MULTIPLE DIMENSIONAL TABLES

. Up to seven levels of occurs in Cobol 85 are permitted.
. Three levels in COBOL 74 and earlier versions.
. Use multiple level tables for accumulating totals in arrays or storing multi-dimensional tables in a program.

A. Defining a Two-Dimensional Array.

Consider an array of temperatures taken hourly for one week.

We could represent this in working storage as:

```
01      TEMPERATURE-ARRAY-1.
        05  DAY-IN-WEEK   OCCURS 7 TIMES.
                10      1AM-TEMP            PIC 999.
                10      2AM-TEMP            PIC 999
                ...     ...                 ...
                10      MIDNIGHT            PIC 999.
```

But, what about if there were readings taken every 15 minutes: We'd need a table with 96 entries! We have a better way to do this:

```
01      Temperature-Array-1.
        05  Day-In-Week                    Occurs 7 TIMES.
                10      Hour                Occurs 24 TIMES.
                        15      Mean-Temp   Pic 999.
```

This data structure represents a 7 x 24 array: 7 rows and 24 columns
==> 168 elements in the array; 504 bytes stored.

For each day in the week, there are 24 column entries: (1,1), (1,2), ... (1,24)

Access?

To access any specific temperatures, we need to access the element by specifying both its row and its column entry.

Therefore, Mean-Temp(4,16) refers to day 4, 4pm sample;
Mean-Temp(7,2) refers to Day 7, 2 am sample...

We may write this as:

```
01      Temperature-Array-1.
        05  Day-In-Week                                Occurs  7 TIMES.
              10                          Hour                  Occurs 24 TIMES    PIC
                                          999.
```

and reference the array elements as Hour(x,y).

May be written this way, but "Mean-Temp" is more descriptive.

So it is ok too and means the same thing.

Use of Subscripts

First subscript = row (higher level occurs) and

Second subscript = column (lower level occurs) - "major and minor occurs"

Range: for a 7 x 24 array:
subscript 1 (row) can vary from 1 to 7, while
subscript 2 (column) can vary from 1 to 24.

Any extension yields "subscript out of range" and an abnormal end (abend).

Remember:

. to access any specific entry, you need both subscripts.

. subscripts must be enclosed in parentheses

. subscripts must be positive (not zero) integers or integer variables having positive values.

. generally: the reference is: arrayb(n1,bn2)b

Accessing a 2-dimensional array:

(different from book)

print average temperature for a day and a week:

```
.....
600-average-rtne.
        move 0 to weektotal
        perform 700-loop-on-days varying day-sub from 1 by 1 until day-sub > 7.
        compute weekly-average = weektotal/168
        write printrec ...   (with week average in it)
700-loop-on-days.
        move 0 to daytotal
        perform 800-loop-on-hours varying hour-sub from 1 by 1 until hour-sub > 24.
        compute day-average = daytotal/24
        write printrec ... (with day-average in it)...
800-loop-on-hours.
        add mean-temp (day-sub,hour-sub) to daytotal, weektotal.
```

Using in-line performs:

```
        move 0 to weektotal
        perform varying day-sub from 1 by 1 until day-sub>7
                move 0 to daytotal
                perform varying hour-sub from 1 by 1 until hour-sub>24.
                        add mean-temp(day-sub,hour-sub) to daytotal, weektotal
                end-perform
                compute dayaverage = daytotal/24
                write printrec.....
        end-perform
        compute weekly-average = weektotal/168
        write print-rec .....
```

Note the indentation!!!

How about the perform...varying...after?

Convenient when varying two nested subscripts with a single perform:

Varies minor subscript more rapidly than the major subscript.

So: perform p1 varying x from 1 by 1 until x > n1
after y from 1 by 1 until y > n2.

Consider a population array: