Cobol Tutorial

Using Tables

Introduction

Aims

In most programming languages the term "array" is used to describe repeated, or multiple- occurrence, data-items. COBOL uses the term - "table".

In this tutorial you will discover why a table, or array, is a useful structure for solving certain kinds of problems.

Objectives

By the end of this unit you should -

1. Understand why you might want to use a table as part of your solution to a programming problem
2. Understand how a numeric field in a record can be used as an index or subscript for a table..
3. Understand how the subscript of one table can be used an index into another.

Prerequisites

Introduction to COBOL

Declaring data in COBOL

Basic Procedure Division Commands

To top of page

Why use tables?

Introduction

Let's start our discussion of tables and table handling, by looking at a programming problem.

Suppose we are asked to write a program to sum the taxes paid by people all over the country. The data is obtained from a file containing the amount paid in taxes by PAYE workers all over the country. The TaxFile is a sequential file sequenced on ascending PAYENum and its records have the following description.

```
01 TaxRec.
   88 EndOfTaxFile  VALUE HIGH-VALUES.
   02 PAYENum    PIC 9(8)
   02 CountyCode PIC 99.
   02 TaxPaid    PIC 9(7)V99.
```

The program to perform this task is very simple. All we have to do is to set up a variable to hold the tax total and then add the TaxPaid from each record to the TaxTotal. The program to do this is shown below;

Program Fragment

```
PROCEDURE DIVISION.
Begin.
   OPEN INPUT TaxFile
   READ TaxFile
      AT END SET EndOfTaxFile TO TRUE
   END-READ
   PERFORM UNTIL EndOfTaxFile
      ADD TaxPaid TO TaxTotal
      READ TaxFile
         AT END SET EndOfTaxFile TO TRUE
      END-READ
   END-PERFORM.
   DISPLAY "Total taxes are ", TaxTotal
   CLOSE TaxFile
   STOP RUN.
```

But what if, instead of just calculating the tax total for the whole country, we were asked to produce a breakdown of the tax totals by county. How could we solve that problem?

Exploring the problem

One approach to the problem of producing the County Tax Report would be to sort the file on CountyCode. This would turn the problem into a simple control break problem (i.e. process all the records for one county, output the result and then go on to the next). But the TaxFile contains millions of records and sorting is a comparatively slow, disk-intensive, procedure. We will only sort the file as a last resort. Is there any other way to solve the problem?

Another solution that we might adopt is to create 26 variables to hold the county tax totals. Then, in the program, we could use an EVALUATE statement to add the TaxPaid to the appropriate total. For example -

```
EVALUATE CountyCode
   WHEN      1      ADD TaxPaid TO County1TaxTotal
   WHEN      2      ADD TaxPaid TO County2TaxTotal
   WHEN      3      ADD TaxPaid TO County3TaxTotal
         ..... 23 more WHEN branches
END-EVALUATE
```

But this solution is not very satisfactory. We have to have a specific WHEN branch to process each county and we have to declare 26 variables to hold the tax totals. And when we want to print the results we have to have 26 DISPLAY statements such as -

```
DISPLAY "County 1 total is ", County1TaxTotal
DISPLAY "County 2 total is ", County2TaxTotal
DISPLAY "County 3 total is ", County3TaxTotal
        ..... 23 more DISPLAY statements
```

But the suggestion above does contain the germ of a satisfactory solution to the problem. It is interesting to note that the processing of each WHEN branch is exactly the same - the TaxPaid is added to a particular county total variable. We could replace all 26 WHEN branches with one statement if we could generalize it to something like - *ADD the TaxPaid TO the CountyTotal location indicated by the CountyCode.*

There is also something interesting we can note about the 26 variables. They all have exactly the same PICTURE and they all have, more or less, the same name - CountyTaxTotal. The only way we distinguish between one CountyTaxTotal and another is by attaching a number to the name e.g. County1TaxTotal, County2TaxTotal, County3TaxTotal etc.

When we see a group of variables which all have the same name and the same description and are only distinguished from one another by some number attached to the name, we know that we have a problem crying out for a table-based solution.

To top of page

Using a CountyTax table

Introduction

A table may be defined as a contiguous sequence of memory locations which all have the same name, and which are uniquely identified by that name and by their position in the sequence. The position index is called a "subscript", and the individual components of the table are referred to as "elements".

Tables have the following attributes

- a single name is used to identify all the elements
- individual elements can be identified using an �index� or �subscript�
- all elements have the same type or structure

- COBOL arrays/tables, always start at element 1 (not 0) and go on to the maximum size of the table.
- We indicate which element we want by using the element name followed by the index/subscript in brackets (see examples below).

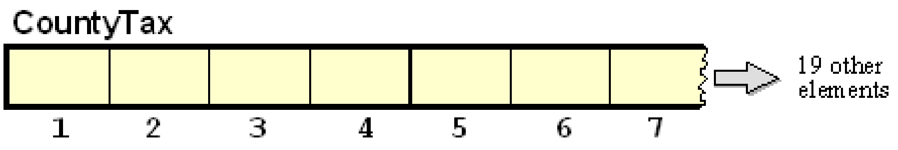
Declaring the CountyTax table

In COBOL, a table is declared by specifying the type (or structure) of a single item (element) of the table and then specifying that the data-item is to be repeated *x* number of times. For instance, the CountyTax table might be defined as -

```
01 CountyTaxTable.
   02 CountyTax       PIC 9(8)V99  OCCURS 26 TIMES.
```

The CountyTax table can be represented diagrammatically as shown below. All the elements of the table have the name CountyTax and we can refer to a specific one by using that name, followed by an integer value in brackets. So CountyTax(2) refers to the second element of the table and CountyTax(23) refers to the twenty third element.

But when we refer to an element we don't have to use an numeric literal. We can use anything that evaluates to a numeric value between 1 and the size of the table; even a simple arithmetic expression.



Animation

If TaxPaid has a value of 74.75 what do you think each of the program statements opposite will do when they execute. Click on the diagram or the icon to see an animated answer.

```
MOVE 10 TO CountyTax(3)
ADD TaxPaid TO CountyTax(CountyCode)
ADD TaxPaid TO CountyTax(CountyCode + 1)
ADD TaxPaid TO CountyTax(CountyCode - 2)
```

The County Tax Report program

The solution to the problem of producing the County Tax Report, is to use a table to hold the tax totals for each county and to use the CountyCode to allow us to access the correct element of the table.

Once we realise that we can use a table to hold the county tax totals and can use the CountyCode as an index into the table, the solution to the problem becomes very simple.

The entire Procedure Division of a program that reads the Tax file, accumulates the tax totals for each county and then displays the tax totals, is given below.

Program Fragment

```
PROCEDURE DIVISION.
Begin.
   OPEN INPUT TaxFile
   READ TaxFile
      AT END SET EndOfTaxFile TO TRUE
   END-READ
   PERFORM UNTIL EndOfTaxFile
      ADD TaxPaid TO CountyTax(CountyCode)
      READ TaxFile
         AT END SET EndOfTaxFile TO TRUE
      END-READ
   END-PERFORM
   PERFORM VARYING Idx FROM 1 BY 1
           UNTIL Idx GREATER THAN 26
      DISPLAY "County ", CountyCode
              " tax total is " CountyTax(Idx)
   END-PERFORM
   CLOSE TaxFile
   STOP RUN.
```

To top of page

Displaying the County Name

Introduction

When the program above displays the accumulated tax totals, instead of displaying a county name, it displays a county code. The problem with this is that, if the user is going to make any sense of the report, he has to remember which codes represent which counties. In a real environment, you couldn't present the user with the information in this form. The county name would have to be displayed.

So how would we go about displaying the county name?

A table-based solution

One solution might be to use an EVALUATE to examine the CountyCode and then display the appropriate message. For example -

```
EVALUATE CountyCode
  WHEN      1       DISPLAY "Carlow tax total is " CountyTax(1)
  WHEN      2       DISPLAY "Cavan tax total is "  CountyTax(2)
         .... 24 more WHEN branches
END-EVALUATE.
```

But this solution is not very satisfactory. It is simply reverting to the 26 WHEN branch solution presented earlier. But this time, we know that there must be a more elegant solution. And there is!

If we declare a CountyName table and fill its elements with the names of the counties, we can replace the EVALUATE solution above, with one simple statement -

```
      DISPLAY CountyName(Idx) " tax total is " CountyTax(Idx).
```

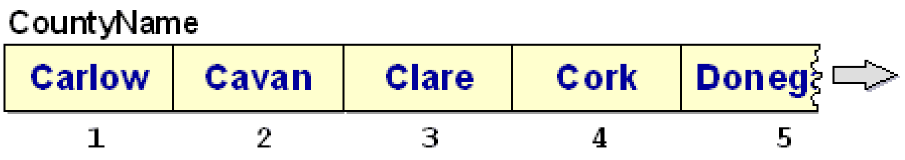
We can incorporate this statement into the County Tax Report program as follows -

Program Fragment

```
PROCEDURE DIVISION.
Begin.
   OPEN INPUT TaxFile
   READ TaxFile
      AT END SET EndOfTaxFile TO TRUE
   END-READ
   PERFORM UNTIL EndOfTaxFile
      ADD TaxPaid TO CountyTax(CountyCode)
      READ TaxFile
         AT END SET EndOfTaxFile TO TRUE
      END-READ
   END-PERFORM
   PERFORM VARYING Idx FROM 1 BY 1
           UNTIL Idx GREATER THAN 26
      DISPLAY CountyName(Idx) " tax total is " CountyTax(Idx)
   END-PERFORM
   CLOSE TaxFile
   STOP RUN.
```

A diagrammatic representation of the CountyName table

In the next tutorial we will examine how we can use the REDEFINES clause to set up a predefined table of values. For the moment, we won't concern ourselves with the exact mechanics of setting up the CountyName table but we can represent it diagrammatically as follows;



The elements of the table may be referenced in the usual way, so the statement DISPLAY CountyName(3) will display the value "Cork" and DISPLAY CountyName(5) will display "Dublin".

To top of page

Using one table to index into another

Introduction

In the County Tax Report program, it proved very convenient that the tax records used a code to represent the county instead of using the actual county name. By using the CountyCode as a direct index into the CountyTax table we were able use the statement *ADD TaxPaid TO CountyTax(CountyCode)* to add the TaxPaid to the appropriate total.

Sometimes, however, things are not arranged so conveniently. Suppose that instead of a CountyCode, the tax record contained the actual CountyName. For instance -

```
01 TaxRec.
   88 EndOfTaxFile  VALUE HIGH-VALUES.
   02 PAYENum    PIC 9(8)
   02 County     PIC X(9).
   02 TaxPaid    PIC 9(7)V99.
```

How could we get the program to work then? How would we know, which element of the CountyTax table to add the TaxPaid to?

An EVALUATE based solution

If the tax record contains a county name instead of a county code then we must devise some way to convert the county name into a numeric value that we can use as a table index. What we need to do, is to convert the first county name into the value 1, the second into the value 2 and so on.

One approach we might try, is the now discredited EVALUATE based solution. For example -

```
EVALUATE County
   WHEN "Carlow" MOVE 1 TO CountyNum
   WHEN "Cavan"  MOVE 2 TO CountyNum
   ..... 24 more WHEN branches
END-EVALUATE
ADD TaxPaid TO CountyTax(CountyNum)
```

But by now we realise that there must be a more elegant solution. The question is - what is it?

Using the subscript from one table as the index into another

To convert the county name to a numeric value, we can use the table of CountyNames that we created in the previous section. We will compare the county name in the record with the contents of each element of the CountyName table and, when they match, we will use the value of the CountyName subscript as an index into the CountyTax table.

To compare the value of County with each element of the CountyNames table we can use a PERFORM..VARYING. For example -

```
PERFORM VARYING CountyNum FROM 1 BY 1
     UNTIL CountyName(CountyNum) = County
END-PERFORM
ADD TaxPaid TO CountyTax(CountyNum)
```

You can see how this PERFORM..VARYING works, in the following animation -

Animation

and you can see where it fits into the County Tax Report program, in the example below.

Program Fragment

```
PROCEDURE DIVISION.
Begin.
   OPEN INPUT TaxFile
   READ TaxFile
      AT END SET EndOfTaxFile TO TRUE
   END-READ
   PERFORM UNTIL EndOfTaxFile

      PERFORM VARYING CountyNum FROM 1 BY 1
         UNTIL CountyName(CountyNum) = County
      END-PERFORM
      ADD TaxPaid TO CountyTax(CountyNum)
      READ TaxFile
         AT END SET EndOfTaxFile TO TRUE
      END-READ
   END-PERFORM
   PERFORM VARYING Idx FROM 1 BY 1
           UNTIL Idx GREATER THAN 26
      DISPLAY CountyName(CountyNum)
              " tax total is " CountyTax(CountyNum)
   END-PERFORM
   CLOSE TaxFile
   STOP RUN.
```

To top of page