Access VBA Made Easy

Conditionals and Branching

07

www.accessallinone.com

This guide was prepared for AccessAllInOne.com by:
Robert Austin

This is one of a series of guides pertaining to the use of Microsoft Access.

Contents

Conditionals and Branching

Introduction

In everyday life we create scenarios for ourselves and base our actions upon them. An example would be someone saying “If it rains tomorrow, we will stay in; otherwise we will go to the park”.

This type of statement is known as a conditional (in both human and computer language). The idea is that we have a statement that can be evaluated to true or false and then act based on that evaluation.

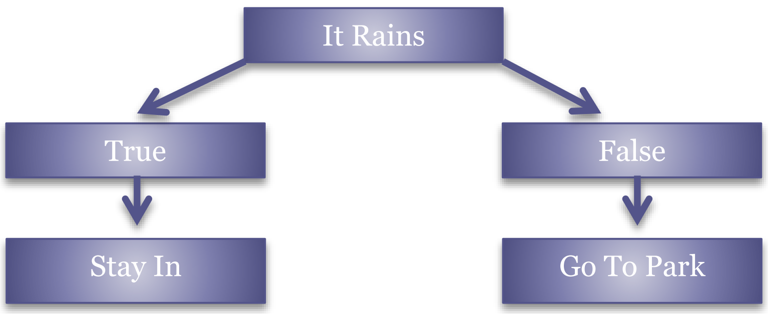

Figure 7.1

Figure 7.1 shows that we are evaluating what will happen if it rains or not. The concept of raining is either true (it is raining) or false (it is not raining) and depending on the answer we either stay in or go out.

Programming languages work in much the same way. A statement is evaluated to be either true or false and the code is executed depending on the answer.

All conditionals use an operator in an expression which concludes that the expression is either True or False. We will start with the straightforward If statement and “=” operator.

If…Then, Evaluating Expressions, Operators, Statement Blocks

Diving straight into some examples, you can execute the following in the immediate window of the VBA Editor.:

```
a = 10 : If a=10 Then Debug.Print “a=10”
```

Figure 7.2

The <u>If statement</u> is a very simple statement that asks a straightforward question – *is an <u>expression</u> True or False?* If the <u>expression</u> is True then execute some code – in this case the “a=10” is printed in the immediate window. You can check this by changing the value of “a” to any other number and re-execute.

The <u>expression</u> above uses what is termed an <u>operator</u>, the “=”equals operator. To clearly demonstrate what the *expression part* of an If statement is, the above has been rewritten with brackets around the expression in the box below.

```
a = 10 : If (a=10) Then Debug.Print “a=10”
```

Figure 7.3

Here are some other examples. All of them evaluate to True.

```
a=20 : If a=20 Then Debug.Print “a=20”
c=5  : If c=5  Then Debug.Print “c=5”
d:10 : If d=10 Then Debug.Print “d=10”
```

Figure 7.4

“AND” Operator

We can also use the keyword <u>AND</u> which asks if two expressions are both equal to True. All the statements below evaluate to True.

```
1   a=10 : b=10 : If (a=10   And b=10)  Then Debug.Print "a and b = 10"
2   c=5  : a=5  : If (c=5)   And (a=5)  Then Debug.Print "c=5 and a=5"
3   a=12 : b=12 : If ((a=12) And (b=a)) Then Debug.Print "a and b = 10"
```

Figure 7.5

In Figure 7.5 you can see that an expression doesn't have to include actual values – numbers like 10 and 5 – but can consist of comparing variable against variable. Line 3 demonstrates this; b is never asked if it equals 12, but is asked if it equals a (which does equal 12).

Nested Expressions

Line 3 also shows that the expression ((a=10) And (b=a)) is what is termed a <u>nested expression</u>; that is, there are expressions inside expressions.

