



M

VA M Programming Introduction

Lesson 2

Selection Techniques

VA M PROGRAMMING

Introduction

Table of Contents

Guide to Symbols Used	4
Optional Reading References	4
References	4
Document Update Note	4
Objectives	5
Conditional transfer of control – types of comparisons	5
Single alternative	5
Double alternative	6
\$TEST special variable	8
Argumentless IF command	8
Extending the true branch with subroutines	9
ELSE command	11
Relational operators	13
Numeric relations	13
Numeric interpretation of string data	13
String relations	15
Equals operator	15
Contains operator	16
Follows operator	18
Boolean expressions	20
Boolean (logical) operators	20
AND operator	20
OR operator	22
NOT operator	23
Combining logical operators	25
Single variable conditions	25
Time-out on READ	27
Postconditionals	28
Postconditionals on commands	29
Postconditionals on arguments	30
Menus and subroutines	30
\$SELECT function	32
Using sentinel values	34

VA M PROGRAMMING

Introduction

Table of Contents *(Continued)*

Sample routines	35
Assignment (optional)	41
Assignment specifications	41
Optional Exercise	42

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

VA M PROGRAMMING

Introduction

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.

VA M PROGRAMMING

Introduction

Objectives

The objective of this lesson is to prepare you to write your first interactive M routine using the following programming techniques and M commands:

- Commands that impose conditional transfer of control: IF and ELSE
- \$TEST special variable
- DO command and subroutines
- Relational operators: numeric (> and <) and string (=, [,])
- Boolean expressions using AND, OR, NOT
- Single variable conditions
- Postconditionals on arguments and commands
- Menus and subroutines
- \$SELECT function
- Sentinel values for stopping a routine

Conditional Transfer of Control –Types of Comparisons

Single Alternative

With a single alternative, one or more statements are executed if the condition is true. If the condition is false, these statements are ignored and the normal program flow continues.

Pseudocode for Single Alternative Comparison

IF condition being tested is TRUE DO all statements that are listed on the same line as the IF statement. Processing continues with the next line.

M Code Sample

	Output
SET AGE=66 IF AGE>65 WRITE !,"Senior Citizen"	Senior Citizen
SET AGE=50 IF AGE>65 WRITE !,"Senior Citizen"	(nothing displayed)

Look at these two examples. In the first, we see the variable age being SET to 66, then we see a line of code with two statements.

VA M PROGRAMMING

Introduction

The first is if the variable age is greater than 65. In this case, the IF command is followed by a single space. The argument to the IF command is the test of the variable AGE being greater than 65. This statement is followed by a single space before the next statement.

The second statement is WRITE a line feed followed by the literal "senior citizen". We see that the output column indicates that the words senior citizen will be displayed. This is because the value of AGE is 66, which is greater than 65.

In the second example, we SET the variable age equal to 50. Then we execute the same line of code as the first example. This time however, nothing will be displayed. This is because once the IF command's argument is found to be FALSE, no other commands on that line of code will be executed. Since the variable AGE is now 50, which is not greater than 65, execution of the routine will continue with the next line of code.

This is important to remember because the M interpreter will not evaluate the rest of the line of code, which may hide an error in logic or syntax that will only appear under certain occasions.



Double Alternative

With a double alternative, there is a choice between two sets of statements. One is executed if the condition is TRUE and the other if the condition is FALSE.

Pseudocode for Double Alternative

IF condition being tested is TRUE DO all statements that are listed on the same line as the IF command.

ELSE the condition being tested is FALSE DO all statements that are listed on the same line as the ELSE.

VA M PROGRAMMING

Introduction

M Code Sample

```
SET BALANCE=-10
IF BALANCE<0 WRITE !,"Account is overdrawn"
ELSE WRITE !,"Account is fine"
```

Output: Account is overdrawn

```
SET BALANCE=25
IF BALANCE<0 WRITE !,"Account is overdrawn"
ELSE WRITE !,"Account is fine"
```

Output: Account is fine

Note: ELSE is an argumentless command so it always has two spaces after it.

In the first M code sample we SET the variable BALANCE equal to -10. We then see a line of code similar to the single alternative. This line states, IF the variable BALANCE is less than 0, WRITE a line feed and the literal, account is overdrawn. Now comes the second alternative. ELSE WRITE a line feed and the literal, account is fine. Note that the ELSE command has no argument. It is implied that its argument is all other possibilities not included in the IF command's argument. By rule, the ELSE command never has arguments, so it always is followed by two spaces.

In the first sample, the output is the statement account is overdrawn. This is because the argument of the IF command evaluates to TRUE since negative ten is less than 0. Code execution will continue on the line following the ELSE command.

As we look at the second sample, we see the variable BALANCE being SET to 25. The output of this example shows how the control of execution is transferred to the ELSE line. We see that 25 is not less than 0, so we do not display the overdrawn message but instead we display the statement account is fine.



\$TEST Special Variable

In M, the result of a comparison is expressed as a 1 if TRUE or a 0 if FALSE. When a comparison on an IF command is evaluated, a special variable called \$TEST (\$T) is set to reflect the result of the comparison.

- IF condition is true, \$T is set to 1.
- IF condition is false, \$T is set to 0.

If \$T is 1, the rest of the IF command is executed. If \$T is 0, the rest of the IF command is skipped.

ELSE also tests the value of \$T: if \$T is 0, the ELSE is executed. If \$T is 1, the ELSE is skipped.

As we saw in the two previous sets of examples, when the variable AGE was greater than 65 we displayed the words `senior citizen`. This was because the \$TEST variable was 1. When \$TEST was 0, because the variable AGE was not greater than 65, we displayed nothing.

In the second set of examples we saw that \$TEST was set to one when the variable BALANCE was less than 0. There, we displayed an overdrawn message. Then, the interpreter checked the value of \$TEST for the ELSE statement, and continued past the ELSE command. In the second sample the variable BALANCE was greater than 0 so the IF command set \$TEST to 0. This caused the overdrawn message not to be displayed, but rather the ELSE command to pass allowing the account is fine message.

\$TEST is set on all IF commands. You can control the value of \$TEST by entering an IF command with a known argument. If you enter IF 1, that command will force \$TEST to be set to TRUE. If you enter IF 0, that command will force \$TEST to be set to FALSE.

