

Chapter 12

Array Processing and Table Handling

Chapter Objectives

To familiarize you with

- How to establish a series of items using OCCURS clause
- How to access and manipulate data stored in an array or table
- Rules for using OCCURS clause in DATA DIVISION
- Use of SEARCH or SEARCH ALL for table look-up

Why OCCURS Clauses Used

- To indicate repeated occurrence of fields with same format
- Defines series of related fields with same format as an array or table

Uses of OCCURS Clause

- Defining series of input or output fields, each with same format
- Defining series of totals to which amounts added
- Defining a table to be accessed using contents of input field to 'look up' required data

Defining Series of Input Fields

- Suppose 72-character input record consists of 24 hourly temperature fields
- Each field is three positions long
- Fields represent temperature for given city at particular hour

Defining Series of Input Fields

- Coding record with 24 independent hourly fields is cumbersome

01 Temp-Rec.

05 One-AM Pic S9(3).

05 Two-AM Pic S9(3).

... ..

05 Midnight Pic S9(3).

24 entries

Defining Series of Input Fields

- To obtain average temperature requires summing 24 fields

Compute Avg-Temp = (One-AM +
Two-AM + ... + Midnight) / 24

Defining Fields with OCCURS

- Since all 24 fields have same PICTURE
 - Can define entire 72-position area as array
 - Array divided into 24 three-position fields, called elements

01 Temp-Rec.

05 Temperature Occurs 24 Times
Pic S9(3).

Accessing Elements in Array

- Identifier Temperature is array name
- Use array name along with a subscript to access fields or elements within array
- Subscript indicates which of the 24 elements to access

Statement

Output

Display Temperature (2)

2 AM value

Display Temperature (23)

11 PM value

Valid Subscripts

- Valid values are 1 to number of elements in array
- For array Temperature valid subscripts are 1 to 24
- Invalid use of subscripts
 - Display Temperature (0)
 - Display Temperature (25)

Subscripts

- May be integers or numeric fields with integer value
- If field Sub defined in Working-Storage:
05 Sub Pic 99 Value 5.
- To display 5 AM temperature:
Display Temperature (Sub)

Subscripts

- Using a data-name as a subscript enables its contents to be varied
- Each time the value of a data-name changes, Temperature (Sub) refers to a different element in the array
- Then a single routine can be used to process all elements in array

Processing Elements in Array

Example: Find average daily temperature

- Use loop repeated 24 times
- Each time through loop add one temperature to array
- Vary contents of a subscript from 1 to 24 so that all 24 temperatures are added
- Standard or in-line PERFORM UNTIL or PERFORM VARYING may be used

In-line PERFORM UNTIL

Move 1 to Sub

Move Zeros to Total-Temp

Perform Until Sub > 24

 Add Temperature (Sub) To Total-Temp

 Add 1 to Sub

End-Perform

Compute Avg-Temp = Total-Temp / 24

In-line PERFORM VARYING

Move Zeros to Total-Temp

Perform Varying Sub

From 1 By 1 Until Sub > 24

Add Temperature (Sub) To Total-Temp

End-Perform

Compute Avg-Temp = Total-Temp / 24

Relative Subscripting

- Integer literal or data-name used as subscript may be modified within parentheses

<u>Statement</u>	<u>Output</u>
Move 3 To Sub	
Display Temperature (Sub + 1)	4 AM value
Display Temperature (Sub - 2)	1 AM value

Debugging Tip

- Define subscript large enough to hold values to reference all elements
 - Subscript for array of 100 elements should be Pic 9(3), not Pic 9(2)
- Define subscript large enough to store value one more than upper subscript limit
 - Pic 99 needed to allow for number 10 to exit loop with condition `Sub > 9`

Using OCCURS for Totals

- Define array to hold 12 monthly totals

Working-Storage Section.

01 Totals.

05 Mo-Tot Occurs 12 Times

Pic 9(5)V99.

Initialize Array of Totals

- Use VALUE clause (Value Zeros) after PIC clause in OCCURS level entry
- Three ways in Procedure Division
 1. INITIALIZE (Initialize Totals)
 2. MOVE (Move Zeros To Totals)
 3. Perform Varying Sub1
 - From 1 By 1 Until Sub1 > 12
 - Move Zeros to Mo-Tot (Sub1)
 - End-Perform

Add Value to Array Totals

- Assume input record with transactions for same year
- Field Month-In determines Mo-Tot to which contents of input field Amt-In is to be added
 - For example, if Month-In is 3, Amt-In should be added to third Mo-Tot element

Add Value to Array Totals

- For each record read in, if month valid, add amount to corresponding total

200-Calc-Rtn.

 If Month-In \geq 1 And \leq 12

 Add Amt-In To Mo-Tot (Month-In)

 Else

 Perform 400-Err-Rtn

 End-If

Print Array of Totals

- Print array of totals after all input read
 - Move each array entry to output area
 - Write a line

Print Array of Totals

Perform Varying Sub From 1 By 1

Until Sub > 12

Move Mo-Tot (Sub) To Mo-Tot-Out

Write Pr-Rec From Mo-Tot-Line

After Advancing 2 Lines

End-Perform

Elementary Items with OCCURS

- If item defined by OCCURS has PIC clause, it defines a series of elementary items

01 Totals.

05 Mo-Tot Occurs 12 Times Pic 9(5)V99.

- Defines Totals as 84-byte array (12 x 7) of 12 elementary items

Group Items with OCCURS

- Identifier used with OCCURS may also be group item

01 Tax-Table.

05 Group-X Occurs 20 Times.

10 City Pic X(6).

10 Tax-Rate Pic V999.

- City and Tax-Rate each occur 20 times in group item Group-X.

Initializing Elements

- Two ways to use VALUE clause to initialize all elements to zero
 1. 01 Array-1 Value Zero.
05 Totals Occurs 50 Times Pic 9(5).
 2. 01 Array-1.
05 Totals Occurs 50 Times Pic 9(5)
Value Zero.

Initializing Elements

- Can also initialize each element to different value

01 Day-Names

Value 'SUNMONTUEWEDTHUFRISAT'.

05 Days Occurs 7 Times Pic X(3).

- Entries define 21-character array with 7 three-position fields - Days(1) = SUN, Days(2) = MON, etc.

Tables

- Table is list of stored fields
- Stored same way as array but used for different purpose
- Used with table look-ups, a procedure to find specific entry in a table

Data for Table

- Data often read in from separate file
- Stored in `WORKING-STORAGE` table
- Suppose file contains records with two fields, zip code and sales tax rate
- Read in records, storing each zip code and sales tax rate in element in table

Data for Table

- After data stored in table, read in input records from customer transaction file
- Look up a customer's zip code in table to find corresponding sales tax rate
- More efficient to store tax rates in table file than in each transaction record
 - Minimizes data entry operations
 - Easier to maintain or update in table file

Table Look-Up Terms

- Table argument (zip code) is table entry field used to locate desired element
- Table function (sales tax rate) is table entry field to be used when match found
- Search argument (zip code in transaction record) is input field used to find a match

Table Look-Up

Table entries in WORKING-STORAGE

	<u>Table Argument</u>	<u>Table Function</u>		
	<u>Zip Code</u>	<u>Sales Tax Rate</u>		
Input Record	00123	^060		
<table border="1"><tr><td></td><td>12344</td></tr></table>		12344	00456	^075
	12344			
Zip Code	10111	^065		
(search	→12344	^080 ← Rate for Zip of 12344		
argument)	25033	^070		
		

Looking Up Data in Table

- Compare search argument (zip code in transaction record) to each table argument (zip code in table) until match is found
- When table and search arguments match, use corresponding sales tax rate (table function) with same subscript as table's zip code to calculate sales tax

Table Look-up with PERFORM

Move 0 To WS-Sales-Tax

Perform Varying X1 From 1 By 1

Until X1 > 1000

If Zip-In = WS-ZipCode (X1)

 Compute WS-Sales-Tax Rounded =
 WS-Tax-Rate (X1) *

 Unit-Price-In * Qty-In

End-If

End-Perform

SEARCH Statement

Format

SEARCH identifier-1

[AT END imperative-statement-1]

WHEN condition-1

{ imperative-statement-2 } ...
CONTINUE

[END-SEARCH]

- Use in place of PERFORM to search table

SEARCH Statement Example

Set X1 To 1

Search Table-Entries

At End Move 0 To WS-Sales-Tax

When Zip-In = WS-ZipCode (X1)

Compute WS-Sales-Tax Rounded =

WS-Tax-Rate (X1) *

Unit-Price-In * Qty-In

End-Search

SEARCH Statement

- Identifier used after SEARCH is table name specified in OCCURS entry
- Condition compares search argument to table argument
- WHEN clause indicates action to take when condition is met
- AT END clause specifies action to take if table searched but no match found

