M

VA M Programming

Introduction

Lesson 3

Standard Reports

Table of Contents

Table of Contents *(Continued)*

Acknowledgements

Developing a comprehensive training package for the ***M*** computer programming language as it is used in VA was not an easy task. Much gratitude goes to the members of the M Programming Language Training Development Team who spent countless hours developing and reviewing material. Sincere appreciation to:

Debbie Channell
Tuscaloosa OI Field Office, Customer Service

Harris H. Lloyd
Central Arkansas Veterans Health Care System

Bob Lushene
Hines OI Field Office, Technical Service

Guide to Symbols Used

<ENTER>	Press the Enter or Return key
<TAB>	Press the Tab key
<space>	One space
	What follows is from part of the VA Programming Standards and Conventions.
📖	This indicates an OPTIONAL reading reference. Additional information on this topic may be found in the references listed below.

Optional Reading References

Lewkowicz, John. *The Complete M: Introduction and Reference Manual for the M Programming Language.* 1989.

Walters, Richard. *M Programming – A Comprehensive Guide.* 1997.

The optional reading references are available through:

M Technology Association
1738 Elton Road, Suite 205
Silver Spring, MD 20903
301-431-4070

References

The VA M Programming Standards and Conventions shown were taken from "VA DHCP Programming Standards and Conventions" document, prepared March 19, 1991, modified in 1995 and approved January 22, 1996.
http://vaww.vista.med.va.gov/Policies/sacc.htm.

Document Update Note

January 2007 – To meet current standards on personal data privacy and accessibility, this document was updated. It was edited to include document metadata and edited to de-identify any personal data. Only the information in this document is affected. This should have no effect on the M routines.

Objectives

The objective of this lesson is to prepare you to write M routines to produce standard reports using the following commands and techniques:

Data organization,

- Putting fields together into records using the concatenation operator,
- Taking fields apart using the $PIECE function,
- The $TEXT function,
- Aligning columns with the $JUSTIFY function,
- Displaying subtotals and final totals, and
- Using the FOR command to control repetition of statements.

Characteristics of a Standard Report

- Batch mode - usually doesn't require user input during processing
- Headings - report and column
- Detail lines
- Subtotals (possibly) and final totals

Standard reports are usually executed in batch mode, that is, once the user requests the report, no further interaction is required or allowed. On occasion; however, it may be desirable to provide some initial information, like a date range. In such a case, the program would request the required information before proceeding to produce the report.

The report program will begin by printing appropriate headings, describing or identifying the data or information in report rows and columns.

Rows of specific information are written corresponding to the headings.

It may be appropriate to include subtotals and totals as separate lines in the report.

How Data is Organized

Character
The smallest unit of data, a single character or a single keystroke.

Field (variable)
A group of logically related characters or set of keystrokes, e.g., NAME.

Record
A group of logically related fields or a collection of fields, e.g., NAME, SSN, DOB.

File
A group of logically related records or a collection of records, e.g., FirstName1 LastName1, FirstName2 LastName2, FirstName3 LastName3.

Sample Data

NAME	ADDRESS	CITY	STATE	ZIP
First1 Last1	545 Any St.	Pasadena	CA	00001
First2 Last2	545 Every St.	Chicago	IL	00002
First3 Last3	6987 Any Ct.	Boise	ID	00003
First4 Last4	1 Your St.	Stone Mtn.	GA	00004

The smallest unit of data is a **character**, that is, an alphabetic, numeric or punctuation character formed by a single keystroke. Capital letters, control characters, and special characters that require the simultaneous pressing of two or more keys are still considered single characters.

A **field** is a sequence of characters often forming a descriptive and meaningful word. A field is also referred to as a variable.

A **record** is a group of fields, or variables, which are logically related, for example, demographic information about a single person – name, social security number, date of birth, and so forth.

A **file** is a collection of logically related records, for example, the demographic information about a group of individuals.

Shown above is a sample data set. Faux names and faux complete addresses of a group of individuals are listed.

Putting Fields Together into Records

Concatenation Operator (the _ operator)

The purpose of the concatenation operator is to join two or more strings to form a single string.

```
Sample M Code                       Output

SET AA="NIGHT"_"MARE"
WRITE AA                            NIGHTMARE
SET BB="SAN"
SET CC="FRAN"
SET XDD="CISCO"
SET EE=BB_" "_CC_XDD
WRITE EE                            SAN FRANCISCO
```

The concatenation operator is designated by the underscore symbol. It is used to join two or more strings to form one string. The first line of the sample M code concatenates the string `night` to the string `mare` and sets the result equal to the variable AA.

The second line writes the variable AA displaying `nightmare` on the screen.

Next, the sample M code assigns values to the variables BB, CC, and XDD.
The variable EE is set equal to `san` concatenated with a space concatenated with `fran` concatenated with `cisco`.

If EE is now written, then the name of that famous city appears on the screen.

The concatenation operator is most often used to join individual fields together to form records. Each field is separated by a boundary marker (in this case the ^ character).

M Code Sample

```
SET RECORD=NAME_"^"_ADDRESS_"^"_CITY_"^"_STATE_"^"_ZIP
```

The concatenation operation is often used to form records from a group of fields, or variables. In the sample M code above the fields are NAME, ADDRESS, CITY, STATE and ZIP, each of which would have meaningful values assigned.

To form the record, the fields are concatenated together, but with an arbitrary separation character between each one. The separation character in this case is the caret, most commonly referred to as simply the up arrow. The separation character, or delimiter, marks the boundary between fields in a record and should be chosen so that it will not to be confused with actual data.

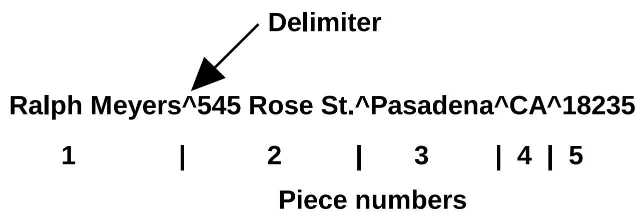
VA Programming Standards and Conventions

The following is **NOT** an official VA Programming Convention, but reflects the practice in VA information systems:

Use the symbol ^ (up-arrow) to separate fields,
(it does not normally appear as part of a data field).

Taking Records Apart with $PIECE

The purpose of the $PIECE function is to separate a string into pieces using boundary markers called delimiters.