1. (a=10)
2. (b=a)
3. ( ) And ( ) which is written as ((a=10) And (b=a))

Nested expressions are more common than non-nested expressions and as programmers you will be using them everywhere. For this reason we will use nested expressions wherever possible in order to get accustomed to dealing with them. Here are some examples of nested expressions (the result of which are all False); so they do not execute the Debug statement.

```
a=11 : b=10 : If (a=10   And b=10)  Then Debug.Print "a and b = 10"
c=5  : a=4  : If (c=5)   And (a=5)  Then Debug.Print "c=5 and a=5"
a=13 : b=12 : If ((a=12) And (b=a)) Then Debug.Print "a and b = 10"
```

Figure 7.6

***Exercise:** Change the above expressions so that they evaluate to true.*

If…Then…End If

So, how do we write the IF statement in the normal code window. Let's take a look:

```
Sub standardExpressions()

Dim a As Integer

a = 10
If a = 10 Then Debug.Print "a = 10"

'The If statement is a very simple statement that asks a
'straight forward question - is an expression True or False?

'If the expression is True then execute some code - in this case
'the "a=10" is printed in the immediate window.

'You can check this by changing the value of "a" to any other
'number and re-running the code (nothing will print out).

'The expression above uses what is termed an operator,
'the "="equals operator.  To clearly show what the expression
'part of an If statement is, the above has been rewritten with
'braces around the expression in the box below.
a = 10
If (a = 10) Then Debug.Print "a = 10"

'Whatever is in brackets above, must equate to true for the
'code to run (Debug.Print "a = 10")

End Sub


```

Figure 7.6a

The output to the immediate window will be:

```
a and b = 10
c=5 and a=5
a and b = 12
```

If…Then…Else

In Figure 7.6 none of the Debug.Print statements are executed as all of the expressions evaluate to False. So, what do we do if we want to execute some code when the expression evaluates to false?

VBA extends the If…Then statement to include an Else part to tackle such situations. The Else part is executed then the expression evaluates to False. Here are some examples (put the examples in a new Module and call the procedure from the Immediate window:

```
Sub ifThenElse()
'In this sub-procedure we take a look at the
'if...then...else statement.
Dim a As Integer
a = 12

If (a = 14) Then
    Debug.Print "a equals 14"
Else
    Debug.Print "a does not equal 14"
End If
'The above if statement essentially says:

'if a equals 14, print "a equals 14" to the immediate window
'HOWEVER if a does not equal 14, print "a does not equal 14"
'to the immediate window

'The 5 line syntax above is very common and should appear
'in your code often
End Sub
```

Figure 7.7

The output to the immediate window will be:

```
a does not equal 14
```

Note: *You will notice If…Then…Else have been spread across five lines. This format is a standard used in practically every programming language to help us read code more easily.*

When using the Else part we must also end the whole If statement with the words End If. If this is not done the compile won't execute the code.

Now that we have introduced End If we can also bring in statement blocks.

If…Then…End If and Statement Blocks

Essentially all your code is divided into statement blocks. Like everything in your Sub or Functions, it is simply a way for us human's to say *here is a list of code I want executed*. A statement block is code between some start keyword and some end keyword, e.g. Sub…End Sub, If…End If, Property Get…End Property, While…End While, For…Next.

```
 ' Example of statement blocks
Sub myStatementBlock1
   ' Here we put our statement blocks
End Sub

Function myStatementBlock2
   ' Here we put our statement blocks
End Function

Function myStatementBlock3
   ' Statement Block 1
   If (expression) Then
      ' Statement Block 2
      ' Statement blocks are indented by spaces or tab to aid _
        understanding
   Else
      ' Statement Block 3
   End If
   ' Statement Block 1 continues
End Function
```

Figure 7.8

With an If…Then statement the statement block must be only one statement in length – zero or many statements are forbidden.

```
Sub myExampleSub2()
   Dim a As Integer, b As Integer, c As Integer
   If a = b And c = a Then Debug.Print "executed when evaluates to true"
End Sub
```

Figure 7.9

```
Sub myExampleSub3()
  Dim a As Integer, b As Integer, c As Integer
  ' subroutine's statement block
  If a = b And c = a Then
    ' If statement block code for True
    Debug.Print "executed when evaluates to true"
    Debug.Print "and so is this"
  End If

  ' Back to subroutine's code block
  Debug.Print "but this is outside the statement block"

  ' it is possible to separate statements using : But it makes for _
    really difficult reading,
  If a = b And c = a Then
    Debug.Print "executed when evaluates to true"
    Debug.Print "and so is this"
  End If
End Sub
```

Figure 7.10

When If is closed off with an End If then all the lines between them are a statement block. This block may contain zero, one or many statement lines or even nested statements; so a statement block may contain yet another If statement within its own blocks of code.

If...Then...ElseIf

The final extension of the If statement is the ElseIf. ElseIf is useful in that it makes nested If statements much easier to read.

The concept of the ElseIf statement is that each condition will be evaluated in order until one of them evaluates to true at which point the code for that condition (and only that condition) will be executed.

```
Sub ifThenElseIf()
'In this sub-procedure we take a look at the
'if...then...elseif statement.

Dim a As Integer
a = 7

If (a = 5) Then
    Debug.Print "a equals 5"
ElseIf (a = 6) Then
    Debug.Print "a equals 6"
ElseIf (a = 7) Then
    Debug.Print "a equals 7"
ElseIf (a = 8) Then
    Debug.Print "a equals 7"
End If

'The above if statement essentially says:

```

```
'if a equals 5, print "a equals 5" to the immediate window
'no, ok in that case:
'if a equals 6, print "a equals 6" to the immediate window
'no, ok in that case:
'if a equals 7, print "a equals 7" to the immediate window

End Sub
```

Figure 7.14

In figure 7.14, the *if statement* first evaluates whether a=5; as this returns false, it then evaluates whether a=6; as this also returns false it evaluates a=7. This returns true and the *Debug.Print "a equals 7"* runs. We now exit the *if statement* and do not evaluate a=8.

The key point here is that the conditions will be evaluated until the first condition that is found to be true. The code will then be executed and then leave the if statement.

Note: *The alternative to the ElseIf statement is the Select…Case statement later in this unit.*

Expressions: Operators

An expression is a single or collection of variables and operators that ultimately evaluate to True or False. Here we will list all the arithmetic operators with example code and introduce some operators you'll need when programming.Boolean as an Expression.

```
Sub expressionEqualsTrue()
    Dim TrueOrFalse As Boolean
    TrueOrFalse = True

    If TrueOrFalse Then
    'We only need to provide the variable here
    'We do not need to write:
    '--If TrueOrFalse = true then--
        Debug.Print "The expression is True"
    Else
        Debug.Print "The expression is False"
    End If
End Sub
```

Figure 7.15

The output to the immediate window will be:

```
The expression is True
```

In Figure 7.15, *TrueOrFalse* evaluates to true so we don't need to use *If TrueOrFalse=True Then…*

And if it evaluates to False:

```
Sub expressionEqualsFalse()
    Dim TrueOrFalse As Boolean
    TrueOrFalse = False

    If TrueOrFalse Then
    'We only need to provide the variable here
    'We do not need to write:
    '--If TrueOrFalse = true then--
        Debug.Print "The expression is True"
    Else
        Debug.Print "The expression is False"
    End If
End Sub
```

Figure 7.16

The output to the immediate window will be:

```
The expression is False
```

Arithmetic Operators

Arithmetic operators work by comparing expression A with expression B. We say comparing expressions because A and B may be nested expressions that must be evaluated first to yield an answer to the If statement, or be values themselves.

A=B	Equal To	Tests for value equality
A>B	Greater Than	Evaluates to True when A is Greater Than B
A>=B	Greater Than or Equal To	Evaluates to True when A is at least the value of B
A<B	Less Than	Evaluates to True when A is Less Than B
A<=B	Less Than or Equal To	Evaluates to True when A is at most B
A<>B	Great than Or Less than or, Doesn't Equal	Evaluates to True when A doesn't equal B

Figure 7.17

Examples of uses for these operators are in myExampleSub8 below.

```
Sub myExampleSub8()
  Dim A As Integer, B As Integer

  ' Test Greater Than
  A = 20: B = 21
  If A > B Then
    Debug.Print "A is Greater than B"
  Else
    Debug.Print "B is Greater than A"
  End If

  ' Test Less Than
  A = 20: B = 19
  If A < B Then
    Debug.Print "A is Less Than B"
  Else
    Debug.Print "B is Less Than A"
  End If

  ' Test Not Equal To
  A = 20: B = 50
  If A <> B Then
    Debug.Print "A and B are Not Equal."
  Else
    Debug.Print "A and B are Equal"
  End If

End Sub
```

Figure 7.18

Exercise: *Use the above procedure and change the values of A and B so that the other part of each If statement is executed.*