INDEXED BY clause

- Special field called index must be used with SEARCH
- Similar to subscript but defined along with table as part of OCCURS

05 Table-Entries Occurs 1000 Times

Indexed By X1. ← X1 defined as index

- Compiler automatically supplies appropriate PICTURE clause for index

Index with SEARCH

- Must initialize index before SEARCH
- SEARCH performs table look-up, automatically incrementing index
- Internally, computer can use faster method to access table entries with an index than with a subscript, even when SEARCH not used
- Both can have values from 1 to number of table elements

Modifying Index

- PERFORM ... VARYING can modify subscript or index
- SET is only other statement that can modify index

Format

SET index-name-1 { TO
UP BY
DOWN BY } integer-1

SET Statement Examples

- SET options used to initialize, increment or decrement index value
- Assume $X1 = 3$ before each statement

<u>Statement</u>	<u>Value of X1 after SET</u>
Set X1 To 1	1
Set X1 Up By 2	5
Set X1 Down By 1	2

Subscripts vs Indexes

- Subscript
 - Represents occurrence of array or table element
- Index
 - Represents value used internally to actually access table entry (a displacement from first address in array or table)

Subscripts vs Indexes

- Subscript
 - Defined in separate WORKING-STORAGE entry
 - May be used any where field with its PICTURE is allowed
- Index
 - Defined along with OCCURS
 - May be used only with table for which it was defined

Subscripts vs Indexes

- Subscript
 - Value may be changed using PERFORM ... VARYING
 - Also by MOVE or arithmetic statements
- Index
 - Value may be changed using PERFORM ... VARYING
 - SET only other statement to modify index

Serial Search

Each entry (usually starting with first) checked in order until

- Condition is met
- Table completely searched

Serial Search

Best used when

- Entries not in order by table argument value (not in numerical or alphabetical order)
- Entries can be organized so first values are ones searched for most frequently, minimizing search time

Binary Search

- Most efficient type of look-up when table entries in sequence by some table field
- On average, takes fewer comparisons to find match than serial search
- Called binary search because each comparison eliminates half of entries under consideration

Binary Search Example

- Assume a table contains 50 customer numbers in ascending sequence
- Search table to find match for customer number 5000 stored in input field Cust-No-In

Binary Search Example

Comparison	Entry #	T-Customer-No
	1.	0100
	2.	0200

1st	25.	4300

3rd	31.	4890

4th	34.	5000

2nd	37.	5310

Binary Search Method

- Compare Cust-No-In to middle table argument for T-Customer-In (25th element)
- $\text{Cust-No-In} > \text{T-Customer-No (25)}$ or $5000 > 4300$
 - First half of table eliminated from search since entries are in ascending order

Binary Search Method

- Compare Cust-No-In to middle table argument in top of table
 - Entry 37 is middle of remaining entries 26-50
- Continue until
 - Match is found
 - Table search complete and no match found

Binary Search Method

- Serial search would require 34 comparisons in this example
- Binary search requires only four
- Use binary search for large table (50 or more entries)
- Table entries must be arranged in sequence by some table field

Binary Search Statement

Format (partial)

SEARCH ALL identifier-1

[AT END imperative-statement-1]

WHEN { data-name-1 = { identifier-2
literal-1
arithmetic-expression-1 } }
condition-1

{ imperative-statement-2
CONTINUE }

[END-SEARCH]

SEARCH ALL Limitations

- Condition in *WHEN* can test only for equality between table and search argument
- Condition following *WHEN* may be compound
 - Only *ANDs* permitted, not *Ors*
 - Each relational test can test only for equality

SEARCH ALL Limitations

- Only one *WHEN* clause can be used
- *VARYING* option may not be used
- Table argument and its index must appear to left of equal sign
 - Valid:
When T-Customer-No (X1) = Cust-No-In
 - Invalid:
When Cust-No-In = T-Customer-No (X1)

Key Field

- Must include clause to indicate which table entry serves as key field
- Must specify whether KEY is
 - ASCENDING KEY - entries in sequence, increasing in value
 - DESCENDING KEY - entries in sequence, decreasing in value

Multiple-Level OCCURS

- Up to seven levels of OCCURS permitted in COBOL
- For example, define an array to store hourly temperature readings for each day during a given week
 - Need two-dimensional array with 7 rows, each with 24 columns

