COBOL - TABLE PROCESSING

Arrays in COBOL are known as tables. An array is a linear data structure and is collection of individual data items of same type. Data items of a table are internally sorted.

Table Declaration

Table is declared in Data Division. **Occurs** clause is used to define a table. Occurs clause indicates the repetition of data name definition. It can be used only with level numbers starting from 02 to 49. Do not use occurs clause with Redefines. Description of one-dimensional and two-dimensional table is as follows:

One-Dimensional Table

In a one-dimensional table, **occurs** clause is used only once in declaration. WS-TABLE is the group item that contains table. WS-B names the table elements that occur 10 times.

Syntax

Following is the syntax for defining a one-dimensional table:

```
01 WS-TABLE.
   05 WS-A PIC A(10) OCCURS 10 TIMES.
```

Example

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.

DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 WS-TABLE.
      05 WS-A PIC A(10) VALUE 'TUTORIALS' OCCURS 5 TIMES.

PROCEDURE DIVISION.
   DISPLAY "ONE-D TABLE : "WS-TABLE.
STOP RUN.
```

JCL to execute the above COBOL program:

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS=A,MSGCLASS=C
//STEP1 EXEC PGM=HELLO
```

When you compile and execute the above program, it produces the following result:

```
ONE-D TABLE : TUTORIALS TUTORIALS TUTORIALS TUTORIALS TUTORIALS
```

Two-Dimensional Table

A two dimensional table is created with both data elements being variable length. For reference, go through the syntax and then try to analyze the table. The first array$WS-A$ can occur from 1 to 10 times and the inner array$WS-C$ can occur from 1 to 5 times. For each entry of WS-A, there will be corresponding 5 entries of WS-C.

Syntax

Following is the syntax for defining a two-dimensional table:

```
01 WS-TABLE.
   05 WS-A OCCURS 10 TIMES.
```

```
      10 WS-B PIC A(10).
      10 WS-C OCCURS 5 TIMES.
         15 WS-D PIC X(6).
```

Example

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.

DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 WS-TABLE.
      05 WS-A OCCURS 2 TIMES.
         10 WS-B PIC A(10) VALUE ' TUTORIALS'.
         10 WS-C OCCURS 2 TIMES.
            15 WS-D PIC X(6) VALUE ' POINT'.

PROCEDURE DIVISION.
   DISPLAY "TWO-D TABLE : "WS-TABLE.

STOP RUN.
```

JCL to execute the above COBOL program:

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS=A,MSGCLASS=C
//STEP1 EXEC PGM=HELLO
```

When you compile and execute the above program, it produces the following result:

```
TWO-D TABLE :  TUTORIALS POINT POINT TUTORIALS POINT POINT
```

Subscript

Table individual elements can be accessed by using subscript. Subscript valus can range from 1 to the number of times the table occurs. Subscript can be any positive number. It does not require any declaration in data division. It is automatically created with occurs clause.

Example

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.

DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 WS-TABLE.
      05 WS-A OCCURS 3 TIMES.
         10 WS-B PIC A(2).
         10 WS-C OCCURS 2 TIMES.
            15 WS-D PIC X(3).

PROCEDURE DIVISION.
   MOVE '12ABCDEF34GHIJKL56MNOPQR' TO WS-TABLE.
   DISPLAY 'WS-TABLE  : ' WS-TABLE.
   DISPLAY 'WS-A(1)   : ' WS-A(1).
   DISPLAY 'WS-C(1,1) : ' WS-C(1,1).
   DISPLAY 'WS-C(1,2) : ' WS-C(1,2).
   DISPLAY 'WS-A(2)   : ' WS-A(2).
   DISPLAY 'WS-C(2,1) : ' WS-C(2,1).
   DISPLAY 'WS-C(2,2) : ' WS-C(2,2).
   DISPLAY 'WS-A(3)   : ' WS-A(3).
   DISPLAY 'WS-C(3,1) : ' WS-C(3,1).
   DISPLAY 'WS-C(3,2) : ' WS-C(3,2).

STOP RUN.
```

JCL to execute the above COBOL program.

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS=A,MSGCLASS=C
//STEP1 EXEC PGM=HELLO
```

When you compile and execute the above program, it produces the following result:

```
WS-TABLE  : 12ABCDEF34GHIJKL56MNOPQR
WS-A(1)   : 12ABCDEF
WS-C(1,1) : ABC
WS-C(1,2) : DEF
WS-A(2)   : 34GHIJKL
WS-C(2,1) : GHI
WS-C(2,2) : JKL
WS-A(3)   : 56MNOPQR
WS-C(3,1) : MNO
WS-C(3,2) : PQR
```

Index

Table elements can also be accessed using index. An index is a displacement of element from the start of the table. An index is declared with Occurs clause using INDEXED BY clause. The value of index can be changed using SET statement and PERFORM Varying option.