Using the operators above it can be demonstrated that nested expressions are also values.

```
1   Sub myExample9()
2
3     Dim A As Integer, B As Integer, C As Integer, D As Boolean
4     A = 12: B = 48: C = 24
5     If (C / A) = 2 Then
6       Debug.Print "A multipled by 2 = C"
7     Else
8       Debug.Print "A multipled by 2 <> C"
9     End If
10
11    A = 24: B = 24: D = True
12    If (A >= B) = D Then
13      Debug.Print "A multipled by 2 = C"
14    Else
15      Debug.Print "A multipled by 2 <> C"
16    End If
17
18  End Sub
```

```
There are two expressions in line 5: (C / A) = 2

  (C / A) is the first expression, which equates to 2
  2 = 2   is the second expression which equates to True

On line 12 there are also two expressions: (A >= B) = D

  (A >= A) is the first expression, which equates to True
  (True=D) is the second expression, which equates to True

Ultimately the expression comes down to a True or False value.
```

Figure 7.19

Arithmetic Operators on Strings

It is possible to perform the same operators to Strings as one would numbers.

```
Sub arithmeticOperators2()

    Dim A As String
    Dim B As String
    Dim C As String
    Dim D As String

    A = "farming"
    If (A = "farming") Then
        Debug.Print "A equals " & A
    Else
        Debug.Print "A does not equal " & A
    End If


    A = "1"
    B = "02"
    If (A > B) Then
        Debug.Print "A is higher than B"
    Else
        Debug.Print "B is higher than A"
    End If

    C = "a"
    D = "1"
    If (C >= D) Then
        Debug.Print C & " is equal to or greater than " & D
    Else
        Debug.Print C & " is less than " & D
    End If

End Sub
```

```
    When comparing strings, we are actually asking Access
    to say which value comes first. Imagine a list of values
    like this:
      0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M etc.
    D comes after 4 so D is "greater than" 4
```

Figure 7.20

Logical Operators

Logical operations work with Boolean expressions to yield an answer for expressions. Individually they are quite straightforward but can be brought together.

And Operator

The And operator requires 2 Boolean values, gives a True answer when both sides of the argument are also True, otherwise False. A logic table demonstrates this more clearly.

For the expression:

```
Z And X
```

	X	
Z	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

```
Sub myExample11()

  Dim X As Boolean, Z As Boolean

  X = True  : Z = True  : Debug.Print X and Z ‘ True
  X = True  : Z = False : Debug.Print X and Z ‘ False
  X = False : Z = True  : Debug.Print X and Z ‘ False
  X = False : Z = False : Debug.Print X and Z ‘ False

End Sub
```

Figure 7.21

Or Operator

The Or operator requires 2 Boolean values, gives a value of True when either side of the argument is True. A logic table demonstrates this more clearly.

For the expression:

```
Z Or X
```

	X	
Z	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

```
Sub myExample12()

  Dim X As Boolean, Z As Boolean

  X = True  : Z = True  : Debug.Print X or Z ‘ True
  X = True  : Z = False : Debug.Print X or Z ‘ True
  X = False : Z = True  : Debug.Print X or Z ‘ True
  X = False : Z = False : Debug.Print X or Z ‘ False

End Sub
```

Figure 7.22

Not Operator

The Not operator requires 1 Boolean value, gives a value of True when the argument is False, and False when the argument is True. A logic table demonstrates this more clearly.

For the expression:

`Not X`

X

TRUE	FALSE
FALSE	FALSE

```
Sub myExample13()

  Dim X As Boolean

  X = True  : Debug.Print Not X
  X = False : Debug.Print Not X

End Sub
```

Figure 7.23

Nested If Clauses

“If” statements can also be nested by putting “If” statements inside the execution block of an outer “If” statement.

```
Sub myExample14()
  Dim X As Boolean, Y As Boolean
  X = True : Y = 1

  If X Then ' Outer If Start
    Debug.Print ”X is True”

    If Y <2 then ' start of nested If statement
      Debug.Print “Y is < 2”
    Else
      Debug.Print “Y is >= 2”
    End If        ' end of nested If statement

  Else
    Debug.Print ”X is False”
  End If ' Outer If Ended
End Sub
```