Argumentless IF Command

If you want to continue the true branch of an IF statement because you cannot fit all the statements on the same line with the IF, one way is to use the argumentless IF.

There is no comparison after the IF; instead, there are two spaces. This argumentless IF is done if the value of \$T is 1. If the value of \$T is 0, the argumentless IF is skipped.

VA M PROGRAMMING

Introduction

M Code Sample

```
IF AGE>65 WRITE !,"Senior Citizen" SET XDISC=10
IF SET CODE=1
```

Note the two spaces after the argumentless IF.

As with the ELSE statement, there are two spaces after the second IF command. This tells the interpreter to evaluate the current state of \$TEST and if it is true, to execute the commands on the line with the argument less IF.

In the example shown above, we see that the variable CODE will be set to 1 if the age is greater than 65. We also see that if the variable AGE is less than or equal to 65, the variables CODE and XDISC will not have their values changed by the shown SET commands.



Extending the True Branch with Subroutines

Argumentless IFs are best used when you only have a few extra operations to continue to the next line. A better way to handle multiple statements for the true branch is to create what we call a **subroutine** with those statements and use the DO command to execute the subroutine.

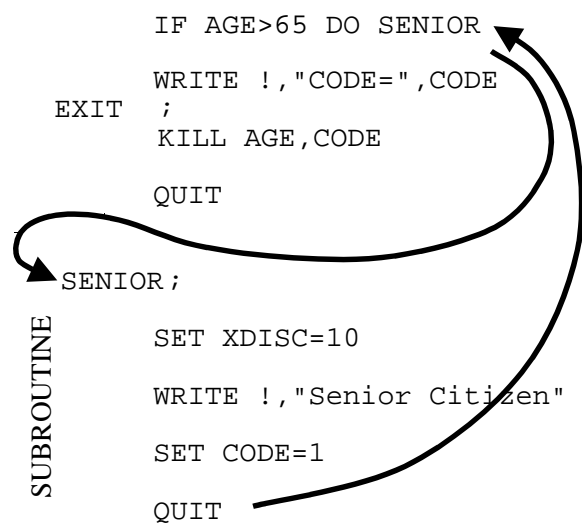
A subroutine is a routine within a routine that begins with a line label and ends with a QUIT command. When we use the DO command with the line label, M goes to that line label and executes all the code it finds until it encounters the first QUIT. When M encounters the QUIT, it returns to the place where the DO "called" the subroutine originally. The routine continues execution from that point.

VA M PROGRAMMING

Introduction

M Code Sample

```
BEGIN ;  
  
    SET AGE=66, CODE=0  
  
    IF AGE>65 DO SENIOR  
    WRITE !, "CODE=", CODE  
EXIT ;  
    KILL AGE, CODE  
  
    QUIT  
  
    SENIOR ;  
SUBROUTINE  
    SET XDISC=10  
  
    WRITE !, "Senior Citizen"  
  
    SET CODE=1  
  
    QUIT
```



```
Output:  Senor Citizen  
         Code=1  
Variable XDISC is set to 10.00  
Variable CODE is set to 1
```

In the sample above, IF AGE is greater than 65 we DO the subroutine labeled SENIOR. In this subroutine we execute the commands to SET the variable XDISC equal to 10, WRITE a line feed and the literal string senior citizen, SET the variable CODE equal to 1, and QUIT. After this QUIT the interpreter would return to execute any other arguments of the DO command or the next command. In this sample, the literal string CODE= with the value of the variable CODE, would be displayed on a line following the words senior citizen.

One thing to consider important is that the QUIT command to end, what we refer to as the DO call, is the first logical quit. Later you will be introduced to what is called nesting, which is the practice of using a DO command inside of a subroutine that was called from another DO command. For now, just focus on the simple single call to a subroutine.

VA M PROGRAMMING

Introduction

ELSE Command

If you want to continue the false branch because you cannot fit all the false branch statements on the line with the ELSE, you can use successive ELSE commands.

Each of the ELSE statements is done if the value of \$TEST is 0.

M Code Sample

```
IF AGE>65 WRITE !,"SENIOR CITIZEN" SET XDISC=10
IF SET CODE=1
ELSE WRITE !,"REGULAR PURCHASE" SET XDISC=0
ELSE SET CODE=2

** or **

IF AGE>65 DO SENIOR
ELSE DO REGULAR
.
.   other processing
GOTO EXIT
SENIOR
;
SET XDISC=10
WRITE !,"SENIOR CITIZEN"
SET CODE=1
QUIT
REGULAR
;
SET XDISC=0
WRITE !,"REGULAR PURCHASE"
SET CODE=2
QUIT
EXIT
;
KILL CODE,XDISC,AGE
QUIT
```

In the sample code above, we see the long way of doing things first. We have the IF command with the argument checking to see IF AGE is greater than 65 followed by an argument less IF command. We follow this with two ELSE commands. This type of coding is hard to read and understand. The code following ****or**** is a much easier way to do the same thing.

This is a more preferred method. In the code following ****or****, we see the line of code executing IF AGE is greater than 65, DO the subroutine, SENIOR. ELSE DO the subroutine, REGULAR. Any other code would be executed until we got to the command to go to the EXIT tag.

VA M PROGRAMMING

Introduction

In either technique, you must be careful that you do not inadvertently change the value of \$TEST before you execute the ELSE. Shown below is a variation of the second technique that will provide erroneous results if AGE is greater than 65 because the IF command in the subroutine SENIOR resets \$T to 0 when AGE is not equal to 66. Because \$T is reset, both the IF and ELSE will be executed!

M Code Sample

```
      SET AGE=76
      IF AGE>65 DO SENIOR
      ELSE  DO REGULAR
      .
      .      other processing
      GOTO EXIT
SENIOR ;
      IF AGE=66 WRITE !,"PERSON IS 66 YEARS OLD"
      SET XDISC=10
      WRITE !,"SENIOR CITIZEN"
      SET CODE=1
      QUIT
REGULAR ;
      SET XDISC=0
      WRITE !,"REGULAR PURCHASE"
      SET CODE=2
      QUIT
EXIT ;
      KILL CODE,AGE,XDISC
      QUIT
```

Relational operators

Numeric Relations

- $A > B$ If A is greater than B, the truth value is 1.
- $A < B$ If A is less than B, the truth value is 0.