Multiple-Level OCCURS

- Define array as follows:

01 Temperature-Array.

05 Day-In-Week Occurs 7 Times.

10 Hour Occurs 24 Times.

15 Temp Pic S9(3).

- For each Day-In-Week there are 24 Hour figures, each consisting of a Temp three integers long

Double-Level Subscripts

- To access temperatures, use data-name on lowest OCCURS level or any data-name subordinate to it
 - Either Temp or Hour could be used
- Since Temp defined with two OCCURS, two subscripts must be used to access each hourly temperature
- For example, Temp (3, 5) refers to the temperature for the third day, fifth hour

Table with Subscripts

Temperature Array						
Hour	1 AM	2 AM	3 AM	4 AM	...	12 Mid
Day-In-Week						
Day 1 (Sun)	(1,1)	(1,2)	(1,3)	(1,4)	...	(1,24)
Day 2 (Mon)	(2,1)	(2,2)	(2,3)	(2,4)	...	(2,24)
Day 3 (Tue)	(3,1)	(3,2)	(3,3)	(3,4)	...	(3,24)
Day 4 (Wed)	(4,1)	(4,2)	(4,3)	(4,4)	...	(4,24)
...
Day 7 (Sat)	(7,1)	(7,2)	(7,3)	(7,4)	...	(7,24)

Accessing Double-Level Array

- Find average temperature for entire week
- Add all array entries and divide by 168 (7 x 24)
- Use nested PERFORMs
 - First PERFORM varies major (row) subscript from 1 to 7
 - Second PERFORM varies minor (column) subscript from 1 to 24

Accessing Double-Level Array

Move 0 To Total

Perform Varying Day-Sub From 1 By 1
Until Day-Sub > 7

Perform Varying Hour-Sub From 1 By 1
Until Hour-Sub > 24

Add Temp (Day-Sub, Hour-Sub)
To Total

End-Perform

End-Perform

Compute Weekly-Average = Total / 168

PERFORM ... VARYING ... AFTER

- Use in place of multiple nested PERFORM ... VARYING statements
- VARYING clause varies major subscript
- AFTER clause varies minor subscript
- Requires procedure name after PERFORM

PERFORM ... VARYING ... AFTER

Move 0 To Total

Perform 700-Loop1

 Varying Day-Sub From 1 By 1

 Until Day-Sub > 7

 After Hour-Sub From 1 By 1

 Until Hour-Sub > 24

 Compute Weekly-Average = Total / 168.

700-Loop1.

 Add Temp (Day-Sub, Hour-Sub)
 To Total.

PERFORM ... VARYING ... AFTER

- Sequence of values for subscripts is
(1, 1), (1, 2) ... (1, 24), (2, 1), (2, 2) ...
(2, 24) ... (7, 1) ... (7, 24)
- Day-Sub initialized to 1 and Hour-Sub varied from 1 to 24
- Then Day-Sub incremented to 2 and Hour-Sub varied from 1 to 24 and so on

Chapter Summary

- OCCURS clause used to specify repeated occurrence of items with same format
 - Use in 02-49 level entries
 - May use with elementary or group item
 - COBOL permits seven levels of OCCURS
- OCCURS defines
 - Array: area used for storing data or totals
 - Table: set of fields used in table look-up

Chapter Summary

- SEARCH statement used for table handling
 - Identifier used with SEARCH is data-name used on OCCURS level
 - AT END clause specifies action if table searched but condition not met
 - WHEN clause indicates what to do when condition met

Chapter Summary

- SEARCH statement
 - Index defined along with OCCURS and used by SEARCH
 - Use SET or PERFORM ... VARYING to change value of index
 - SET index to 1 before using SEARCH

Chapter Summary

- SEARCH ALL statement
 - Used to perform binary search
 - Can test only an equal condition
 - Can use compound condition with ANDs
 - Only one WHEN clause can be used
 - Define index to use in searching table
 - Include ASCENDING or DESCENDING KEY clause in table definition

Chapter Summary

- Multiple-Level OCCURS
 - Used with array or table
 - Lowest-level OCCURS data-name or item subordinate to it used to access table entry
 - INDEXED BY must be included on all OCCURRS levels if SEARCH used
 - Identifier used with SEARCH typically one on lowest OCCURS level
 - Only index on same level incremented