Figure 7.24

Select...Case...Else

All languages have alternatives to explaining the same thing; VBA is no exception. In the area of conditionals Select...Case...Else is an alternative to If...Then...ElseIf...End If.

```
Enum StatusCode ' Status of myStoreStatus variable
  CLOSED
  OPENING
  RESTOCKING
  OUT_OF_OPERATION
End Enum

Function myExample15(storeStatus As StatusCode)

  Select Case storeStatus
    Case Is = StatusCode.CLOSED
      Debug.Print "Store CLOSED"

    Case Is = StatusCode.OPENING
      Debug.Print "Store OPENING"

    Case Is = StatusCode.OUT_OF_OPERATION
      Debug.Print "Store OUT_OF_OPERATION"

    Case Is = StatusCode.RESTOCKING
      Debug.Print "Store RESTOCKING"

    Case Else
      Debug.Print "Unknown store code:" + CStr(storeStatus)
  End Select
End Function
```

Figure 7.25

Compared to the ElseIf statement Select...Case is a bit bigger in structure, but the ease of adding new code and its regular structure is appealing in certain situations. In terms of execution speed, Select...Case carries the same cost as ElseIf.

Common Problems

Too Many conditionals

A common problem is extending a set of conditionals and making a collection of statements really difficult to read.

Note: *Where necessary don't be afraid to alter the structure of your code to improve its readability. Readability is far more important in VBA than execution time, line count or conciseness. Something that is easy to read will naturally contain fewer errors.*

Too Many Expressions

It is possible to include too many expressions and operators in a statement and get very confused about which takes precedence over another. Where possible, place brackets around your expressions to make them easier to read.

Very Long Select... Case Statements

Select Case is quite a verbose syntax to use because each Case line is accompanied by a Code Block. To reduce the length of the statement use a function or sub procedure in the code block as this will markedly improve code readability and reliability.

Questions

1. Correct the following code

```
Dim myName as String
myName = getUsername() ‘ returns user’s name

If myName = “Mat” Then MsgBox “Hi Mat”
If myName = “John Then MsgBox “Hi John”
If myName = “Sarah” Then Print “Hi Sarah”

Dim l as Integer
l = len myName
If l > 4 Amd l < 10 Then
  Debug.Print “Length of myName is  + CStr(l)
Nend If
```

2. What must an expression evaluate to?
    a. Class or Object
    b. True or False
    c. Null or Nothing
    d. Empty or Full
    e. T or F

3. Which of the following are expressions
    a. ((a+b)=c)
    b. (a) < (b-c)
    c. a)b-1
    d. a And b
    e. a Tan b

4. If A is True and C is True and B is False (True or False)
    a. Not A = False
    b. A = C
    c. Not A = B
    d. Not B = A
    e. D = A Or B : D = True
    f. A = C = Not B

5. Why is indentation a good thing?

6. How many statements can an If Statement without an End If have? How many must it have?

7. What are Nested If statements?
    a. A group of statements inside an If Statement
    b. A resting place for birds and bugs
    c. A conditional statement buried in an execution block in an Else clause.
    d. End of an If statement

8. What is wrong with the following ElseIf?

```
Enum StatusCode ' Status of myStoreStatus variable
  CLOSED
  OPENING
  RESTOCKING
  OUT_OF_OPERATION
End Enum

If myStoreStatus = CLOSED Then ' executed on close

ElseIf myStoreStatus = OPENING Then ' executed on OPENING

ElseIf myStoreStatus = CLOSED Then ' executed on Restocking

End If
```

9. Link up the Operator with the Description

1	A=B
2	A>B
3	A>=B
4	A<B
5	A<=B
6	A<>B

	A	Greater Than
	B	Equal To
	C	Less Than or Equal To
	D	Great than Or Less than or, Doesn’t Equal
	E	Greater Than or Equal To
	F	Less Than

10. Examine the following code:

```
A = ?
B = ?

If A < B Then

  Debug.Print "Rome"

ElseIf A > B Then

  Debug.Print "Paris"

Else

  Debug.Print “London”

End If
```

a) Set the value of A and B so that London is displayed

b) Set the value of A and B so that Paris is displayed

c) Using A from (b) change B so that Rome is displayed

11. Two variables A and B. Both display 1.1 when Debug prints out their value, yet they are of different data types. What types might they be?

12. Assign the following Logical Operators to the logic diagram below: And, Or, Not

    a. For an extra point, what Logical Operator might (D) be?

