH POINTER clause, the position at which the src-string is copied into the target-string is the next character position after the last character position occupied by the previous src-string.
12. The ON OVERFLOW condition is triggered when the pointer variable does not contain a positive value, or when the length of target-string is less than the combined length of the src-strings named in the STRING statement.
13. When an ON OVERFLOW condition exists, the STRING is terminated, and the ON OVERFLOW statement-1 is executed.
14. The NOT ON OVERFLOW condition exists when the STRING statement has completed, an the ON OVERFLOW condition does not exist.
15. When a NOT ON OVERFLOW condition exists, statement-2 is executed.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. STRING-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ADDRESS-1
         PIC X(40)  VALUE"231/RUE SAINT-HONORE/75001 PARIS/FRANCE".
       01  ADDRESS-2.
         05  ADDRESS-NUM         PIC X(5)       VALUE SPACES.
         05  ADDRESS-NAME        PIC X(20)      VALUE SPACES.
         05  ADDRESS-CITY        PIC X(20)      VALUE SPACES.
         05  ADDRESS-COUNTRY     PIC X(20)      VALUE SPACES.
       01  ADDRESS-3           PIC X(60)        VALUE SPACES.
       01 DUMMY  PIC X.
       PROCEDURE DIVISION.
       MAIN.
           UNSTRING ADDRESS-1 DELIMITED BY "/"
              INTO ADDRESS-NUM
                   ADDRESS-NAME
                   ADDRESS-CITY
                   ADDRESS-COUNTRY.

           DISPLAY "THE STREET #:     " ADDRESS-NUM LINE 4 COL 10.
           DISPLAY "THE STREET NAME: " ADDRESS-NAME LINE 5 COL 10.
           DISPLAY "THE CITY:         " ADDRESS-CITY LINE 6 COL 10.
           DISPLAY "THE COUNTRY:      " ADDRESS-COUNTRY LINE 7 COL 10.

           ACCEPT DUMMY LINE 7 COL 33.
           STRING  ADDRESS-NUM     DELIMITED BY " "
                          " "      DELIMITED BY SIZE
                   ADDRESS-NAME    DELIMITED BY "  "
                          ", "     DELIMITED BY SIZE
                   ADDRESS-CITY    DELIMITED BY "  "
                          ", "     DELIMITED BY SIZE
                   ADDRESS-COUNTRY DELIMITED BY SIZE INTO ADDRESS-3
             ON OVERFLOW
              DISPLAY "STRING FAILED ON OVERFLOW" LINE 10 COL 10
             NOT ON OVERFLOW
              DISPLAY "THE ADDRESS IS " ADDRESS-3 LINE 10 COL 10
           END-STRING.

           DISPLAY "STRING-1 FINISHED!" LINE 12 COL 10.
           ACCEPT DUMMY LINE 12 COL 30.
           STOP RUN.
```

5.3.44 SUBTRACT Statement

The SUBTRACT Statement performs an arithmetic subtract operation on a number of operands and allows for the storage of the result in a number of data items.

Format 1

```
SUBTRACT {integer-1} ... FROM {integer-data-1 ROUNDED] } ...
         [ ON SIZE ERROR statement-1 ]
         [ NOT ON SIZE ERROR statement-2 ]
         [ END-SUBTRACT ]
```

Format 2

```
SUBTRACT {integer-2} ... FROM integer-3
         GIVING { integer-data-2 [ROUNDED] } ...
         [ ON SIZE ERROR statement-1 ]
         [ NOT ON SIZE ERROR statement-2 ]
         [ END-SUBTRACT ]
```

Format 3

```
SUBTRACT {CORRESPONDING} group-1 FROM group-2 [ROUNDED]
         {CORR }
         [ ON SIZE ERROR statement-1 ]
         [ NOT ON SIZE ERROR statement-2 ]
         [ END-SUBTRACT ]
```

Syntax

1. integer-n is a data element, literal or data returned from a function call that is an integer.
2. integer-data-n is a data item that is an integer.
3. group-n is a group data item containing one or more elementary data items.
4. statement-n is an imperative statement.
5. The SIZE ERROR exception is triggered if the receiving field is not large enough to accommodate the result of the SUBTRACT function.
6. In a SUBTRACT CORRESPONDING operation, elementary items in separate group items must have the same elementary data name for the SUBTRACT function to be performed.

General Rules

1. In a FORMAT 1 SUBTRACT statement with no GIVING clause, all integer-n data items are

subtracted, in sequence, from each of the data items following the FROM clause.

2. In a FORMAT 2 SUBTRACT statement containing a GIVING clause, all integer-n data items are subtracted, in sequence, from the data item following the FROM clause with the result stored in the integer-data item following the GIVING clause.
3. The ROUNDED clause is applied when an arithmetic operation produces a result that includes more decimal places than are included in the description of the data item given to hold the final result of the arithmetic operation.
4. For rules regarding the ROUNDED clause, see the entry for ROUNDED in 5.3.1 Common General Rules.
5. The SIZE ERROR exception is triggered if the ON SIZE ERROR clause is present and if the receiving field is not large enough to accommodate the result of the SUBTRACT function.
6. For rules regaring the ON SIZE ERROR clause, see the entry for ON SIZE ERROR in 5.3.1 Common General Rules.
7. In a SUBTRACT CORRESPONDING operation, elementary items in separate group items must have the same elementary data name for the SUBTRACT function to be performed.
8. The SUBTRACT CORRESPONDING operation causes multiple SUBTRACT operations to be performed between elementary data items in separate group items, where the elementary items have the same elementary data name, and are not described as FILLER.
9. The rules for identifying a CORRESPONDING data item for the purposes of the SUBTRACT CORRESPONDING clause are the same as the rules used for the purposes of the MOVE CORRESPONDING clause. For more detail, see the General Rules of the MOVE CORRESPONDING clause.
10. For details on how data items are identified as CORRESPONDING, see 5.3.1 Common General Rules / Rules for identifying CORRESPONDING data elements.
11. In a SUBTRACT CORRESPONDING operation, all SUBTRACT operations will be completed before the ON SIZE ERROR condition will be triggered.

Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  SUBTRACT-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 FIELD-1 PIC 9V9.
       77 FIELD-2 PIC 9V9.
       77 FIELD-3 PIC9.9.

       01 GROUP-1.
         03 FLD-1    PIC 99 VALUE 5.
         03 FLD-2    PIC 99 VALUE 15.
       01 GROUP-2.
         03 FLD-1    PIC 99 VALUE 10.
         03 FLD-2    PIC 99 VALUE 20.

       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
      *SUBTRACT {INTEGER-1} ... FROM {INTEGER-2 [ROUNDED] } ...
      *   [ ON SIZE ERROR STATEMENT-1 ]
      *   [ NOT ON SIZE ERROR STATEMENT-2 ]
      *   [ END-SUBTRACT ]
      *
```

```
           MOVE 2.5 TO FIELD-1.
           MOVE 3.4 TO FIELD-2.

           SUBTRACT FIELD-1 FROM FIELD-2
             ON SIZE ERROR
               DISPLAY "INVALID SUBTRACTION" LINE 10 COL 10
             NOT ON SIZE ERROR
               DISPLAY FIELD-2 LINE 10 COL 10
           END-SUBTRACT.


      *SUBTRACT {INTEGER-3} ... FROM INTEGER-4
      *   GIVING { INTEGER-5 [ROUNDED] } ...
      *   [ ON SIZE ERROR STATEMENT-1 ]
      *   [ NOT ON SIZE ERROR STATEMENT-2 ]
      *   [ END-SUBTRACT ]
      *

           MOVE 2.5 TO FIELD-1.
           MOVE 3.4 TO FIELD-2.

           SUBTRACT FIELD-1 FROM FIELD-2 GIVING FIELD-3
             ON SIZE ERROR DISPLAY "INVALID SUBTRACTION" LINE 10 COL 10
             NOT ON SIZE ERROR DISPLAY FIELD-3 LINE 10 COL 10
           END-SUBTRACT.


      *SUBTRACT {CORRESPONDING} GROUP-1 FROM GROUP-2 [ROUNDED]
      *         {CORR }
      *   [ ON SIZE ERROR STATEMENT-1 ]
      *   [ NOT ON SIZE ERROR STATEMENT-2 ]
      *   [ END-SUBTRACT ]

           SUBTRACT CORRESPONDING GROUP-1 FROM GROUP-2
             ON SIZE ERROR
               DISPLAY "INVALID SUBTRACTION" LINE 10 COL 10
             NOT ON SIZE ERROR
               DISPLAY FLD-1 OF GROUP-2 LINE 12 COL 10
               DISPLAY FLD-2 OF GROUP-2 LINE 13 COL 10
           END-SUBTRACT.

           DISPLAY "SUBTRACT-1 FINISHED!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.
           STOP RUN.
```

5.3.45 TRANSFORM Statement

The TRANSFORM statement converts characters in a data item from one character string to another.

General Format

```
TRANSFORM identifier-1 FROM literal-1 TO literal-2
```

Syntax

1. Identifier-n is a data item, literal, or data returned from a function call that is alphanumeric.
2. literal-n is a character string.

General Rules

1. The TRANSFORM statemnt causes a conversion of characters within identifier-1 according to rules established by the ordinal positions of characters in identifier-2 and identifier-3. To clarify with an example, the statement :

   TRANSFORM identifier-1 FROM "AB" TO "CD"

   causes every instance of "A" in identifier-1 (the first character in identifier-2), to be converted to "C" (the first character in identifier-3), and causes every instance of "B" in identifier-1 (the second character in identifier-2) to be replaced by "D" (the second character in identifier-3.

   Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  TRANSFORM-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77 IDENTIFIER-1 PIC X(10) VALUE"AAAXXXXBBB".
       77 DUMMY PIC X.
       PROCEDURE DIVISION.
       MAIN.
           TRANSFORM IDENTIFIER-1 FROM "AB" TO "CD".
           DISPLAY IDENTIFIER-1 LINE 10 COL 10.
           DISPLAY "TRANSFORM-1 FINISHED!" LINE 15 COL 10.
           ACCEPT DUMMY LINE 15 COL 30.
           STOP RUN.
```

5.3.46 UNLOCK Statement

The UNLOCK statement removes record locks.

General Format

```
UNLOCK file-1  {RECORD }
               {RECORDS}
```

Syntax

1. file-n is a file described in the File Section with an FD.

General Rules

1. file-1 must be OPEN when the UNLOCK statement is executed.
2. The UNLOCK statement releases record locks for file-1.

Code Sample

```
       IDENTIFICATION DIVISION.
```

```
       PROGRAM-ID. UNLOCK-1.
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT RESWORDS ASSIGN TO "RESWORDS"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS RESERVED-WORD
           FILE STATUS IS RESWORDS-STAT.

       DATA DIVISION.
       FILE SECTION.
       FD RESWORDS.
       01 RESWORDS-RECORD.
         03 RESERVED-WORD         PIC X(30).
         03 COMMENT               PIC X(20).

       WORKING-STORAGE SECTION.
       77 RESWORDS-STAT PIC XX.
         88 END-OF-RESWORDS  VALUE "10".
       77 DUMMY                   PIC X.

       PROCEDURE DIVISION.
       MAIN.
           OPEN OUTPUT RESWORDS.
           MOVE "ACCEPT" TO RESERVED-WORD.
           MOVE "HELLO WORLD" TO COMMENT.
           WRITE RESWORDS-RECORD.
           CLOSE RESWORDS.

           OPEN I-O RESWORDS.
           MOVE "ACCEPT" TO RESERVED-WORD.
           READ RESWORDS WITH LOCK.

           UNLOCK RESWORDS RECORD.
           DISPLAY "UNLOCK [FILE] RECORD" LINE 5 COL 10.

           MOVE "ACCEPT" TO RESERVED-WORD.
           READ RESWORDS WITH LOCK.

           UNLOCK RESWORDS RECORDS.
           DISPLAY "UNLOCK [FILE] RECORDS" LINE 7 COL 10.

           DISPLAY "UNLOCK-1 FINISHED!" LINE 11 COL 10.
           ACCEPT DUMMY LINE 11 COL 35.

           CLOSE RESWORDS.
           STOP RUN.
```

5.3.47 UNSTRING Statement

The UNSTRING statement separates parts of an existing string that share known delimiters into different data items.

General Format

```
UNSTRING identifier-1
    [ DELIMITED BY [ALL] delimiter-1
    [ OR [ALL] delimiter-2 ] ... ]
```