M Code Sample

	Output
SET AGE=66 IF AGE>65 WRITE !,"Senior Citizen"	Senior Citizen
SET AGE=50 IF AGE>65 WRITE !,"Senior Citizen"	(Nothing displayed)

M Code Sample

	Output
SET AGE=3 IF AGE<5 WRITE !,"Nursery School"	Nursery School
SET AGE=50 IF AGE<5 WRITE !,"Nursery School"	(Nothing displayed)

These operators are the same as in basic math. Greater than and less than are used to determine relationship between two numeric values. As you probably learned in elementary school, if one number is not greater than another then it is either less than or equal to that number. And visa-versa.



Numeric Interpretation of String Data

Normally, when you use $>$ or $<$, you are testing numeric values. However, unlike other languages, if any of the values being compared are strings, M will not give you an error but will convert the string to a number following the rules of automatic mode conversion.

Reading a string from left to right, if the leftmost character is a digit, +, -, or decimal point, M will continue to look at the string. M will look at the second character; if it is a digit, M will continue. If M encounters a non-numeric character, M stops.

VA M PROGRAMMING

Introduction

The numeric value of a string will be the + or - and any digits up to the first nonnumeric character. The "." (point or period) character may be the exception depending on where it is located. If it is located between numeric characters, it is a decimal point and part of the value; if it is between nonnumeric characters it is a period.

If the first character of a string is not a +, -, decimal point or digit, the numeric interpretation of that string is zero.

The numeric interpretation of a null string ("") is zero.

If M encounters a string that contains valid exponential notation (e.g., E2), M will recognize it as a valid numeric value.

<i>Sample M Code</i>	<i>Output</i>
WRITE +"ABC"	0
WRITE +"1E2XX"	100
WRITE +" "	0
WRITE +"234R"	234
WRITE +"B3"	0
WRITE +"-15XX"	-15
WRITE +"15-XX"	15
WRITE +".5N"	.5
WRITE +"N"	0

Note: Putting the unary operator + in front of a string will force a numeric interpretation.

In the M code sample above, we see the statement WRITE +"ABC" and the output being 0. This is because the literal character A evaluates to a zero since it is nonnumeric.

The second sample shows us the scientific notation where we have a literal string starting with a 1 followed by the letter E, the number 2 and two letter Xs. When M evaluates this statement it will see the numeric value of 1, followed by the letter E and another numeric value. This is a valid numeric expression so the M interpreter will do the conversion to a decimal value and disregard the meaningless Xs at the end of the string.

String Relations

- `A=B` If A is equal to B, the truth value is 1.
- `A[B` If the string A contains the string B, the truth value is 1.
- `A]B` If the string A follows B in ASCII collating sequence, the truth value is 1.

Equals Operator

The Equals operator is a string operator, which means that the values being compared are compared character-by-character from left to right.

M Code Sample

Output

<code>SET A="HELLO"</code> <code>IF A="HELLO" WRITE !,"Both are equal"</code>	Both are equal
<code>SET A="HELLO!!"</code> <code>IF A="HELLO" WRITE !,"Both are equal"</code>	(Nothing is displayed)
<code>SET A="HeLLO"</code> <code>IF A="HELLO" WRITE !,"Both are equal"</code>	(Nothing is displayed)
<code>SET A="HELLO "</code> <code>IF A="HELLO" WRITE !,"Both are equal"</code>	(Nothing is displayed)

This character-by-character comparison is done of the ASCII values of each character, which means that the evaluation is case sensitive, where an upper case A is not equal to a lower case a. Also, there is no evaluation done to remove any meaningless spaces at the ends of strings. We see these in the code samples displayed.



VA M PROGRAMMING

Introduction

Because Equals is a string operator, any numeric value stored as a string will not be converted.

M Code Sample

```
SET CODE="09"  
IF CODE=9 WRITE !,"Both are equal"      (Nothing is displayed)
```

Output

In the sample above, the variable containing the string 09 is not equal to the numeric value 9.

If you are comparing numeric values and want to make sure that both values are truly numeric, place a + in front of the variable to force numeric interpretation before comparison.

In the example below, unlike the previous example, the variable has numeric interpretation forced on it by the plus sign.

M Code Sample

```
SET CODE="09"  
IF +CODE=9 WRITE !,"Both are equal"      Both are equal
```

Output

Contains Operator

The Contains operator checks to see if the first string contains the characters in the second string. If the characters in the second string appear exactly in the first string, the truth value is 1 or true.

M Code Sample

```
READ !,"Enter name: ",NAME  
IF NAME["^" GOTO EXIT
```

M Code Sample

```
READ !,"Do you want to continue? YES//",ANS  
IF ANS["Y" GOTO INPUT  
IF ANS["y" GOTO INPUT
```


VA M PROGRAMMING

Introduction

The Contains operator checks to see if the first string contains the characters in the second string. If the characters in the second string appear exactly as in the first string, the truth value is 1 or true.

The Contains operator is the left square bracket. An easy relation for remembering this is that it almost looks like the letter C.



VA Programming Standards and Conventions

All READs must be escapable; that is, the user should be able to type an ^ at any prompt to exit the function.

M Code Sample

Output

```
IF "ABCD"["AD" WRITE !,"Contains is true" (Nothing is displayed)
```

```
IF "ABCD"["AB" WRITE !,"Contains is true"      Contains is true
```

In the M code sample above, we see the example alluded to on the previous page. Note that the string A B C D does have the individual letters A and D. It does not have the string A D.

There is one fact about contains that you will need to know later in the course: the null string (") is contained in all strings, as shown in the M code sample below.

M Code Sample

Output

WRITE "MUD" [" "	1
WRITE "^" [" "	1
WRITE 123 [" "	1
WRITE "ABCXYZ" [" "	1

VA M PROGRAMMING

Introduction

Follows Operator

Each character (as represented on a standard computer keyboard) has a decimal value associated with it called an ASCII code. The collection of all possible characters and codes is called the ASCII character set. The relative positions of characters within the set are called ASCII collating sequence. When you read the table from lowest decimal codes to highest codes, you can say that higher ASCII codes follow lower ASCII codes. For example, the character A follows the digits. In general, every character follows the space, letters follow digits, and lowercase letters follow uppercase letters.