There are several forms of the $PIECE function. The most commonly used form is:

$PIECE(<string to be separated>,<delimiter>,<piece number>)

The sample record above has five pieces each delimited by the up arrow. There are several forms of the $PIECE function but the most commonly used form has three arguments – the string to be processed, the delimiter character, and the piece, specified by position number, to be separated.

Use the data in the previous example to create a record. Separate the record into its individual components using the $PIECE function and then display the pieces using the WRITE command.

Set the variable RECORD to the string in the preceding graphic.

Obtain the first piece and set it to the variable NAME.

Continue to use the $PIECE function to obtain the other variables, ADDRESS, CITY, STATE, ZIP and XDATE. Notice that the string contains five pieces.

Use the write command to display the variables one at a time: FIRST NAME, followed by ADDRESS, CITY, STATE, ZIP and finally XDATE. Notice: since the record only contained five pieces, that the variable XDATE was set to null – there is no information in the sixth piece.

Sample M Code

```
SET RECORD="First1 Last1^545 Any St.^YourTown^CA^00005"
SET NAME=$PIECE(RECORD,"^",1)
SET ADDRESS=$PIECE(RECORD,"^",2)
SET CITY=$PIECE(RECORD,"^",3)
SET STATE=$PIECE(RECORD,"^",4)
SET ZIP=$PIECE(RECORD,"^",5)
SET XDATE=$PIECE(RECORD,"^",6)
                                     Output
WRITE NAME                         First1 Last1
WRITE ADDRESS                      545 Any St.
WRITE CITY                         YourTown
WRITE STATE                        CA
WRITE ZIP                          00005
WRITE XDATE                        (a null character is printed)
```

In the example below, use a social security number, SSN. The $PIECE function is now embedded in the WRITE command to extract the third piece separated by a hyphen and display the four digits on the screen. Since a function always returns a value, it is not necessary to set that value to another variable before using it.

Sample M Code

```
SET SSN=”000-62-7777”
WRITE $PIECE(SSN,”-“,3)            7777
```

Putting Data into the M Program

$TEXT Function

The purpose of the $TEXT function is to access a line of a routine. This line can then be used like data. The syntax of $TEXT is:

$TEXT(<line label>) or $TEXT(<line label><^routine name>)

VA Programming Standards and Conventions

Any lines that will be used with $TEXT must begin with two semicolons (;;). Also, it is acceptable to use a line label + offset with the $TEXT (but not with DO or GOTO). The two semicolons instruct M to retain the line in the precompiled code. A line beginning with one semicolon is considered a comment and is stripped from the precompiled code.

The $TEXT function is used to obtain a line of code from a M routine. This line of code can then be treated as any other string of information or data. The $TEXT function lets an M routine read itself or to read lines of any other accessible M routine. Including data or information in a program is frequently a useful technique, especially when only a small amount is involved which will always be used in a fixed manner. Lines of code containing data that will be accessed by the $TEXT function must begin with two semicolons followed by the rest of the line. The line may contain pieces of data separated by your choice of a delimiter character.

Let's assume the following paragraph is in your routine:

Sample M Code

```
DATA  ;
      ;;First1 Last1^545 Any St.^YourTown^CA^00006
      ;;First2 Last1^545 Any St.^YourTown^CA^00006
      ;;First3 Last3^6987 Every Ct.^AnyTown^ID^00006
      ;;First4 Last4^1 My St.^MyCity.^GA^00003
```

This short M program segment begins with the line label DATA followed by four lines of information. Each line begins with two semicolons and, the up arrow separates the pieces of data.

The output from each of the following WRITE commands is shown below each line:

```
WRITE $TEXT(DATA+1)
     ;;First1 Last1^545 Any St.^YourTown^CA^00006
WRITE $TEXT(DATA+2)
     ;;First2 Last1^545 Any St.^YourTown^CA^00006
WRITE $TEXT(DATA+3)
     ;;First3 Last3^6987 Every Ct.^AnyTown^ID^00006
WRITE $TEXT(DATA+4)
     ;;First4 Last4^1 My St.^MyCity.^GA^00003
WRITE $TEXT(DATA+5)
     (a null character is printed; there is no line DATA+5)
```

Use the $TEXT function embedded in a WRITE command to display each of the lines. Notice that the each line is obtained by using the line label plus offset in the $TEXT function.

The first line of data is displayed using DATA+1, the second using DATA+2, the third using DATA+3, the fourth using DATA+4 and the fifth using DATA+5. Note that since there is no fifth line, that the $TEXT function returns null, so nothing is printed.

This procedure would give us each line of the DATA paragraph one line at a time. Theoretically, we could take a line and split out the pieces as we saw in the previous section. The only problem, however, is that when we get piece one of any of those lines, we will also get the two semicolons (;;).

To solve the problem of the two semicolons, we could use the following code:

M Code Sample

```
    SET STEP1=$TEXT(DATA+1)
```

which would give us:

```
          ;;First1 Last1^545 Any St.^YourTown^CA^00006
```

```
    SET STEP2=$PIECE(STEP1,";;",2)
```

which would give us:

```
          First1 Last1^545 Any St.^YourTown^CA^00006
```

Here is one way to handle the two semicolons.

First, obtain the line of data using the $TEXT function and set it equal to a variable, in this example, STEP1. You can see that the two semicolons are at the beginning of the line.

If you consider the two semicolons to be a separator, or piece delimiter, then the next step is to use the $PIECE function to obtain all the characters of the string to the right of the separator, that is, the second piece of the string when the separator is two semicolons.

This is shown in the code assigning a value to the variable STEP2. Now the two semicolons have been removed form the line of data.

Here is another method for dealing with the two semicolons.

We can actually combine the two steps in the preceding example into one statement. In the sample M code below, at the line INPUT+1, the $TEXT function is nested inside a $PIECE function. Remember, when M finds parentheses nested inside other parentheses, it will do what is in the innermost parentheses first.

M Code Sample

```
(segment of complete routine)
INIT ;
     SET RECNR=1
     .
     .
     .

INPUT     ;
     SET RECORD=$PIECE($TEXT(DATA+RECNR),";;",2)
     IF RECORD="" GOTO EXIT
     SET NAME=$PIECE(RECORD,"^",1)
     SET ADDRESS=$PIECE(RECORD,"^",2)
     SET CITY=$PIECE(RECORD,"^",3)
     SET STATE=$PIECE(RECORD,"^",4)
     SET ZIP=$PIECE(RECORD,"^",5)
     .
     .                    (processing continues)
     .
     SET RECNR=RECNR+1
     GOTO INPUT
EXIT ;
     KILL RECORD,NAME,ADDRESS,CITY,STATE,ZIP,RECNR
     QUIT
```

```
            DATA ;
            ;;First1 Last1^545 Any St.^YourTown^CA^00006
            ;;First2 Last1^545 Any St.^YourTown^CA^00006
            ;;First3 Last3^6987 Every Ct.^AnyTown^ID^00006
            ;;First4 Last4^1 My St.^MyCity.^GA^00003
```

The $TEXT function returns the entire string of data including the two semicolons. The $PIECE function then obtains the second piece using the two semicolons as the separator.