```
        INTO { identifier-2 [ DELIMITER in delimiter-2 ]
        [ COUNT IN integer-data-1 ] } ...
        [ WITH POINTER integer-data-2 ]
        [ TALLYING IN integer-data-3 ]
        [ ON OVERFLOW statement-1 ]
        [ NOTON OVERFLOW statement-2 ]
        [ END-UNSTRING ]
```

Syntax

1. identifier-n is a literal or data item that is not numeric.
2. delimiter-n is an alphanumeric literal.
3. integer-data-n is a numeric data item designed to hold an integer.
4. statement-n is an imperative statement.

General Rules

1. There is no limit to the number of target identifiers that may be listed.
2. The DELIMITED BY clause provides a way to identify a substring of a identifier-1, for purposes of the UNSTRING operation.
3. The DELIMITED BY delimiter-1 clause causes the UNSTRING operation to stop copying data from identifier-1 when delimiter-1 is encountered. Delimiter-1 is not copied to the target-string.
4. The COUNT phrase counts the number of characters in a substring.
5. The WITH POINTER clause provides a way to set a position in the source string from which the UNSTRING statement will copy the selected source using a pointer-variable.
6. The WITH POINTER clause causes an explicit pointer variable to be referenced by the UNSTRING operation when it begins to copy data from a source string to a target string. The value of the pointer-variable is used as the position in the target-string at which to copy the data.
7. When the WITH POINTER clause is used, the pointer variable must be set to a positive integer value before the UNSTRING statement is executed. Most commonly, the pointer variable is initialized to 1, indicating that the UNSTRING statement should begin parsing the source string from the first byte.
8. The pointer-variable is automatically incremented by 1 for each character that is parsed in the source string.
9. If there is no WITH POINTER clause, the position at which the source string is copied into the next target-string is the first character, or, the next character position after the last delimiter character encountered in the original source string.
10. The TALLY phrase counts the number of substrings that have been created by the UNSTRING statement.
11. The ON OVERFLOW condition is triggered when the pointer variable does not contain a positive value, or when the length of target-string is less than the length of the substring identified in the UNSTRING statement.
12. When an ON OVERFLOW condition exists, the UNSTRING is terminated, and the ON OVERFLOW statement-1 is executed.
13. The NOT ON OVERFLOW condition exists when the UNSTRING statement has completed, and the ON OVERFLOW condition does not exist.
14. When a NOT ON OVERFLOW condition exists, statement-2 is executed.

15. The ALL phrase indicates that a sequence of delimiters should be treated as a singled delimiter.

    Code Sample

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. UNSTRING-1.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  ADDRESS-1
         PIC X(40)  VALUE"231/RUE SAINT-HONORE/75001 PARIS/FRANCE/".
       01  ADDRESS-2.
         05  ADDRESS-NUM        PIC X(5)       VALUE SPACES.
         05  ADDRESS-NAME       PIC X(20)      VALUE SPACES.
         05  ADDRESS-CITY       PIC X(20)      VALUE SPACES.
         05  ADDRESS-COUNTRY    PIC X(20)      VALUE SPACES.
       77 COUNTER-1    PIC 99 VALUE 0.
       77 COUNTER-2    PIC 99 VALUE 0.
       77 COUNTER-3    PIC 99 VALUE 0.
       77 COUNTER-4    PIC 99 VALUE 0.
       77 TALLY-1      PIC 99 VALUE 0.
       01 DUMMY  PIC X.
       PROCEDURE DIVISION.
       MAIN.
           UNSTRING ADDRESS-1 DELIMITED BY "/"
            INTO ADDRESS-NUM, COUNT IN COUNTER-1,
                ADDRESS-NAME, COUNT IN COUNTER-2,
                ADDRESS-CITY, COUNT IN COUNTER-3,
                ADDRESS-COUNTRY, COUNT IN COUNTER-4
                TALLYING IN TALLY-1.

           DISPLAY "THE STREET #:    "  LINE 4 COL 10.
           DISPLAY ADDRESS-NUM LINE 4 COL 28.
           DISPLAY COUNTER-1 LINE 4 COL 50.
      *
           DISPLAY "THE STREET NAME: "  LINE 5 COL 10.
           DISPLAY ADDRESS-NAME LINE 5 COL 28.
           DISPLAY COUNTER-2 LINE 5 COL 50.

           DISPLAY "THE CITY:        "  LINE 6 COL 10.
           DISPLAY ADDRESS-CITY LINE 6 COL 28.
           DISPLAY COUNTER-3 LINE 6 COL 50.

           DISPLAY "THE COUNTRY:     "  LINE 7 COL 10.
           DISPLAY ADDRESS-COUNTRY LINE 7 COL 28.
           DISPLAY COUNTER-4 LINE 7 COL 50.

           DISPLAY "TALLY:           " LINE 9 COL 10.
           DISPLAY TALLY-1             LINE 9 COL 28.

           DISPLAY "UNSTRING-1 FINISHED!" LINE 12 COL 10.

           ACCEPT DUMMY LINE 12 COL 30.

           STOP RUN.
```

5.3.48 USE Statement

The USE statement determines the error handling that takes place inside the DECLARATIVES section of the PROCEDURE DIVISION.

General Format

```
USE [GLOBAL] AFTER STANDARD {EXCEPTION} PROCEDURE ON {{file-1 }... }
                            {ERROR }                 { INPUT  }
                                                     { OUTPUT }
                                                     { I-O    }
                                                     { EXTEND }
```

Syntax

1. file is a data file described in the FILE Section.
2. The USE statement may only be used in the DECLARATIVES section of the PROCEDURE DIVISION.
3. file-n is a file described in the File Section with an FD.

General Rules

1. The USE AFTER STANDARD ERROR PROCEDURE clause includes statements to be executed in the event of file errors if programmatic phrases have not been included to handle the file error conditions inline. Such programmatic phrases include the AT END and INVALID KEY phrases. In each of these cases, the statement lists that are executed when the file error conditions are triggered are listed after the AT END/INVALID KEY phrase.
2. AT END conditions are identified with FILE STATUS conditions whose first byte is “1”. INVALID KEY conditions are identified with FILE STATUS conditions whose first byte is “2”. FILE STATUS conditions whose first byte is “0” are considered to be successful I-O operations.
3. The USE AFTER STANDARD ERROR PROCEDURE clause can be associated with file errors in specific files, as represented by the file-1 notation in the General Format, or with specific I-O types, as represented by the INPUT, OUTPUT, I-O, and EXTEND notations in the General Format.
4. If a specific file is named in the USE clause, then the following statement list is executed when an unsuccessful file operation is returned for that file.
5. After the statement list associated with an unsuccessful file operation has been executed, control returns to the statement after the statement that caused the unsuccessful file operation.
6. Unsuccessful file operations are any file operations that return a value to the FILE STATUS variable described in the SELECT statement of the file, where the first byte of the FILE STATUS variable is greater than zero.
7. If a specific file is named in the USE clause, this takes precedence over the more general association with specific I-O types. That is, if there is a USE AFTER STANDARD ERROR PROCEDURE FOR file-1 statement, and a USE AFTER STANDARD ERROR PROCEDURE FOR INPUT statement, and file-1 is OPEN INPUT and returns a file error, the USE AFTER STANDARD ERROR PROCEDURE FOR file-1 clause will take precedence over the USE AFTER STANDARD ERROR PROCEDURE ON INPUT clause.
8. The USE GLOBAL phrase allows the error handling described by the USE statement to be applicable in all of the programs in the entire compilation unit.
9. EXCEPTION and ERROR are synonyms.
10. The statements that can generate unsuccessful file operations are CLOSE, DELETE, OPEN,