Partial ASCII Chart

Decimal Code	Character
0	null
1-31	control characters (unprintable)
32	space
33-47	some punctuation
48-57	0-9
58-64	more punctuation
65-90	A-Z
97-122	a-z



The follows operator is the right square bracket. Its purpose is to use the ASCII collating sequence to determine if one string follows the other. Because this is a string operator, strings are compared character-by-character from left to right.

<i>M Code Sample</i>	<i>Output</i>
WRITE "a"] "A"	1
WRITE "A"] 1	1
WRITE "A"] "B"	0



VA M PROGRAMMING

Introduction

If the strings being compared are comprised of multiple characters, M starts with the first characters of each and compares their ASCII values.

M Code Sample

```
SET STRING1="FIRST, LAST"
SET STRING2="LAST, F"
IF STRING1]STRING2 WRITE !, "Follows is true"
```

STRING2	LAST, F
STRING1	LAST, FIRST

M starts
comparing here.
Since letters are the
same, it continues until it
reaches the first inequality.

The ASCII codes of I and null are compared.
ASCII of I is 73
ASCII code of null is -1

The question now becomes: is 73 > -1 ? Because it is, the truth
value is 1.

Note in the sample above how the comparison is done.

The ASCII code of the letter I is compared to the null character.
The ASCII code of A is 73.
The ASCII code of null is -1.

Boolean Expressions

Boolean (Logical) Operators

Boolean operators are used to combine the results of separate logical comparisons.

AND Operator

The M language has three Boolean operators. We will examine the AND operator first. In M expressions, the & sign (ampersand) is used as the symbol for the AND operator.

Boolean expressions evaluate to either 1 or 0, representing true or false respectively. They are sometimes referred to as logical expressions.

Boolean operators combine the values of two or more Boolean expressions or relational comparisons, by well-defined rules, and evaluate to a single Boolean value.

When combining expressions with the AND operator, the final truth-value is determined according to the rules shown below.

Expression 1	Expression 2	Exp. 1 AND Exp. 2
1 (T)	1 (T)	1 (T)
0 (F)	0 (F)	0 (F)
1 (T)	0 (F)	0 (F)
0 (F)	1 (T)	0 (F)

The rules for the resulting value when Boolean expressions are combined with the AND operator can be shown in tabular form. For the table shown above, it is assumed that two individual expressions have been evaluated resulting in the truth value listed under columns labeled Expression 1 and Expression 2.

When the results are combined with the AND operator, the truth-value given in the last column is the final result. This table may be summarized by stating that the final truth-value is true if and **only** if both Expression 1 and Expression 2 are true. The final value is false for every other possible combination.

VA M PROGRAMMING

Introduction

The second expression and any successive expressions used in a combined operation must be enclosed in parentheses (). You can optionally enclose the first expression in parentheses.

M Code Sample

```
IF SEX="F"&(STATUS="M") DO MARRFEM
IF SEX="M"&(STATUS="M") DO MARRMALE
```

Let's examine an important syntactical rule for writing M code using Boolean operators. The sample code above illustrates the rule. When combining expressions, the second expression and any successive expressions **must** be enclosed in parentheses. When this rule is not followed, an unintended, and often unpredictable, final truth-value can result.



Let's look at another example. The table below displays the average daily total attendance at a park under the column labeled All and the number that spent time swimming under the column labeled Pool for two different age groups, those over 18 and those 18 and younger. This data can be used to write M code to specify the rate, that is, number per day, of attendance and the rate of those swimming. In the M code, the variable type represents the level of participation. A is for attending and P is for entering the pool.

Table of rates:

Age Group	----- Rates/Day -----	
	All	Pool
18 or under	14	9
over 18	16	11

M Code:

```
RATE ;
IF AGE<18&(TYPE="A") SET RATE=14
IF AGE=18&(TYPE="A") SET RATE=14
IF AGE>18&(TYPE="A") SET RATE=16
IF AGE<18&(TYPE="P") SET RATE=9
IF AGE=18&(TYPE="P") SET RATE=9
IF AGE>18&(TYPE="P") SET RATE=11
```

VA M PROGRAMMING

Introduction

OR Operator

The next Boolean, or logical, operator that we will discuss is the OR operator. The exclamation mark (!) is used to designate the OR operator.

When combining expressions using the OR operator, the final truth-value is governed by rules illustrated in the chart below.

The table below illustrates the rules for combining two expressions with the OR operator. In the table, the possible truth values for expressions 1 and 2 are listed. When the expressions are combined, the final truth-value is given in the last column. The table may be summarized by stating that when either expression is true then the final combined truth-value will be true. The final value is false only when both expressions are false.

Expression 1	Expression 2	Exp. 1 OR Exp. 2
1 (T)	1 (T)	1 (T)
0 (F)	0 (F)	0 (F)
1 (T)	0 (F)	1 (T)
0 (F)	1 (T)	1 (T)

The second expression must be enclosed in parentheses (). You can optionally enclose the first expression in parentheses.

M Code Sample

```
IF PAYCODE=1!(PAYCODE=2) DO REGPAY  
IF PAYCODE=3!(PAYCODE=4) DO OVTPAY
```

Above, is a sample of M code using the OR operator. When writing M code with logical operators, always enclose the second and successive expressions in parentheses. This is an important syntactical rule to ensure the proper result, since M evaluates expressions from left to right.



VA M PROGRAMMING

Introduction

NOT Operator

Next, we will examine the NOT operator which is designated by the apostrophe ('). The NOT operator reverses the truth-value of a variable or constant. In the example shown below, not true becomes false and not false becomes true. Perhaps that is obvious, but the truth-value of variables is not always as obvious.

'1 (NOT T) = 0 (F)

'0 (NOT F) = 1 (T)

M Code Sample

```
IF SEX'="M" DO FEMALE  
IF SEX'="F" DO MALE
```

Also, the NOT operator may proceed a relational operator to reverse the outcome. In the example shown above, the IF statements are testing for sex **not** equal m and sex **not** equal f to determine which program module to execute.

The NOT operator can be used with all relational operators.

- '> not greater than (equivalent to less than or equal)
- '< not less than (equivalent to greater than or equal)
- '= not equal (equivalent to less than or greater than)
- '[not contain
- '] not follow
- '? not match
- ']] not sort after

The NOT operator may be used with all relational operators. The syntax and explanation are shown above. The NOT operator extends the power of relational comparisons and often reduces the number of expressions needed to obtain a final truth-value.