Syntax

Following is the syntax for defining Index in a table:

```
01 WS-TABLE.
   05 WS-A PIC A(10) OCCURS 10 TIMES INDEXED BY I.
```

Example

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.

DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 WS-TABLE.
      05 WS-A OCCURS 3 TIMES INDEXED BY I.
         10 WS-B PIC A(2).
         10 WS-C OCCURS 2 TIMES INDEXED BY J.
            15 WS-D PIC X(3).

PROCEDURE DIVISION.
   MOVE '12ABCDEF34GHIJKL56MNOPQR' TO WS-TABLE.
   PERFORM A-PARA VARYING I FROM 1 BY 1 UNTIL I >3
   STOP RUN.

   A-PARA.
   PERFORM C-PARA VARYING J FROM 1 BY 1 UNTIL J>2.

   C-PARA.
   DISPLAY WS-C(I,J).
```

JCL to execute the above COBOL program.

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS=A,MSGCLASS=C
//STEP1 EXEC PGM=HELLO
```

When you compile and execute the above program, it produces the following result:

```
ABC
DEF
GHI
JKL
```

```
MNO
PQR
```

Set Statement

Set statement is used to change the index value. Set verb is used to initialize, increment or decrement the index value. It is used with Search and Search All to locate elements in table.

Syntax

Following is the syntax for using a Set statement:

```
SET I J TO positive-number
SET I TO J
SET I TO 5
SET I J UP BY 1
SET J DOWN BY 5
```

Example

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.

DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 WS-TABLE.
      05 WS-A OCCURS 3 TIMES INDEXED BY I.
         10 WS-B PIC A(2).
         10 WS-C OCCURS 2 TIMES INDEXED BY J.
            15 WS-D PIC X(3).

PROCEDURE DIVISION.
   MOVE '12ABCDEF34GHIJKL56MNOPQR' TO WS-TABLE.
   SET I J TO 1.
   DISPLAY WS-C(I,J).
   SET I J UP BY 1.
   DISPLAY WS-C(I,J).

STOP RUN.
```

JCL to execute the above COBOL program.

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS=A,MSGCLASS=C
//STEP1 EXEC PGM=HELLO
```

When you compile and execute the above program, it produces the following result:

```
ABC
JKL
```

Search

Search is a linear search method, which is used to find elements inside the table. It can be performed on sorted as well as unsorted table. It is used only for tables declared by Index phrase. It starts with the initial value of index. If the searched element is not found, then the index is automatically incremented by 1 and it continues till the end of table.

Example

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.

DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 WS-TABLE.
```

```
      05 WS-A PIC X(1) OCCURS 18 TIMES INDEXED BY I.
   01 WS-SRCH PIC A(1) VALUE 'M'.

PROCEDURE DIVISION.
   MOVE 'ABCDEFGHIJKLMNOPQR' TO WS-TABLE.
   SET I TO 1.
   SEARCH WS-A
     AT END DISPLAY 'M NOT FOUND IN TABLE'
     WHEN WS-A(I)=WS-SRCH
     DISPLAY 'LETTER M FOUND IN TABLE'
   END-SEARCH.

STOP RUN.
```

JCL to execute the above COBOL program.

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS=A,MSGCLASS=C
//STEP1 EXEC PGM=HELLO
```

When you compile and execute the above program, it produces the following result:

```
LETTER M FOUND IN TABLE
```

Search All

Search All is a binary search method, which is used to find elements inside the table. Table must be in sorted order for Search All option. The index does not require initialization. In binary search the table is divided into two halves and determines in which half the searched element is present. This process repeats till the element is found or the end is reached.

Example

```
IDENTIFICATION DIVISION.
PROGRAM-ID. HELLO.

DATA DIVISION.
   WORKING-STORAGE SECTION.
   01 WS-TABLE.
      05 WS-RECORD OCCURS 10 TIMES ASCENDING KEY IS WS-NUM INDEXED BY I.
      10 WS-NUM PIC 9(2).
      10 WS-NAME PIC A(3).

PROCEDURE DIVISION.
   MOVE '12ABC56DEF34GHI78JKL93MNO11PQR' TO WS-TABLE.
   SEARCH ALL WS-RECORD
     AT END DISPLAY 'RECORD NOT FOUND'
     WHEN WS-NUM(I)=93
     DISPLAY 'RECORD FOUND '
     DISPLAY WS-NUM(I)
     DISPLAY WS-NAME(I)

END-SEARCH.
```

JCL to execute the above COBOL program:

```
//SAMPLE JOB(TESTJCL,XXXXXX),CLASS=A,MSGCLASS=C
//STEP1 EXEC PGM=HELLO
```

When you compile and execute the above program, it produces the following result:

```
RECORD FOUND
93
MNO
```