READ, REWRITE, START, UNLOCK, and WRITE.

Code Sample:

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  USE-1
       ENVIRONMENT DIVISION.
       INPUT-OUTPUT SECTION.
           SELECT CUSTFILE ASSIGN TO "CUSTOMER"
           ORGANIZATION IS INDEXED
           ACCESS IS DYNAMIC
           RECORD KEY IS CUSTOMER-ID
           FILE STATUS IS CUSTOMER-STAT.
       DATA DIVISION.
       FILE SECTION.
       FD CUSTFILE.
       01 CUSTOMER-RECORD.
         03 CUSTOMER-ID     PIC 9(5).
         03 CUSTOMER-NAME   PIC X(25).
         03 CUSTOMER-ADDR   PIC X(25).
         03 CUSTOMER-CITY   PIC X(25).
         03 CUSTOMER-STATE  PIC XX.
         03 CUSTOMER-PHONE  PIC X(10).

       WORKING-STORAGE SECTION.
       77 CUSTOMER-STAT PIC XX.
       77 DUMMY             PIC X.

       PROCEDURE DIVISION.
       DECLARATIVES.
       USE-1-ERR-HANDLING SECTION.
           USE AFTER STANDARD ERROR PROCEDURE ON CUSTFILE.
       CUSTFILE-ERR.
           DISPLAY "USE AFTER STANDARD ERROR ON [FILENAME]"
                   LINE 8 COL 10.

           DISPLAY "FILE STATUS: " LINE 10 COL 10.
           DISPLAY CUSTOMER-STAT LINE 10 COL 24.
           ACCEPT DUMMY LINE 10 COL 30.
           DISPLAY "USE-1 FINISHED!" LINE 12 COL 10.
           STOP RUN.
       END DECLARATIVES.
       MAIN.
           OPEN OUTPUT CUSTFILE.
           MOVE 11111 TO CUSTOMER-ID.
           MOVE "JOHN SMITH" TO CUSTOMER-NAME.
           MOVE "101 MAIN ST" TO CUSTOMER-ADDR.
           MOVE "SAN DIEGO" TO CUSTOMER-CITY.
           MOVE "CA" TO CUSTOMER-STATE.
           MOVE "6195551212" TO CUSTOMER-PHONE.
           WRITE CUSTOMER-RECORD.

           MOVE 11111 TO CUSTOMER-ID.
           MOVE "JOHN SMITH" TO CUSTOMER-NAME.
           MOVE "101 MAIN ST" TO CUSTOMER-ADDR.
           MOVE "SAN DIEGO" TO CUSTOMER-CITY.
           MOVE "CA" TO CUSTOMER-STATE.
           MOVE "6195551212" TO CUSTOMER-PHONE.
           WRITE CUSTOMER-RECORD.

           STOP RUN.
```

5.3.49 WRITE Statement

The WRITE statement adds a record to a data file.

Format 1

The Format 1 WRITE statement writes a record to a sequential file, and includes language for writing to a PRINT file, which is a special form of a line sequential file. The

```
WRITE record-name [ FROM identifier-1 ]
     [ {BEFORE} ADVANCING { integer-1 [LINE           ] } ]
       {AFTER }           [LINES          ]
                          [PAGE           ]
                          [ mnemonic-name ]
     [ AT {END-OF-PAGE} statement-1 ]
        {EOP }
     [ NOT AT {END-OF-PAGE} statement-2 ]
           {EOP }
     [ END-WRITE ]
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.
2. integer-n is a data element, literal or data returned from a function call that is an integer.
3. statement-n is an imperative statement.

General Rules

1. The file in which record-name is described in the FILE SECTION must be OPEN when the WRITE statement executes.
2. Record-name is an 01-level record name defined in a File Description (FD).
3. The FROM phrase causes the data in identifier-1to be copied to record-name before the execution of the WRITE statement. Identifier-1 may be in the form of a FUNCTION call.:
4. The ADVANCING phrase should only be used with files described with ORGANIZATION IS LINE SEQUENTIAL.
5. The BEFORE ADVANCING integer-1 LINES phrase causes record-name to be written BEFORE introducing line-feeds to the output record. The number of line-feeds is described with integer-1.
6. The AFTER ADVANCING integer-1 LINES phrase causes record-name to be written AFTER introducing line-feeds to the output record. The number of line-feeds is described with integer-1.
7. The BEFORE ADVANCING integer-1 PAGES phrase causes record-name to be written BEFORE introducing page-feeds to the output record. The number of page-feeds is described with integer-1.
8. The AFTER ADVANCING integer-1 PAGES phrase causes record-name to be written AFTER introducing page-feeds to the output record. The number of page-feeds is described with integer-1.
9. Files that contain a LINAGE clause automatically causes lines feeds to be written to the file

as stipulated in the LINES AT BOTTOM and LINES AT TOP clauses.