VA M PROGRAMMING

Introduction

We can shorten the code from a previous example by using the NOT operator.

Original Code

```
RATE ;  
IF AGE<18&(TYPE="A" ) SET RATE=14  
IF AGE=18&(TYPE="A" ) SET RATE=14  
IF AGE>18&(TYPE="A" ) SET RATE=16  
IF AGE<18&(TYPE="P" ) SET RATE=9  
IF AGE=18&(TYPE="P" ) SET RATE=9  
IF AGE>18&(TYPE="P" ) SET RATE=11
```

Code Using the NOT Operator

```
RATE ;  
IF AGE'>18&(TYPE="A" ) SET RATE=14  
IF AGE>18&(TYPE="A" ) SET RATE=16  
IF AGE'>18&(TYPE="P" ) SET RATE=9  
IF AGE>18&(TYPE="P" ) SET RATE=11
```

The illustration above shows how the NOT operator may be used to shorten M program code. A previous example is repeated and then rewritten using the not operator.

Note that the two statements that SET rate to 14 in the original code, using age less than 18 and age equal to 18, are replaced with one statement using age NOT greater than 18.

Also note that the two statements that SET rate to nine in the original code are replaced by one statement using age NOT greater than 19.



VA M PROGRAMMING

Introduction

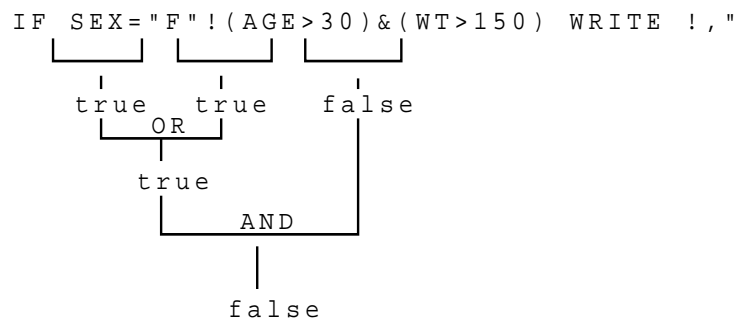
Combining Logical Operators

When you mix the Boolean operators AND and OR, M determines the final truth-value by stepwise evaluation of each comparison from left to right.

Now, let's look at a more complex M statement using Boolean operators. The variables sex, age and weight are SET in the first line of code in the example below. The IF statement first determines that the expression sex equals f is TRUE. Then the expression in parentheses, age greater than 30, is found to be TRUE. The first two expressions are combined with the OR operator, which evaluates to TRUE. Next, the last expression, weight greater than 150, is found to be FALSE. Last, this false value is combined with the AND operator to the true value from the OR combination of the first two expressions, resulting in a final false value.

M Code Sample

```
SET SEX="F",AGE=65,WT=130
```



Single Variable Conditions

All of the comparisons so far have been accomplished using a construct of operand operator operand. `<operand><operator><operand>`

The two operands or variables are compared according to the particular operator and a truth-value is found.

For example:

```
A=B
A>B
A[B
```

VA M PROGRAMMING

Introduction

However, M will derive a truth-value for a single variable, without the use of an operator, according to the following rules:

- If the value of the variable is a string, the string is first converted to a numeric value following the rules of automatic mode conversion.
- If a variable's numeric interpretation is 0, the truth-value is 0.
- If a variable's numeric interpretation is a value other than 0, the truth-value is 1.

Look at some M code that illustrates evaluating a single variable to a truth-value. In this sample code below, three variables, STRING1, STRING2, and STRING3, are SET. Then, these variables are made the arguments in IF statements. If the IF statement evaluates to TRUE then TRUE is printed, otherwise, there is no output. STRING1 evaluates to a truth-value of 0, STRING2 evaluates to a truth-value of one and STRING3 evaluates to a truth-value of 1. Take a moment to study these lines of code to make sure you understand the conversion of a string to a truth-value and to make sure you agree with the output.

Sample M Code

Output

SET STRING1="ABC"	
SET STRING2="9ABC"	
SET STRING3=15	
IF STRING1 WRITE "TRUE"	none
IF STRING2 WRITE "TRUE"	TRUE
IF STRING3 WRITE "TRUE"	TRUE
IF 'STRING1 WRITE "TRUE"	TRUE
IF 'STRING2 WRITE "TRUE"	none

One practical use of these single variable conditions is to create and use variables as ***"flags"*** in a routine. For example, we might create an "error flag" called ERR. We determine that it should be set to 0 at the beginning of the routine (called initializing the flag). At any point in the routine where we detect an error, we SET ERR=1 to indicate an error has occurred. If we follow this practice, the only two values ERR can ever have is 0 or 1. Because of this, we can then take a shortcut on the IF statement and use ERR as a single variable condition.

VA M PROGRAMMING

Introduction

M Code Sample

```
INIT      ;
          SET ERR=0
INPUT     ;
          READ !,"Enter sex (M or F): ",SEX
          IF SEX["^" GOTO EXIT
          DO CHECK
          IF ERR DO MSG1 GOTO INPUT
          QUIT
CHECK      ;
          IF SEX'="M"&(SEX'="F") SET ERR=1
          QUIT
MSG1      ;
          WRITE !,"Please enter correct code M or F"
          SET ERR=0
EXIT      QUIT
```

The sample code above sets ERR to 0, prompts the user to enter their SEX, and then validates the value of the variable SEX. If SEX is neither a capital M nor a capital F, then ERR is set to 1. When ERR is set to 1, the program branches to a subroutine that informs the user that the wrong value has been entered and resets ERR to 0. The user is then asked to reenter the value of SEX.

Time-Out on READ

You may want to give the user a limited amount of time by which to respond to a request. Use a form of the READ command that appends a colon (:) to the variable name followed by an approximate number of seconds to wait. This number of seconds can be a numeric constant or a variable.

If the specified time is exhausted and the user has not responded, \$TEST is set to 0. If the user does time-out, we want to exit this section of the code (or exit the routine depending on where we are at the time). We know that the IF command will let us conditionally execute code depending on the truth-value of the IF argument. The problem here is that if the user times-out, \$TEST is set to 0 which is the opposite of TRUE. To make the IF command work for us here, we can use the NOT operator (!) to reverse the truth-value of \$TEST. That is, if the user times-out, \$TEST will be set to 0 but the NOT operator changes it to a 1 for the IF command and we will GOTO EXIT. If the user does not time-out, \$TEST will be set to 1 and the NOT operator will reverse it to a 0 so we skip to the next line.