The program segments show that each line of data, without the semicolons, would be obtained and then processed in some way. When all the lines of data have been processed then the program would kill all variables and quit.

Producing a Standard Report (no totals)

Report Output

This is the report we would like an M program to produce. As you can see there are appropriate headings and labels for the rows and columns. The numbers in the report are calories burned for the specified interval of exercise.

JOE'S GYM
EXERCISE PLAN--CALORIES BURNED

ACTIVITY	15 MIN	30 MIN	60 MIN
BOXING	199	399	798
WALKING UP STAIRS	150	300	600
RUNNING 5 MPH	150	300	600
ARCHERY	75	150	300
WALKING 4 MPH	105	210	420
SLEEPING	35	69	138
JOGGING	225	450	900
SITTING	26	51	102

Keep active, stay healthy!!!

Report Input

Study the data that will have to be supplied in order to prepare the report. This table shows the calories burned for each minute of a particular activity.

	Calories burned per minute
BOXING	13.3
WALKING UP STAIRS	10
RUNNING 5 MPH	10
ARCHERY	5
WALKING 4 MPH	7
SLEEPING	2.3
JOGGING	1
SITTING	1.7

Pseudocode for the Report Routine

Pseudocode

```
1. Initialize the record number to 1.
2. WRITE the report headings and the column headings.
3. Get an input record.  If the record is null, go to the end of routine,
   step number 8.  If it is not null, get the data for:
      ➢ Activity description
      ➢ Calories burned per minute
4. Calculate:
      ➢ Calories burned in 15 min = calories per min * 15
      ➢ Calories burned in 30 min = calories per min * 30
      ➢ Calories burned in 60 min = calories per min * 60
5. WRITE on one line the activity description, calories per 15 min,
   calories per 30 min, calories per 60 min.
6. Add 1 to the record number.
7. Go back to step 3.
8. WRITE an ending message, KILL any variables and QUIT the
   program.  Place the data at the end.
```

Writing the Report Routine

Consider the steps above that an M program would have to perform in order to prepare this report.

Pseudocode

1. Initialize the record number to 1.

M Code

```
     INIT ;
          SET RECNR=1
```

The M code to perform this step is set the variable RECNR equal to one. RECNR is the record number.

Also, note that the program begins at line label INIT.

Pseudocode

2. WRITE the report headings and the column headings.

M Code

```
HEADING ;
        WRITE !,?24,"JOE'S GYM"
        WRITE !,?13,"EXERCISE PLAN--CALORIES BURNED"
        WRITE !!,"ACTIVITY",?20,"15 MIN",?35,"30 MIN",?50,"60 MIN",!!
```

This segment of M code begins with the line labeled HEADING. Following this line is three lines of code that print or display the headings. You should remember that the exclamation point instructs M to skip to the next line on the display or printout. The question mark followed by a number instructs M to space over to that column number and that the string in quotes is printer or displayed beginning at column specified.

When this program segment is executed the headings will be printed or displayed as shown in the report output.

Pseudocode

3. Get an input record. If the record is null, go to the end of routine, step number 8. If it is not null, get the data for:

- Activity description;
- Calories burned per minute

M Code

```
INPUT ;
     SET RECORD=$PIECE($TEXT(DATA+RECNR),";;",2)
     IF RECORD="" GOTO EXIT
     SET ACTIV=$PIECE(RECORD,"^",1)
     SET CAL=$PIECE(RECORD,"^",2)
```

This segment of M code begins with the line labeled INPUT. The next line obtains the data located at the line referenced by DATA+RECNR using the $TEXT function. The variable RECNR was initialized to one. The $PIECE function is used to remove the two semicolons and the result is assigned to the variable record. The next line tests the contents of the variable record and if it contains no data, processing is finished and the program is instructed to branch to the line of code labeled EXIT. Otherwise, the next two lines of code set piece one and piece two of RECORD into the variables ACTIV and CAL, respectively.

Pseudocode

4. Calculate:

- Calories burned in 15 min = calories/min * 15
- Calories burned in 30 min = calories/min * 30
- Calories burned in 60 min = calories/min * 60

M Code

```
     CALC ;
          SET MIN15=CAL*15
          SET MIN30=CAL*30
          SET MIN60=CAL*60
```

This segment of M code will begin at line label CALC. The following lines of code multiply the variable CAL, which is calories burned per minute, by duration of activity in minutes and sets the result to the variables MIN15, MIN30 and MIN60.

Pseudocode

5. WRITE the activity description, calories per 15 min, calories per 30 min, calories per 60 min

M Code

```
OUTPUT  ;
        WRITE !,ACTIV,?20,MIN15,?35,MIN30,?50,MIN60
```

The segment of M code to perform this task begins at line label OUTPUT. The next line prints or displays the activity and calories burned for 15, 30, and 60 minutes on a single line, spaced to fall under the previously written headings.

Pseudocode

6. Add 1 to the record number.

M Code

```
        SET RECNR=RECNR+1
```

This segment of M code adds one to the current value of the variable record number and then sets the new value back into the same variable.

Pseudocode

7. Go back to step 3.

M Code

```
              GOTO INPUT
```

The M code is a GOTO statement directing M to branch to the line of code labeled INPUT.

Pseudocode

8. WRITE an ending message, KILL any variables and QUIT the program. Place the data at the end.

M Code

```
EXIT ;
     WRITE !!,"Keep active, stay healthy!!!"
     KILL CAL,MIN15,MIN30,MIN60,RECNR,RECORD,ACTIV
     QUIT
DATA ;
     ;;BOXING^13.3
     ;;WALKING UP STAIRS^10
     ;;RUNNING 5 MPH^10
     ;;ARCHERY^5
     ;;WALKING 4 MPH^7
     ;;SLEEPING^2.3
     ;;JOGGING^15.0
     ;;SITTING^1.7
```

This M code segment begins with the line labeled EXIT. The next statement skips two lines and then prints or displays the closing message. The KILL statement destroys variables and their values. The KILL statement clears the symbol table of all the variables in its argument list, avoiding possible conflict with any subsequent programs that may be executed. The QUIT statement terminates execution of the current program.