10. Files that contain a LINAGE clause and a FOOTING clause trigger the END-OF-PAGE condition if a WRITE statement causes any data to be written into the FOOTING area, as described in the FD.
11. If the END-OF-PAGE condition is trapped by an END-OF-PAGE phrase, the associated statement-list is executed, and then control is passed to the next statement after the WRITE statement.
12. The NOT AT END-OF-PAGE condition exists after a WRITE in files that contain a LINAGE clause, and where the WRITE does not cause the internal counter maintained by the LINAGE clause to be reached. If the NOT AT END-OF-PAGE condition is trapped by a NOT AT END-OF-PAGE phrase, the associated statement-list is executed, and then control is passed to the next statement after the WRITE statement.
13. A successful WRITE statement to a sequential file causes the data in record-name to be written to the end of the file.
14. WRITEing a LINE SEQUENTIAL record to a named pipe is supported.
15. The WRITE statement updates the FILE STATUS variable.

Format 2

The Format 2 WRITE statement writes a record to an indexed file, or a relative file, and includes language for trapping the INVALID KEY condition.

```
WRITE record-name [ FROM identifier-1 ]
       [ WITH [NO] LOCK ]
       [ INVALID KEY statement-1 ]
       [ NOT INVALID KEY statement-2 ]
       [ END-WRITE ]
```

Syntax

1. identifier-n is a data element, literal, or data returned from a function call.
2. statement-n is an imperative statement.

General Rules

1. The file in which record-name is described in the FILE SECTION must be OPEN when the WRITE statement executes.
2. Record-name is an 01-level record name defined in a File Description (FD).
3. The FROM phrase causes the data in identifier-1to be copied to record-name before the execution of the WRITE statement.
4. The WITH LOCK phrase causes the record to be LOCK'ed for the duration of the WRITE statement.
5. The WITH NO LOCK phrase indicates that the record is not LOCK'ed for the duration of the WRITE statement.
6. The INVALID KEY condition exists if a file-status error is generated by the WRITE statement. If the INVALID KEY condition is trapped by an ON INVALID KEY phrase, the associated statement-list is executed, and then control is passed to the next statement after the WRITE statement.

7. Any of the following circumstances triggers the INVALID KEY condition:
   a. A WRITE to a relative file is attempted, and a record with the same RELATIVE KEY already exists.
   b. A WRITE to an indexed file is attempted, and a record with the same RECORD KEY already exists.
   c. A WRITE to an indexed file is attempted, and a record with the same ALTERNATE RECORD KEY already exists, and the ALTERNATE RECORD KEY does not allow duplicates.
8. If the INVALID KEY condition is not trapped by an ON INVALID KEY phrase, it can be trapped in DECLARATIVES with the appropriate USE statement. If it is not trapped in DECLARATIVES, it causes the program to abort.
9. The NOT INVALID KEY condition exists after a WRITE statement is executed successfully. If the NOT INVALID KEY condition is trapped by a NOT ON INVALID KEY phrase, the associated statement-list is executed, and then control is passed to the next statement after the WRITE statement.
10. A successful WRITE statement to a relative file causes the data in record-name to be written to the position in the relative file described by the data item containing the relative key.
11. A successful WRITE statement to an indexed file causes the index, and data portions of the file to be updated.
12. The WRITE statement updates the FILE STATUS variable.

Code Sample

```
...
       SELECT PRINT-FILE-1 ASSIGN TO "PRINTER".

       SELECT IDX-FILE-1 ASSIGN TO "IDX-FILE-1"
       ORGANIZATION IS INDEXED
       ACCESS IS DYNAMIC
       RECORD KEY IS IDX-KEY
       FILE STATUS IS FILE-1-STAT.
...
   FD PRINT-FILE-1.
   01 PRINT-RECORD       PIC X(60).

   FDFILE-1.
   01 FILE-1-RECORD.
     03 FILE-KEY PIC X(10).
...
   77 FILE-1-STAT PIC XX.
   01 WS-IDX-FILE PIC X(10).
...

       WRITE PRINT-RECORD BEFORE ADVANCING 1 LINE.

       WRITE PRINT-RECORD AFTER ADVANCING 1 LINE.

       WRITE PRINT-RECORD AFTER ADVANCING 2 LINES
         AT END-OF-PAGE CONTINUE
         NOT AT END-OF-PAGE CONTINUE
       END-WRITE.

       WRITE PRINT-RECORD AFTER ADVANCING PAGE.

       WRITE PRINT-RECORD BEFORE ADVANCING PAGE.
```

```
           WRITE FILE-1-RECORD.

           WRITE FILE-1-RECORD
             INVALID KEY
               DISPLAY "INVALID KEY!" LINE 10 COL 10
             NOT INVALID KEY
               CONTINUE
           END-WRITE.

           WRITE FILE-1-RECORD FROM WS-IDX-FILE.
```

5.3.50 XML Generate Statement

The XML GENERATE statement converts the data items in identifier-2 into an XML document that is stored in identifier-1.