VA M PROGRAMMING

Introduction

Another way of remembering that we must use IF '\$T' is that we ask the question "Did the user not respond in time?" If the answer is TRUE, we exit and if it's FALSE we continue with the routine.

M Sample Code

```
SET DTIME=600
BEGIN      ;
  READ !!,"Enter number of degrees: ",XDEGREES:DTIME
  IF '$T!(XDEGREES["^") GOTO EXIT
```

In the example above, the user is prompted to enter degrees, but it must be entered before 600 seconds elapses. If the user did not respond in time, the truth-value is set to 0. The NOT \$TEST in the IF statement reverses the value to one and the program exits. Note also that the IF statement determines whether the user wishes to exit by typing an up arrow in response to the prompt for degrees.



VA Programming Standards and Conventions

All READs must be timed. In the VA, the variable DTIME is used to hold the timing and is created for you when you log on to a DHCP system. DTIME is a special VA variable so it is appropriate that it begins with the letter D.

DTIME should not be KILLED.

Postconditionals

One of the strengths of the M language is that there are often ways you can reduce the amount of code required. One way is to attach conditional expressions to commands or arguments, thereby eliminating the need for a preceding IF command to test the condition. Thus, expressions appended to a command or argument are called **postconditionals** because they follow the command or argument.

Note:

The value of \$TEST is not changed when postconditionals are executed.

VA M PROGRAMMING

Introduction

Postconditionals on Commands

You attach a colon (:) followed by the conditional expression to an argument or command.

Postconditionals can be added to all M commands except IF, ELSE, and FOR.

If the condition is TRUE, the command is executed. If the condition is FALSE, that command and all its arguments are skipped.

M Code Sample

IF Command

```
IF X=1 SET TIME=15
IF PAY>0 WRITE !,"Net is ",NET
IF OPT=1 DO ADDIT
IF OPT=5 QUIT
```

Postconditional

```
SET:X=1 TIME=15
WRITE:PAY>0 !,"Net is ",NET
DO:OPT=1 ADDIT
QUIT:OPT=5
```

Note about QUIT: QUIT is an argumentless command (a postconditional on a QUIT is not an argument). Therefore, if you follow any QUIT with another command on the same line, you must have two spaces after it as shown below (the symbol |_| indicates a space):

M Code Sample

```
QUIT:OPT=5 |_| |_| GOTO ASK
```

The graphic above shows corresponding lines of code that accomplish the same task. One line uses an IF statement to test a condition and the other uses a postconditional. Examples using the SET, WRITE, DO and QUIT commands are shown. If the condition is TRUE, the command is executed. If the condition is FALSE, that command and all of its arguments are skipped.

It is clear that the amount of code is somewhat reduced, but more importantly, an additional command, the IF statement, is not executed. Note that had any additional commands followed the QUIT with the postconditional, two spaces would be needed to separate the next command. This is an example of the argumentless form of the QUIT command.



VA M PROGRAMMING

Introduction

Postconditionals on Arguments

You can attach a colon (:) followed by a conditional expression to some commands.

There are only three commands with which you can use postconditionals on arguments: DO, GOTO, and XECUTE.

If the condition is TRUE, the command and that argument is executed. If the condition is FALSE, the argument is skipped. If there are other arguments, they are executed.

M Code Sample

IF Command

```
IF (ANS=" ") ! (ANS="YES") GOTO BEGIN  
  
IF OPT=1 DO FIRST  
IF OPT=2 DO SECOND  
  
IF OPT=1 DO FIRST  
IF OPT=2 DO SECOND
```

Postconditional

```
GOTO BEGIN: (ANS=" ") ! (ANS="YES")  
  
DO FIRST: OPT=1  
DO SECOND: OPT=2  
  
DO FIRST: OPT=1, SECOND: OPT=2
```

Look at the examples above. It is the last one that is the most important for this illustration. Note that the conditional execution of the DO statements is reduced to one statement with postconditionals on the arguments. It is permissible to have some arguments in the list with postconditionals and some without. The arguments without postconditionals will be executed unconditionally.



Menus and Subroutines

If you are designing a system that offers the user different options from which to select, one way to present the selection is to display a menu, which is a list of the options. Then you can organize the routine so that the code for each function is kept together in what are called **subroutines**.

VA M PROGRAMMING

Introduction

Subroutines begin with a line label and ends with a QUIT. Subroutines are called (executed) through the use of the DO command. When M encounters the DO command, it goes to the label specified and executes each of the commands listed until it encounters a QUIT. When the QUIT is encountered, M returns to the place where the subroutine was called and continues processing from that point.

Sample output for a short menu:

```
Enter 1 for Fahrenheit to Celsius conversion
Enter 2 for Celsius to Fahrenheit conversion
1
```

Note: The underline indicates that the user types that response.

In the example shown above, two options are displayed in which the user is instructed to type the number one or two to select the action to be taken.

M Code Sample ***(excerpt from a complete routine)***

```
MENU ;
  WRITE !!,"Enter 1 for Fahrenheit to Celsius conversion"
  WRITE !,"Enter 2 for Celsius to Fahrenheit conversion"
  READ !,CODE:DTIME IF '$T!(CODE["^") GOTO EXIT
  IF CODE<1!(CODE>2) WRITE !,"Enter 1 or 2" GOTO MENU
PICK ;
  DO FTOC:CODE=1,CTOF:CODE=2
  ...
  ..other code
EXIT ;
  KILL CODE
  QUIT
FTOC ;
  .    (calculations for Fahrenheit to Celsius)
  .
  QUIT
CTOF ;
  .    (calculations for Celsius to Fahrenheit)
  .
  QUIT
```

The graphic above shows the M code that performs the functions just introduced. The choices are printed, the user input is accepted and validated, and if valid, the selected function is executed. If not, the user is prompted to reenter the selection. Take a moment to study this example.

VA M PROGRAMMING

Introduction

\$SELECT Function

The \$SELECT function is another shortcut we can use when evaluating multiple conditionals to make a choice or selection in a routine.