Finally, the required data is placed at the end of the program following a line labeled DATA. Remember that each data line must begin with two semicolons and that data elements are separated with a delimiter, in this case, the up arrow.

This completes the program. When executed, the program produces the table, illustrated in the report format. To see the completed report, refer to page 13 in this manual.

This is the complete report routine:

```
VECS313 ;ATL/ACE PROGRAMMER-EXERCISE PLAN;2/1/90
        ;;1
        ;VARIABLE LIST
        ;
        ;RECNR.......record number being processed
        ;RECORD......record pulled from text line in DATA
        ;ACTIV.......activity description
        ;CAL.........calories expended per minute
        ;MIN15.......calories expended in 15 min.
        ;MIN30.......calories expended in 30 min.
        ;MIN60.......calories expended in 60 min.
        ;
INIT    ;
        SET RECNR=1
HEADING ;
        WRITE !,?24,"JOE'S GYM"
        WRITE !,?13,"EXERCISE PLAN--CALORIES BURNED"
        WRITE !!,"ACTIVITY",?20,"15 MIN",?35,"30 MIN",?50,"60 MIN",!!
INPUT   ;
        SET RECORD=$PIECE($TEXT(DATA+RECNR),";;",2)
        IF RECORD="" GOTO EXIT
        SET ACTIV=$PIECE(RECORD,"^",1)
        SET CAL=$PIECE(RECORD,"^",2)
CALC    ;
        SET MIN15=CAL*15
        SET MIN30=CAL*30
        SET MIN60=CAL*60
OUTPUT  ;
        WRITE !,ACTIV,?20,MIN15,?35,MIN30,?50,MIN60
        SET RECNR=RECNR+1
        GOTO INPUT
EXIT ;
        WRITE !!,"Keep active, stay healthy!!!"
        KILL CAL,MIN15,MIN30,MIN60,RECNR,RECORD,ACTIV
        QUIT
DATA    ;
        ;;BOXING^13.3
        ;;WALKING UP STAIRS^10
        ;;RUNNING 5 MPH^10
        ;;ARCHERY^5
        ;;WALKING 4 MPH^7
        ;;SLEEPING^2.3
        ;;JOGGING^15.0
        ;;SITTING^1.7
```

Producing a Standard Report with Final Totals

Report Output

Here is another example.

Design an inventory system for the Paper Warehouse, a fictitious office supply store. A program will be designed to produce a report similar to the previous example except a final total price for the warehouse and a total number of items will be included.

The report, as shown below, begins with an appropriate heading and the data to be shown in each column is identified. At the end of the report a total price is shown and finally the total number of items in the inventory is displayed.

The Paper Warehouse
Inventory Report

DEPT	ITEM	QTY	UNIT PRICE	TOTAL PRICE
01	Yellow paper pads	5	2.95	14.75
01	Pencils	15	.25	3.75
01	Erasers	2	1	2
02	Picture frames	1	8	8
02	Picture wire	5	2.95	14.75
02	Picture hangers	20	3.75	75

Final Total 118.25

Total number of items is 6

Below is a list of the data items that will be needed to prepare the report. These items will be included in the M program as text preceded by two semicolons.

Department	Item	Quantity	Unit Price
01	Yellow paper pads	5	2.95
01	Pencils	15	.25
01	Erasers	2	1
02	Picture frames	1	8
02	Picture wire	5	2.95
02	Picture hangers	20	3.75

Pseudocode for the Report Routine

Pseudocode

1. Initialize counters and accumulators:

   - Final total = 0 (accumulator)
   - Number of items = 0 (counter)
   - Record number = 1 (index)

2. WRITE the headings.

3. Get an input record. If it is null, go to step number 8, the end of the routine. If it is not, get data for:

- Department
- Item
- Quantity
- Unit price

4. Calculate the total price for this item:

   Total price = Quantity * unit price

5. Add item total price to the final total accumulator, add 1 to the number of items counter.

6. WRITE the detail line.

7. Add 1 to the record number and go back to step number 3.

8. WRITE the final total and number of items, KILL any variables and QUIT the routine. Place the data lines at the end of the program.

As you move though the development of the pseudocode for this report, write out the pseudocode on a sheet of paper as you see it here. This practice will help you to further understand the benefit of developing and writing the pseudocode. Here's a description of each of the above steps.

Step 1	Initialize counters and accumulators and any other variables the program will require. In this case, the final total and total number of items is initialized to 0, and the record number is set to 1.
Step 2	WRITE the headings.
Step 3	Read an input record and test its contents. If it is empty, that is, null, go to the exit point at step number 8. If it contains data, that is, not null, obtain values for department, item, quantity and unit price.
Step 4	Calculate the total price for this item by multiplying the quantity by the unit price.
Step 5	Add this total price to the final total accumulator and increment the number of items counter by one.
Step 6	WRITE a detail line that will include department, item, quantity, unit price, and total price, all appropriately spaced to fall under the corresponding report column headings.
Step 7	Increment the record number variable by one and branch back to step three.
Step 8	WRITE the final total and total number of items, KILL any variables and QUIT the routine. Following the QUIT statement add the data lines preceded by two semicolons and separate each data element with a suitable delimiter.

Writing the Report Routine

The next page shows the program that produces the inventory report. Here are some clarifying points about the routine.

There are a number of line labels included simply to mark the beginning of functional segments. With the exception of the lines labeled EXIT and INPUT, they are not referenced within the program. You should be able to locate both the line labels and references to them.

At the top of the program is a list of variables and a description of each. They were previously described as we discussed the pseudocode.

Next comes the initialization segment followed by the code that writes the headings. Then records are retrieved from the data list at the bottom of the program. Calculations are performed and the detail lines are written.

The EXIT segment writes the final output lines, KILLS variables and terminates execution. The data lines follow.