A converted data element stores the value of the data item, the name of the data element, and XML markup. The element names are derived from the data names in identifier-2.

General Format

```
XML GENERATE identifier-1 FROM identifier-2 [COUNT [IN] identifier-3]
  [[ON] EXCEPTION imperative-statement-1]
  [NOT [ON] EXCEPTION imperative-statement-2]
  [END-XML]
```

Syntax Rules

1. identifier-1 is the data item into which the XML document is generated. identifier-1 must be declared as an elementary or group item of category alphanumeric. identifier-1 must not overlap identifier-2 or identifier-3.
2. identifier-1 must be large enough to hold the XML document. A commonly used guideline is to give identifier-1 a size that is 5-10 times the size of identifier-2. If identifier-1 is too small, the XML GENERATE statement will produce an error condition.
3. identifier-2 is the group or elementary item to be converted into an XML document. It must not overlap identifier-1 or identifier-3.
4. identifier-3 is a numeric data item. It must not overlap identifier-1 or identifier-2.

General Rules

1. identifier-1 must not be described with the JUSTIFIED clause, and cannot be a function identifier.
2. If identifier-1 is larger than the XML document, the trailing bytes will not be altered by the XML Generate operation. Data that was present from a previous GENERATE statement could still be present, and not overwritten. For this reason, it is good practice to INITIALIZE identifier-1 before performing the XML GENERATE statement. Alternatively,

you could use identifier-3 as a reference modifier when referring to the data in identifier-1.

3. identifier-2 cannot be reference-modified, and cannot be described with the RENAMES clause.
4. XML GENERATE will ignore the following in identifier-2:
    a. unnamed elementary data items, or FILLER data items
    b. slack bytes inserted for SYNCRONIZED items
    c. any items described with or subordinate to a REDEFINES clause.
    d. any items decribed with the RENAMES clause,
    e. any group item whose subordinate data items are all ignored.
5. There must be at least one unignored elementary data item in the identifier-2 group item. \
6. Excluding ignored data items, elementary data items in the identifier-2 group item must be CLASS ALPHABETIC, ALPHANUMERIC, NUMERIC, or be an INDEX item.
7. The COUNT IN phrase manages the count of generated XML characters (in bytes) that are stored in identifier-3.
8. identifier-3 must be an integer data item without the symbol "P" in its picture string.

ON EXCEPTION phrase

1. An EXCEPTION condition exists when an error occurs during the generation of the XML document. As an example, if identifier-1 is not large enough to hold the XML document, an EXCEPTION condition will be triggered. When an EXCEPTION condition is triggered, control passes to imperative-statement-1. The contents of identifier-1 are undefined. If a COUNT IN phrase was used, the number of characters generated can be retrieved in identifier-3.

NOT ON EXCEPTION phrase

1. If the ON EXCEPTION phrase is specified, and no EXCEPTION condition is triggered, then control passes to imperative-statement-2.

END-XML phrase

1. The END-XML phrase delimits the scope of both XML GENERATE and XML PARSE statements.

Nested XML statements

1. An XML Generate or XML Parse statement located in an ON EXCEPTION imperative-statement is referred to as a nested (or conditional) XML statement. The scope of a conditional XML GENERATE or XML PARSE statement is terminated by either an END-XM phrase at the same level of nesting, or by a separator period.

Special Registers
XML-CODE

1. The XML-CODE special register indicates whether an XML GENERATE statement executed successfully or an exception occurred during XML generation. A successful

execution of the XML GENERATE statement causes the XML-CODE special register to be set to 0. Exception conditions return non-zero error codes to the XML-CODE special register.

2. The XML-CODE special register is implicitly defineds as:
```
XML-CODE PICTURE S9(9) USAGE BINARY VALUE 0.
```
Details on handling non-zero error codes is detailed in the IBM Enterprise COBOL Programming Guide, in the "Handling XML Parse exceptions" chapter.

Converting data values, and element names
COBOL-IT uses IBM rules for converting elementary data items to character format, for data trimming, and for deriving element names.

Converting elementary data items is the process of translating different data types into a character format. Alphanumeric data values are unchanged. Rules for converting elementary data items are detailed in the IBM Enterprise COBOL Programming Guide.