We construct a \$SELECT statement by creating a sequence of expressions made up of:

<comparison>:<value to be used if comparison is true>

These expressions are enclosed in () after \$SELECT. M evaluates these expressions from left to right until it encounters one that evaluates to TRUE. When M finds a TRUE, it will stop evaluating the \$SELECT.

The \$SELECT and all its expressions must be on the same statement. (It can, however, be a wrap-around line.)

If M should happen to go through all the expressions and not find a TRUE, your routine will stop in its tracks! For that reason, we must always put what is called a default (or escape) true value at the end of the expression, such as:

1:<value to be used if no comparison listed is true>



M Code Sample

Without \$SELECT:

```
IF CODE="F" WRITE !,"FEMALE"  
IF CODE="M" WRITE !,"MALE"  
IF CODE'="M"&(CODE'="F") WRITE !,"ERROR"
```

With \$SELECT:

```
WRITE !,$SELECT(CODE="F": "FEMALE",CODE="M": "MALE",1:"ERROR")
```

M Code Sample

```
TYPE ;  
  WRITE !,"Enter type of membership"  
  READ !,"A=all facilities, P=pool only:",TYPE:DTIME  
  IF '$T!(TYPE["^") GOTO EXIT  
  SET XDESCR=$S(TYPE="A": "All",TYPE="P": "Pool",1:"ERROR")  
  IF XDESCR="ERROR" WRITE !,"Reenter" GOTO TYPE
```


VA M PROGRAMMING

Introduction

The preceding M code illustrates the use of the \$SELECT statement.

In the first example, a selection using IF statements is contrasted to that using a \$SELECT statement. Note that the code is much shorter, one line instead of three, and that the only values allowed for the variable CODE is the capital f or capital m. If the variable CODE is neither, then ERROR is displayed.

The second example is similar. A value for the variable TYPE is entered by the user. Only a capital A or capital P is allowed. Otherwise, the select statement SETs the variable XDESCR to error and the user is asked to reenter.

Note in both \$SELECT statements the escape value to be used when no preceding conditional expression evaluates to true.

Another common use of the \$SELECT is to express a situation where it is certain there are only two options. For example, if we knew for sure that the field CODE from the example above could only have the values M or F, we could use the \$SELECT below.

M Code Sample

```
WRITE !,$S(CODE="F":"FEMALE",1:"MALE")
```

In another situation, we may have prompted the user for a name and street address. We want to allow the user to bypass the street address if it is not known. When we are producing output, however, we want to print the word UNKNOWN if the street address field does not have a value.

M Code Sample

```
WRITE !,NAME  
WRITE !,$SELECT(ADDRESS=" ":"UNKNOWN",1:ADDRESS)
```

Using Sentinel Values

In some systems, you will want the user to enter data over and over again. You will need to have them signal the routine that they are done. This signal is called a sentinel or ending value. Shown below is the method most often used by the VA.

M Code Sample

```
READ !,"Enter name or press ENTER to quit: ",NAME:DTIME  
IF '$T!(NAME["^")!(NAME=" ") GOTO EXIT
```

In the example, three sentinel values signal the end of data entry. The most common one encountered pressing <ENTER>. In this case the variable name has the null value. The other is the up arrow, which signals the end of data entry. Of course, if time allowed for input lapses the \$TEST of false is a more indirect signal, but just as effective.

Sample Routines

Sample Routine Number 1

Pseudocode

1. Read the number of degrees to convert.
2. Display the menu:

 Enter 1 for Fahrenheit to Celsius.
 Enter 2 for Celsius to Fahrenheit.
3. Accept user's choice and check to make sure it is valid.
4. If their choice was 1, do the Fahrenheit to Celsius conversion (step 6). If their choice was 2, do the Celsius to Fahrenheit conversion (step 7).
5. Ask if they want to do another. If so, go back to step 1. If not, go to step 8.
6. Convert Fahrenheit to Celsius:

 $\text{degrees} - 32 * 5/9$
7. Convert Celsius to Fahrenheit:

 $\text{degrees} * (9/5) + 32$
8. KILL any variables and QUIT.

VA M PROGRAMMING

Introduction

Sample Routine Number 1

M Code

```
VECS233      ;ATL/ACE PROGRAMMER-TEMPERATURE CONVERSION ;1/1/90
              ;;1
              ;      Variable List
              ;
              ;      DTIME.....Delay time for READ (since it is a special
              ;      VA variable, it is not to be KILLed)
              ;      TEMP .....Converted temperature
              ;      XDEGREES....degrees input by user
              ;      ANS.....answer to "Do you want to do another?"
              ;      CODE.....user's menu choice number
              ;
BEGIN          ;
              READ !!, "Enter number of degrees: ", XDEGREES:DTIME
              IF '$T!(XDEGREES["^") GOTO EXIT
MENU          ;
              WRITE !!, "Enter 1 for Fahrenheit to Celsius"
              WRITE !, "Enter 2 for Celsius to Fahrenheit"
              READ !, CODE:DTIME IF '$T!(CODE["^") GOTO EXIT
              IF CODE<1!(CODE>2) WRITE !, "Enter 1 or 2" GOTO MENU
PICK          ;
              DO FTOC:CODE=1,CTOF:CODE=2
AGAIN         ;
              ;      Below we have used the Lesson 1 way of ending
              ;      a program
              ;
              READ !, "Do you want to do another? YES//", ANS:DTIME
              IF '$T!(ANS["^") GOTO EXIT
              GOTO BEGIN:(ANS="")!(ANS="YES")!(ANS="Y")
              GOTO EXIT
FTOC          ;
              SET TEMP=(XDEGREES-32)*5/9
              WRITE !!, "Celsius temperature is: ", TEMP
              QUIT
CTOF          ;
              SET TEMP=XDEGREES*(9/5)+32
              WRITE !!, "Fahrenheit temperature is: ", TEMP
              QUIT
EXIT          ;
              KILL TEMP, XDEGREES, ANS, CODE
              QUIT
```

VA M PROGRAMMING

Introduction

Sample routine number 2

Pseudocode

1. Have the user enter the member's name or press <ENTER> to quit. If the user presses <ENTER>, go to step 7.
2. Have the user enter the type of membership where A is all facilities and P is just the pool.
3. Have the user enter the member's age.
4. Have the user enter the number of days that the facility was used.
5. Calculate the member's charge based on the table below:

Age Group	----- Rates/Day -----	
	All	Pool
18 or under	14	9
over 18	16	11

Multiply the rate times the number of days.