The completed routine.

```
VECS316 ;ATL/ACE PROGRAMMER-INVENTORY REPORT ;2/1/90
        ;;1
        ;VARIABLE LIST
        ;RECNR.......record being processed
        ;RECORD......record pulled from text line in DATA
        ;FTOTAL......final total (accumulator)
        ;TOTITEMS....total number of items (counter)
        ;XDEPT.......department number
        ;ITEM........item description
        ;QTY.........quantity
        ;UNITPR......unit price
        ;TOTAL.......total price per item (calculated)
        ;
INIT    ;
        SET RECNR=1,FTOTAL=0,TOTITEMS=0
        ;
HEADING ;
        WRITE !,?30,"The Paper Warehouse"
        WRITE !,?32,"Inventory Report"
        WRITE !!,"DEPT",?10,"ITEM",?30,"QTY",?40,"UNIT PRICE",?55,"TOTAL PRICE",!!
        ;
INPUT   ;
        SET RECORD=$PIECE($TEXT(DATA+RECNR),";;",2)
        IF RECORD="" GOTO EXIT
        SET XDEPT=$PIECE(RECORD,"^",1)
        SET ITEM=$PIECE(RECORD,"^",2)
        SET QTY=$PIECE(RECORD,"^",3)
        SET UNITPR=$PIECE(RECORD,"^",4)
CALC    ;
        SET TOTAL=QTY*UNITPR
        SET FTOTAL=FTOTAL+TOTAL ; add to accumulator
        SET TOTITEMS=TOTITEMS+1 ; add to counter
        ;
OUTPUT  ;
        WRITE !,XDEPT,?10,ITEM,?30,QTY,?40,UNITPR,?55,TOTAL
        SET RECNR=RECNR+1
        GOTO INPUT
EXIT    ;
        WRITE !!,?38,"Final total is $",?55,FTOTAL
        WRITE !!,"Total number of items is ",TOTITEMS
        KILL TOTAL,FTOTAL,TOTITEMS,UNITPR,XDEPT,ITEM,QTY,RECNR,RECORD
        QUIT
DATA    ;
        ;;01^Yellow paper pads^5^2.95
        ;;01^Pencils^15^.25
        ;;01^Erasers^2^1
        ;;02^Picture frames^1^8
        ;;02^Picture wire^5^2.95
        ;;02^Picture hangers^20^3.75
```

Making Numeric Columns Line Up

The problem with the previous report was that the numeric columns were not aligned nicely. We are going to make a minor change, introducing another M function, to produce the output below.

The Paper Warehouse
Inventory Report

DEPT	ITEM	QTY	UNIT PRICE	TOTAL PRICE
01	Yellow paper pads	5	2.95	14.75
01	Pencils	15	.25	3.75
01	Erasers	2	1	2
02	Picture frames	1	8	8
02	Picture wire	5	2.95	14.75
02	Picture hangers	20	3.75	75
			Final Total	118.25

Final Total 118.25

Total number of items is 6.

$JUSTIFY Function

The purpose of the $JUSTIFY function is to align data either at the right of a field of specified width or by aligning the decimal points.

Aligning Data at the Right of a Column

Unless additional steps are taken an M WRITE command will always begin its output at the current column position and continue to the right until all characters are written, i.e., output will begin at the specified ?tab position. To align data at the right of a field, use this form of $JUSTIFY:

$JUSTIFY(<variable to be aligned>,<field width>)

Data will be aligned at the right of the field. Any extra spaces at the beginning of the field will be left blank. If the field width specified is less than the number of characters to be printed, the $JUSTIFY has no effect, i.e., the $JUSTIFY is ignored by M and the WRITE statement behaves as if the function were not present with output beginning at the current column position.

```
M Code Sample                      Output

SET AA=987
WRITE $JUSTIFY(AA,5)               |_||_|987
WRITE $JUSTIFY(AA,4)               |_|987
WRITE $JUSTIFY(AA,3)               987
WRITE $JUSTIFY(AA,2)               987

SET BB=987.5
WRITE $JUSTIFY(BB,7)               |_||_|987.5
WRITE $JUSTIFY(BB,5)               987.5
WRITE $JUSTIFY(BB,3)               987.5
```

The symbol |_| indicates one space

Study the examples above. Notice that the decimal point counts as one column position in the field.

Aligning Data by Decimal Points

To align data by decimal points, use this form of the $JUSTIFY:

$JUSTIFY(<variable to be aligned>,<field width>,<number of decimal places>)

M will first mark off the total field width and reserve the number of decimal places specified (the decimal point will take up a column also which is included in the total field width). Then:

- Any extra spaces to the right of the decimal point will be padded with zeros.
- Any extra spaces to the left of the decimal point will be padded with spaces.
- If the field width specified is smaller than the number of characters to be printed, M ignores the $JUSTIFY function.
- If the number of decimal places specified for the field is less than the number of decimal places in the variable, M will round to the number decimal of places specified (rounding up if the digits lost are five or greater, truncating otherwise).

```
M Code Sample                      Output

SET BB=987.5
WRITE $JUSTIFY(BB,7,2)             |_|987.50
WRITE $JUSTIFY(BB,6,1)             |_|987.5
WRITE $JUSTIFY(BB,6,2)             987.50

SET CC=987.175
WRITE $JUSTIFY(CC,7,2)             |_|987.18
WRITE $JUSTIFY(CC,6,1)             |_|987.2

SET XDD=.15
WRITE $JUSTIFY(XDD,6,1)            |_||_||_|0.2
```

The symbol |_| indicates one space

The M code samples above show the three-parameter form of the $JUSTIFY function. Study the examples and notice the rounding where that is required by the number of decimal places parameter.

The next page shows the modified program using the $JUSTIFY function to align the written data. Lines of code that have been changed are outlined in a box. Examine carefully the parameters that are used with the function. Both the two parameter and three parameter versions are used. The same variable names are used as function parameters as was used in the original program.

You can compare this modified program to the original program on page 24.