Data trimming is the application of rules that determine how leading and trailing zeroes, and spaces are handled for a variety of different cases. Rules for applying data trimming are detailed in the IBM Enterprise COBOL Programming Guide.

Element naming is the process of translating element data names in identifier-2 into element tag names in the XML document. In most cases, you will see that your element tag name corresponds exactly to yur element name. However, there are cases where data names may contain characters that are illegal in XML. Rules for element naming are detailed in the IBM Enterprise COBOL Programming Guide.

5.3.51 XML PARSE Statement

The XML PARSE statement parses an XML document and returns data values to their corresponding data elements. This data can then be processed by the COBOL program.

General Format
```
XML PARSE identifier-1
PROCESSING PROCEDURE [IS]
        procedure-name-1 [THROUGH procedure-name-2]
                          THRU
  [[ON] EXCEPTION imperative-statement-1]
  [NOT [ON] EXCEPTION imperative-statement-2]
  [END-XML]
```

Syntax Rules

1. identifier-1 is an alphanumeric data item that holds an XML document.
2. The PROCESSING PROCEDURE phrase specifies the name of a procedure to respond to the events generated by the XML PARSE statement.
3. procedure-name-1 and procedure-name-2 define a consecutive sequence of operations to

execute, beginning at the procedure named by procedure-name-1 and ending with the execution of the procedure named by procedure-name-2.

4. procedure-name-1, procedure-name-2 indicates a section or paragraph in the PROCEDURE DIVISION. Procedure-name-1 and procedure-name-2 may not be in a declarative section.

General Rules

1. Control passes between the parser and procedure-name1. The parser generates an XML event, procedure-name-1 (or the code executed in procedure-name-1 through procedure-name 2) handles the event, and control is then returned to the parser.
2. The processing procedure interprets the information returned from the parser, and handles it programmatically. When the processing procedure (or procedures) is complete control is returned to the XML parser.
3. The processing procedure can terminate the run unit with a STOP RUN statement.

ON EXCEPTION phrase

1. An EXCEPTION condition exists when an error occurs during the XML PARSE operation. The parser signals the exception by passing control to the processing procedure and updating XML-EVENT and XML-CODE special registers. If the ON EXCEPTION phrase is specified, control passes to imperative-statement-1. For details on the XML-EVENT and XML-CODE special registers, see your IBM documentation.

NOT ON EXCEPTION phrase

1. If the ON EXCEPTION phrase is specified, and no EXCEPTION condition is triggered, then control passes to imperative-statement-2.

END-XML phrase

1. The END-XML phrase delimits the scope of both XML GENERATE and XML PARSE statements.

Nested XML statements

1. An XML Generate or XML Parse statement located in an ON EXCEPTION imperative-statement is referred to as a nested (or conditional) XML statement. The scope of a conditional XML GENERATE or XML PARSE statement is terminated by either an END-XML phrase at the same level of nesting, or by a separator period.

Special Registers

1. In the control flow between the parser and the processing procedure, the parser returns control to the processing procedure, and updates the special registers XML-CODE, XML-EVENT, and XML-TEXT.

XML-CODE

1. The XML-CODE special register is implicitly defineds as:

   01 XML-CODE PICTURE S9(9) USAGE BINARY VALUE 0.

2. The XML-CODE special register communicates status between the XML parser and the processing procedure defined in an XML PARSE statement. The XML-CODE special register also indicates whether an XML GENERATE statement executed successfully or an exception occurred during XML generation.
3. When an XML PARSE statement terminates, the special register XML-CODE is populated either with a "0" to indicate a successful operation, or with a non-zero error code.
4. Details on handling non-zero error codes is detailed in the IBM Enterprise COBOL Programming Guide, in the "Handling XML Parse exceptions" chapter.

XML-EVENT

1. The XML_EVENT special register is implicitly defined as:

   01 XML-EVENT USAGE DISPLAY PICTURE X(30) VALUE SPACE.

2. The XML-EVENT special register communicates event information from the XML parser to the processing procedure defined in an XML PARSE statement.
3. The XML-EVENT special register is set to the name of the XML event.
4. XML events and associated special register contents are listed in the IBM COBOL Enterprise COBOL Programming Guide.

XML-TEXT

1. XML-TEXT is an elementary alphanumeric data item of the length of the contained XML document fragment. The length of XML-TEXT can vary from 0 to 16,777,215 bytes. There is no equivalent COBOL data description entry.
2. The XML-TEXT special register contains document fragments that are of class alphanumeric.
3. The XML parser sets XML-TEXT to the document fragment associated with an XML-EVENT before transferring control to the processing procedure when the operand of the XML PARSE statement is an alphanumeric data item. Use the LENGTH function for XML-TEXT to determine the number of bytes that XML-TEXT contains.
4. XML-TEXT cannot be used as a receiving item.
5. XML events and associated XML-TEXT contents are listed in the IBM COBOL Enterprise COBOL Programming Guide.

www.cobol-it.com

June, 2020