6. Write the member's charge. Go back to step 1.
7. KILL any variables and QUIT the routine.

VA M PROGRAMMING

Introduction

Sample Routine Number 2

M Code

```
VECS235      ;ATL/ACE PROGRAMMER-HEALTH CLUB MEMBERSHIP ;1/1/90
              ;;1
              ;      Variable List
              ;
              ;      NAME.....Member's name
              ;      DTIME.....Delay time for READ (not to be KILLed)
              ;      TYPE.....Type of membership
              ;      XDESCR.....TYPE converted to words
              ;      AGE.....Age of member
              ;      XDAYS.....Nr. of days facility used
              ;      RATE.....Rate from table
              ;      AMOUNT.....Total amount to be paid
BEGIN        ;
              WRITE !!,"Enter member's name or"
              READ !,"Press Enter to quit: ",NAME:DTIME
              IF '$T!(NAME["^") GOTO EXIT
              GOTO EXIT:NAME=""
TYPE         ;
              WRITE !!,"Enter type of membership"
              READ !,"A=all facilities, P=pool only: ",TYPE:DTIME
              IF '$T!(TYPE["^") GOTO EXIT
              SET XDESCR=$S(TYPE="A":"all",TYPE="P":"pool",1:"ERROR")
              IF XDESCR="ERROR" WRITE !,"Reenter" GOTO TYPE
AGE          ;
              READ !,"Enter age: ",AGE:DTIME
              IF '$T!(AGE["^") GOTO EXIT
DAYS         ;
              READ !,"Enter days the facility was used: ",XDAYS:DTIME
              IF '$T!(XDAYS["^") GOTO EXIT
RATE         ;
              IF AGE'>18&(TYPE="A") SET RATE=14
              IF AGE>18&(TYPE="A") SET RATE=16
              IF AGE'>18&(TYPE="P") SET RATE=9
              IF AGE>18&(TYPE="P") SET RATE=11
CALC         ;
              SET AMOUNT=XDAYS*RATE
OUTPUT       ;
              WRITE !!,"The charge for ",NAME,"'s membership is $",AMOUNT
              WRITE !,"which covers their use of ",XDESCR," facilities"
              GOTO BEGIN
EXIT         ;
              KILL AMOUNT,XDAYS,RATE,XDESCR,TYPE,AGE,NAME
              QUIT
```

VA M PROGRAMMING

Introduction

Sample Routine Number 2

Below you will find a sample interaction, that is, how the routine works when it is run. The double underlines show what the user would enter. You will notice that we tested the routine by typing every possible combination of age and type of membership.

>D ^VECS235

Enter member's name or
Press Enter to quit: FIRST1 LAST1

Enter type of membership
A=all facilities, P=pool only: A
Enter age: 17
Enter days the facility was used: 1

The charge for FIRST1 LAST1's membership is \$14
which covers their use of All facilities

Enter member's name or
Press Enter to quit: FIRST2 LAST2

Enter type of membership
A=all facilities, P=pool only: A
Enter age: 18
Enter days the facility was used: 1

The charge for FIRST2 LAST2's membership is \$14
which covers their use of All facilities

Enter member's name or
Press Enter to quit: FIRST3 LAST3

Enter type of membership
A=all facilities, P=pool only: A
Enter age: 19
Enter days the facility was used: 1

The charge for FIRST3 LAST3's membership is \$16
which covers their use of All facilities

VA M PROGRAMMING

Introduction

Sample Routine Number 2

Enter member's name or
Press Enter to quit: FIRST4 LAST4

Enter type of membership
A=all facilities, P=pool only: P
Enter age: 17
Enter days the facility was used: 1

The charge for FIRST4 LAST4's membership is \$9
which covers their use of pool facilities

Enter member's name or
Press Enter to quit: FIRST5 LAST5

Enter type of membership
A=all facilities, P=pool only: P
Enter age: 18
Enter days the facility was used: 1

The charge for FIRST5 LAST5's membership is \$9
which covers their use of pool facilities

Enter member's name or
Press Enter to quit: FIRST6 LAST6

Enter type of membership
A=all facilities, P=pool only: P
Enter age: 19
Enter days the facility was used: 1

The charge for FIRST6 LAST6's membership is \$11
which covers their use of pool facilities

Enter member's name or
Press Enter to quit: <ENTER>
>

VA M PROGRAMMING

Introduction

Assignment (Optional)

In General

- Follow the VA Programming Conventions as illustrated in this lesson (especially note that all READs must be timed and escapable). 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 lesson.
- Save the routine under the namespace assigned to you by your facility IRM Service or IT entity.

Assignment Specifications

Using the previous sample routine as a guide, produce an interactive routine to calculate employee gross pay according to the following criteria:

- Have the user enter the employee's name or press <ENTER> to stop the routine.
- Have the user enter a one-letter code to indicate the employee's status: F=fulltime, P=part-time.
- Have the user enter the employee's hours.
- Have the user enter the employee's rate of pay.
- Calculate the employee's gross pay:
 - If the employee is fulltime and hours are > 40, calculate:
regular pay=40*rate
overtime pay=(hours-40)*(rate*1.5)
gross pay=regular pay + overtime pay
 - If the employee is fulltime and hours are = 40 or less than 40, calculate gross pay=hours * rate
(no overtime)
 - If the employee is part-time, don't calculate overtime pay even if hours are > 40: gross pay=hours * rate.

VA M PROGRAMMING

Introduction

- Produce the following output (your routine will supply what's in < >):
Gross pay for <employee name>, <the WORD part-time or fulltime> employee, is \$<gross pay>.
- After one employee is processed, take the user back to the name prompt.
- Use the \$SELECT to convert the F or P code into words for the output. Use postconditionals wherever possible.

Optional Exercise

Produce a routine that will convert US dollars to pounds, marks, francs, and lira.

- Have the user enter the dollars they want to convert.
- Give them a menu listing the conversion options.
- Convert their dollars using the following values (which may be outdated; you can use the current values if you know them).

```
pounds=dollars*.94
marks=dollars*2.40
francs=dollars*7.20
lira=dollars*1439
```

- Display their result.