The new complete routine.

```
VECS320   ;ATL/ACE PROGRAMMER-INVENTORY REPORT ;2/1/90
     ;;1
     ;VARIABLE LIST
     ;
     ;RECNR.......record being processed
     ;RECORD......record pulled from text line in DATA
     ;FTOTAL......final total
     ;TOTITEMS....total number of items
     ;XDEPT........department number
     ;ITEM........item description
     ;QTY.........quantity
     ;UNITPR......unit price
     ;TOTAL......total price per item (calculated)
INIT ;
     SET RECNR=1,FTOTAL=0,TOTITEMS=0
HEADING   ;
     WRITE !,?30,"The Paper Warehouse"
     WRITE !,?32,"Inventory Report"
     WRITE !!,"DEPT",?10,"ITEM",?30,"QTY",?40,"UNIT PRICE",?55,"TOTAL PRICE",!!
INPUT     ;
     SET RECORD=$PIECE($TEXT(DATA+RECNR),";;",2)
     IF RECORD="" GOTO EXIT
     SET XDEPT=$PIECE(RECORD,"^",1)
     SET ITEM=$PIECE(RECORD,"^",2)
     SET QTY=$PIECE(RECORD,"^",3)
     SET UNITPR=$PIECE(RECORD,"^",4)
CALC ;
     SET TOTAL=QTY*UNITPR
     SET FTOTAL=FTOTAL+TOTAL
     SET TOTITEMS=TOTITEMS+1
OUTPUT    ;
     WRITE !,XDEPT,?10,ITEM,?30,$JUSTIFY(QTY,3),?40
     WRITE $JUSTIFY(UNITPR,5,2),?55,$JUSTIFY(TOTAL,6,2)
     SET RECNR=RECNR+1
     GOTO INPUT
EXIT ;
     WRITE !,?38,"Final total is $",?54,$JUSTIFY(FTOTAL,7,2)
     WRITE !!,"Total number of items is ",TOTITEMS
     KILL TOTAL,FTOTAL,TOTITEMS,UNITPR,XDEPT,ITEM,QTY,RECNR,RECORD
     QUIT
DATA ;
     ;;01^Yellow paper pads^5^2.95
     ;;01^Pencils^15^.25
     ;;01^Erasers^2^1
     ;;02^Picture frames^1^8
     ;;02^Picture wire^5^2.95
     ;;02^Picture hangers^20^3.75
```

Inserting Commas and Signs with $FNUMBER

In the previous example, none of the printed numeric values were greater than 999. If we were to change the quantity, for example, then some of the total prices become greater than 999. In that case, it is customary to print the values with commas as shown below.

The Paper Warehouse
Inventory Report

DEPT	ITEM	QTY	UNIT PRICE	TOTAL PRICE
01	Yellow paper pads	500	2.95	1,475.00
01	Pencils	1,520	.25	380.00
01	Erasers	2,989	1.00	2,989.00
02	Picture frames	1,000	8.00	8,000.00
02	Picture wire	50	2.95	147.50
02	Picture hangers	20,000	3.75	75,000.00

Final Total is $ 87,991.50

Total number of items is 6

$FNUMBER, which was new with the 1990 ANSI M Standard, formats numeric strings based on specified formatting codes. (Note: Some of these operations will produce string values rather than numeric values). Code letters may be in either upper or lower case. The syntax and codes for the $FNUMBER function are:

$FNUMBER(<number>,<code(s)>,<number of decimal places to display(optional)>)

Codes	Action
T	Place the sign (if displayed) at the end of the number
+	Force a plus sign display in front of positive numbers
-	Suppress the display of the minus sign on negative numbers
P	Put parentheses around negative numbers (can be combined only with “,”)
,	Place commas between every three numbers to the left of the decimal point

Some examples using the $FNUMBER function follow.

```
M Code Sample                  Output

S X=-1000
WRITE $FN(X,",P")                  (1,000)
WRITE $FN(X,"T")                   1000-

SET Y=$FN(X,"-")
WRITE Y                            1000

SET B=1000
WRITE $FN(B,"+")                   +1000
WRITE $FN(B,",+")                  +1,000
WRITE $FN(B,",+",2)                +1,000.00

SET C=1987.175
WRITE $FN(C,",+",2)                +1,987.18
WRITE $J($FN(C,"P,",2),15)     |_||_||_||_||_||_||_|1,987.18
```

The symbol |_| indicates one space

The examples above use the abbreviation $FN for the function. Compare the output of each WRITE statement with the formatting code or combination of codes in the corresponding $FNUMBER function. Note that the last example nests the $FNUMBER inside a $JUSTIFY function, abbreviated to $J.

The revised routine using the $FNUMBER function is shon on the next page. Notice that the parts of the program that were changed are enclosed in boxes.

```
VECS321   ;ATL/ACE PROGRAMMER-INVENTORY REPORT ;2/1/90
     ;;1
     ;VARIABLE LIST
     ;
     ;RECNR.......record being processed
     ;RECORD......record pulled from text line in DATA
     ;FTOTAL......final total
     ;TOTITEMS....total number of items
     ;XDEPT........department number
     ;ITEM.......item description
     ;QTY.........quantity
     ;UNITPR......unit price
     ;TOTAL.......total price per item (calculated)
INIT ;
     SET RECNR=1,FTOTAL=0,TOTITEMS=0
HEADING   ;
     WRITE !,?30,"The Paper Warehouse"
     WRITE !,?32,"Inventory Report"
     WRITE !!,"DEPT",?10,"ITEM",?30,"QTY",?40,"UNIT PRICE",?55,"TOTAL PRICE",!!
INPUT     ;
     SET RECORD=$PIECE($TEXT(DATA+RECNR),";;",2)
     IF RECORD="" GOTO EXIT
     SET XDEPT=$PIECE(RECORD,"^",1)
     SET ITEM=$PIECE(RECORD,"^",2)
     SET QTY=$PIECE(RECORD,"^",3)
     SET UNITPR=$PIECE(RECORD,"^",4)
CALC ;
     SET TOTAL=QTY*UNITPR
     SET FTOTAL=FTOTAL+TOTAL
     SET TOTITEMS=TOTITEMS+1
OUTPUT    ;
     WRITE !,XDEPT,?10,ITEM,?30,$J($FN(QTY,","),6),?40
     WRITE $J($FN(UNITPR,"P,",2),9),?55,$J($FN(TOTAL,"P,",2),11)
     SET RECNR=RECNR+1
     GOTO INPUT
EXIT ;
     WRITE !,?38,"Final total is $",?55,$J($FN(FTOTAL,"P,",2),11)
     WRITE !!,"Total number of items is ",TOTITEMS
     KILL TOTAL,FTOTAL,TOTITEMS,UNITPR,XDEPT,ITEM,QTY,RECNR,RECORD,X
     QUIT
DATA ;
     ;;01^Yellow paper pads^500^2.95
     ;;01^Pencils^1520^.25
     ;;01^Erasers^2989^1
     ;;02^Picture frames^1000^8
     ;;02^Picture wire^50^2.95
     ;;02^Picture hangers^20000^3.75
```

Adding Subtotals by Department

Now we are going to advance even further by adding subtotals by department. We now wish to create the report shown below. It is different from the previous report in that subtotals are now included for the two departments, designated as department one and department two.