(A)

	X	
Z	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

(B)

	X	
Z	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

(C)

X	
TRUE	FALSE
FALSE	FALSE

(D)

	X	
Z	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	TRUE

13. Using the above table answer the following True or False questions – substitute () for their logical operator above
    a. True (A) False
    b. True (B) True
    c. (C) True
    d. True (A) ((C) False (B) True)
    e. False (D) False
    f. (C) True (A) (C) True

14. A statement block within a statement block. Explain.

15. Write the following in nicely indented code:
    If a = b And c = a Then: MsgBox "Might be true": Debug.Print "and so may this": Else: Debug.Print "It's Twins!" : End If

16. Rewrite the following as a select statement:

```
Enum Status
  INCREASE_TEMP
  DECREASE_TEMP
  WARM_UP
  COOL_DOWN
  FAN_ON
  FAN_OFF
End Enum

Function P(airconStatus As Status) As Long

```

```
  If airconStatus = INCREASE_TEMP Then
    s = 1
  ElseIf airconStatus = DECREASE_TEMP Then
    s = 2
  ElseIf airconStatus = FAN_ON Then
    s = 4
  ElseIf airconStatus = FAN_OFF Then
    s = 8
  ElseIf airconStatus = WARM_UP Then
    s = 16
  ElseIf airconStatus = COOL_DOWN Then
   s = 32
  Else
   s = 64
  End If
  P = s
End Function
```

17. What is the default value of semaphore?

```
Sub myExampleSub7(Optional semaphore As Boolean = True)
  If semaphore Then
    Debug.Print "The Semaphore is True"
  Else
    Debug.Print "The Semaphore is False"
  End If
End Sub
```

a) myExampleSub7(Not True). What is the outcome of myExampleSub7?
b) myExampleSub7(Not False And Not False). What is the outcome?
c) myExampleSub7(Not False And Not True). What is the outcome?
   a. For an extra point, what's the outcome? myExampleSub7(Not False XOR Not Not True)

18. What are the values of a, b and c ?

```
Sub J()
  Dim SMS_a As String, SMS_b As String

  SMS_a = "On the way home!" ' trick question
  SMS_b = "0n the way home!"

  Debug.Print "a="; SMS_a < SMS_b
  Debug.Print "b="; SMS_a = SMS_b
  Debug.Print "c="; SMS_a > SMS_b

End Sub
```

19. How can a With block make code easier to read?

20. Why does this equal False?

```
print CInt(20001.1) = "20001.1"
```

Answers

1.

```
Dim myName as String
myName = getUsername()  ‘ returns user’s name

If myName = “Mat” Then MsgBox “Hi Mat”
If myName = “John” Then MsgBox “Hi John”
If myName = “Sarah” Then Print “Hi Sarah”

Dim l as Integer
l = len(myName)
If l > 4 And l < 10 Then
  Debug.Print “Length of myName is “ + CStr(l)
End If
```

2. True or false

3. a) true, b) true, c) false, d) true, e) false

4. a) false, b) true, c) true, d) true, e) true, f) true

5. It aids readability which reduces the likelihood of errors.

6. 1 and 1

7. (a), (c)

8. Line 12 should read RESTOCKING not CLOSED.

9. See Arithmetic Operators on page 11

10. a) A and B must be the same
 b) A must be larger than B
 c) A must not change and B > A

11. One may be a String, one may be a single or float.

12. (A) = AND, (B) = OR, (C) = NOT, (D) = XOR

13. a) False, b) True, c) False, d) True, e) True, f) False

14. Think nested expressions!

15.

```
If a = b And c = a Then
  MsgBox "Might be true"
  Debug.Print "and so may this"
Else
  Debug.Print “It’s Twins!”
End If
```

16.

```
Enum Status
  INCREASE_TEMP
  DECREASE_TEMP
  WARM_UP
  COOL_DOWN
  FAN_ON
  fan_off
End Enum

Function P(airconStatus As Status) As Long

  Select Case airconStatus
    Case Is = Status.INCREASE_TEMP: s = 1
    Case Is = Status.DECREASE_TEMP: s = 2
    Case Is = Status.FAN_ON: s = 4
    Case Is = Status.fan_off: s = 8
    Case Is = Status.WARM_UP: s = 16
    Case Is = Status.COOL_DOWN: s = 32
    Case Else: s = 64
  End Select
  P = s

End Function
```

17. a) The Semaphore is False
b) The Semaphore is True
c) The Semaphore is False

18. a=False, b=False, c=True

19. With can make a block easier to read by taking out repetitive code. Particularly useful when you are accessing deeply nested properties of objects.

20. Because the types are not equal. If you put <> between them the answer is True.