The Paper Warehouse
Inventory Report

DEPT	ITEM	QTY	UNIT PRICE	TOTAL PRICE
01	Yellow paper pads	5	2.95	14.75
01	Pencils	15	0.25	3.75
01	Erasers	2	1.00	2.00

Subtotal for this department is $20.50

DEPT	ITEM	QTY	UNIT PRICE	TOTAL PRICE
02	Picture frames	1	8	8
02	Picture wire	5	2.95	14.75
02	Picture hangers	20	3.75	75

Subtotal for this department is $97.75

Final total is $118.25

Total number of items is 6

The revised routine with additional M code to display department subtotals is shown on thje next page. Notice that the parts of the program that were changed are enclosed in boxes. In particular, note that some new variables have been introduced to hold information for the calculating of the subtotals.

```
VECS322   ;ATL/ACE PROGRAMMER-INVENTORY REPORT ;2/1/90
     ;;1
     ;VARIABLE LIST
     ;RECNR.......record being processed
     ;RECORD......record pulled from text line in DATA
     ;FTOTAL......final total
     ;TOTITEMS....total number of items
     ;CURRDEPT....current record's department number
     ;ITEM........item description
     ;QTY.........quantity
     ;UNITPR......unit price
     ;TOTAL.......total price per item (calculated)
     ;PREVDEPT....a temporary variable that holds the
     ; previous record's department number
     ;SUBTOT......subtotal for current department
INIT ;
     SET RECNR=1,FTOTAL=0,TOTITEMS=0,SUBTOT=0
HEADING   ;
     WRITE !,?30,"The Paper Warehouse"
     WRITE !,?32,"Inventory Report"
     WRITE !!,"DEPT",?10,"ITEM",?30,"QTY",?40,"UNIT PRICE",?55,"TOTAL PRICE",!!
INPUT     ;
     SET RECORD=$PIECE($TEXT(DATA+RECNR),";;",2)
     IF RECORD="" GOTO EXIT
     SET CURRDEPT=$PIECE(RECORD,"^",1)
     SET ITEM=$PIECE(RECORD,"^",2)
     SET QTY=$PIECE(RECORD,"^",3)
     SET UNITPR=$PIECE(RECORD,"^",4)
     IF RECNR=1 SET PREVDEPT=CURRDEPT
     IF PREVDEPT'=CURRDEPT DO SUBT
     SET PREVDEPT=CURRDEPT
CALC ;
     SET TOTAL=QTY*UNITPR
     SET SUBTOT=SUBTOT+TOTAL
     SET FTOTAL=FTOTAL+TOTAL
     SET TOTITEMS=TOTITEMS+1
OUTPUT    ;
     WRITE !,CURRDEPT,?10,ITEM,?30,$J($FN(QTY,","),6),?40
     WRITE $J($FN(UNITPR,"P,",2),9),?55,$J($FN(TOTAL,"P,",2),11)
     SET RECNR=RECNR+1
     GOTO INPUT
SUBT ;
     WRITE !!,?21,"Subtotal for this department is $",?55,$J($FN(SUBTOT,"P,",2),11),!!
     SET SUBTOT=0
     QUIT
EXIT ;
     DO SUBT
     WRITE !,?38,"Final total is $",?55,$J($FN(FTOTAL,"P,",2),11)
     WRITE !!,"Total number of items is ",TOTITEMS
     KILL TOTAL,FTOTAL,TOTITEMS,UNITPR,CURRDEPT,ITEM,QTY,RECNR,RECORD
     KILL SUBTOT,PREVDEPT
     QUIT
DATA ;
     ;;01^Yellow paper pads^5^2.95
     ;;01^Pencils^15^.25
     ;;01^Erasers^2^1
     ;;02^Picture frames^1^8
     ;;02^Picture wire^5^2.95
     ;;02^Picture hangers^20^3.75
```

Controlled Repetition with the FOR Command

The logic of a routine often requires that parts of a program be repeated. This repetition of statements is called looping. Sometimes, the repetition continues until stopped by the occurrence or detection of some condition. One way M controls looping is by use of the FOR command.

Initial, Incremental, Ending Values Given

The most commonly used form of the FOR command has a parameter called a control variable to set limits on the repetition. An initial value, an increment value, and an ending value are specified. The syntax for this form of the FOR command is:

FOR<space><control variable = initial value>:<increment>:<ending value><space><body of loop>

The control variable is incremented or decremented by the quantity specified for each execution of the M code following the FOR command. Whenever the control variable reaches or exceeds the ending value execution of the loop stops.

With all forms of the FOR command, all the M code to be repeated must be placed on the same line as the FOR command. Compare the use of the IF command to control looping to that of the FOR command as shown in the following examples.

M Sample Code ***Output***

Using the **IF** command:

```
       SET COUNT=1
NEXT   ;
       IF COUNT>5 GOTO EXIT
       WRITE !,COUNT                    1
       SET COUNT=COUNT+1                2
       GOTO NEXT                        3
EXIT   ;                                4
       QUIT                             5
```

Using the **FOR** command:

```
       FOR COUNT=1:1:5 WRITE !,COUNT    1
       QUIT                             2
                                        3
                                        4
                                        5
```

Notice that the IF command tests the value of COUNT and as soon as it exceeds five, the program branches to a line labeled EXIT and terminates.

For each value of COUNT, it is displayed with a WRITE command and then incremented. The program then branches back to NEXT where the process repeats.

All of this can be accomplished with one FOR statement. In this case, the variable COUNT is also the control variable. It is initialized to one, incremented by one and terminates the loop as soon as it reaches five. Inside the FOR, the WRITE command prints the value of COUNT for each of its values within the range of one to five.

The FOR statement is very powerful tool for controlling looping because it is very compact compared to other methods of control.

Initial and Increment Values Only

In the examples below using the for command, the ending value is not specified.

In the first case the loop never ends and the value of COUNT will increment indefinitely. If the output were directed to a printer then a lot of paper would be used.

In the second case, a QUIT command, conditional on the value of COUNT equaling five, is placed inside the FOR statement. This is an example in which a loop is terminated on the occurrence of some event, that is, COUNT reaches the value five.

```
M Code Sample                                      Output

FOR COUNT=1:1 WRITE !,COUNT                          1
                                                     2
                                                  ...endless loop!

FOR COUNT=1:1 WRITE !,COUNT QUIT:COUNT=5             1
                                                     2
                                                     3
                                                     4
                                                     5
```

List of Control Variable Values

This is another form of the FOR command. The control variable is a list of string values. The variable RAINBOW is assigned the values in the list from left to right. The M code inside the FOR command is executed for each value assigned to RAINBOW producing the output shown. The program ends when there are no more values for the variable RAINBOW.

M Code Sample

```
FOR RAINBOW="RED","ORANGE","YELLOW","GREEN","BLUE","INDIGO","VIOLET" WRITE !,RAINBOW
```

```
Output:

     RED
     ORANGE
     YELLOW
     GREEN
     BLUE
     INDIGO
     VIOLET
```

Combining Argument Types of the FOR Command

Here are two examples in which argument types are combining in a single FOR command.

In the example below, the standard argument structure is combined with a list of values for the control variable. The output produced by this combination is shown at the right.

M Code Sample ***Output***

```
FOR FIELDS=1:1:5,8,10 WRITE !,FIELDS         1
                                             2
                                             3
                                             4
                                             5
                                             8
                                             10
```

The next example below combines the standard form followed by a list followed by another standard form. The output is again shown at the right.

M Code Sample ***Output***

```
FOR FIELDS=1:1:5,8,10,20:-1:15 WRITE !,FIELDS
                                             1
                                             2
                                             3
                                             4
                                             5
                                             8
                                             10
                                             20
                                             19
                                             18
                                             17
                                             16
                                             15
```

Nested FOR Loop

When two FOR commands are included in one statement, this is referred to as a nested FOR loop. The first one is called the outer loop; the second one is called the inner loop.

The outer loop is started, then the inner loop is started. The inner loop is completed before M returns to the outer loop. An everyday example of this concept is the time as displayed on a digital clock. The minutes progress from 00 to 59 before the hour increments, then the minutes go from 00 to 59 again before another hour changes.

The example below produces a portion of the multiplication table. The outer loop increments from one to five while the inner loop increments form one to three for each cycle of the outer loop. The outer loop increments to one and then the outer loop increments from one to three. When the inner loop processes three, the program returns to the outer loop and increments to two and then returns to the inner loop beginning with one and ending at three. Then, it returns back to the outer loop for three. This continues until the outer loop equals five and the inner loop equals three. The control variables are printed and their product is printed to produce the output shown on the next page.

M Code Sample

```
VECS328   ;
     WRITE !,?20,"MULTIPLICATION TABLE",!
     FOR X=1:1:80 WRITE "*"
     FOR OUTER=1:1:5 FOR INNER=1:1:3 WRITE !,?5,OUTER," X ",INNER," = ",OUTER*INNER
     KILL INNER,OUTER
     QUIT
```

Output:

```
                   MULTIPLICATION TABLE
************************************************************
     1 X 1 = 1
     1 X 2 = 2
     1 X 3 = 3
     2 X 1 = 2
     2 X 2 = 4
     2 X 3 = 6
     3 X 1 = 3
     3 X 2 = 6
     3 X 3 = 9
     4 X 1 = 4
     4 X 2 = 8
     4 X 3 = 12
     5 X 1 = 5
     5 X 2 = 10
     5 X 3 = 15
```

Sample Routine Using a FOR Loop

Output:

```
                        SALARY TABLE
************************************************************************

SALARY      4%          4.5%        5%          5.5%        6%
            INCR        INCR        INCR        INCR        INCR

12000       12480       12540       12600       12660       12720
12500       13000       13062.5     13125       13187.5     13250
13000       13520       13585       13650       13715       13780
13500       14040       14107.5     14175       14242.5     14310
```

Sample M code for the output from the previous page.

```
VECS330   ;ATL/ACE PROGRAMMER-SALARY CALCS ;1/1/90
     ;;1
     ;VARIABLE LIST
     ;
     ;RECNR......record number of record being processed
     ;INCR......control variable of loop; percent of increase
     ;SALARY.....base salary
     ;TAB........current tab position
     ;X..........control variable
INIT ;
     SET RECNR=1
HEADING   ;
     WRITE !,?20,"SALARY TABLE"
     FOR X=1:1:60 WRITE "*"
     WRITE !!,"SALARY",?15,"4%",?25,"4.5%",?35,"5%"
     WRITE ?45,"5.5%",?55,"6%"
     WRITE !,?15,"INCR",?25,"INCR",?35,"INCR"
     WRITE ?45,"INCR",?55,"INCR",!!
INPUT ;
     SET SALARY=$PIECE($TEXT(DATA+RECNR),";;",2)
CHECK     ;
     IF SALARY="" GOTO EXIT
CALC ;
     WRITE !,SALARY
     SET TAB=0
     FOR INCR=.04:.005:.06 DO INCRSAL
     SET RECNR=RECNR+1
     GOTO INPUT
INCRSAL   ;
     SET TAB=TAB+10
     WRITE ?TAB,SALARY+(SALARY*INCR)
     QUIT
EXIT ;
     KILL SALARY,RECNR,TAB,INCR,X
     QUIT
DATA ;
     ;;12000
     ;;12500
     ;;13000
     ;;13500
```

Assignment

In general:

- Follow the VA Programming Conventions as illustrated in this lesson. Also, remember that the second line of the routine contains the version number; use 1 for this lesson.
- Use only the techniques, commands, and features described in this lesson or the previous lessons.
- Save the routine(s) under the naming convention specified by your facility systems manager.

Assignment Specifications

Using the Sample Routine immediately above as your guide, produce a report for Harry's Video Emporium that uses the following input:

Dept.	Movie Title	Rating	Nr/sold	Price/tape
1	Accidental Tourist	4	12	9.95
1	Alien Nation	3	2	29.95
1	The 'Burbs	3	15	19.95
1	Rain Man	5	75	39.95
2	Slaves of New York	2	1	8.95
2	Fly II	1	15	16.95
2	Star Trek V	3	50	29.95
3	Batman	4	33	29.95
3	Dead Poets Society	5	80	49.95

The output should be a report that prints the following for each movie title:
Dept, Movie Title, Rating-converted to *s (use a FOR loop to display
one * for each movie rating point; see tip below), the nr. sold, the Price/tape, and the total amount per movie (calculate it by taking the number of tapes sold times price per tape).

The output should also have subtotals on the total amount per movie for each department. A final total of all individual movie total amounts and a total count of the number of movie titles on the report (there are nine movies on the report) should be printed at the end.

Hint about printing ratings in *s: remember that the ending value of a FOR loop can be a variable so the following should make sense:

```
            FOR X=1:1:RATING WRITE "*"
```

If RATING had a value of 4, the output would be ****.

Make sure all numeric columns line up as in the Sample Routine and have commas if needed.