Part 3 - Variables, Loops, Conditionals

Variables

Data Types

Type Checking

Type compatibilities

Arithmetic Operators

Option EXPLICIT and STRICT

Constants

Input and Output in VB

If - then - else
   Nested IF Statement
   ElseIf Statement

Select Case Statement

Loops
   Do Loops
   For Loop

Boolean Variables, Operators and Expressions
   Boolean Expressions

Boolean Type
   Boolean Constants and Variables
   Boolean Operators
   Complex Boolean Expressions
   Truth Table
   Boolean Functions
Variables:

# A variable is the name given to a memory location. (the memory location may hold data items such as numbers, characters, etc.)

# VB programs manipulate their data by using variables.

# Syntax:
   Dim variable_name AS data_type [ = value]

# Example:

Declaring Variables:
   Dim x As Integer = 0
   Dim y As Double
   Dim z As char

# Using the Variables:
   x = 5
   y = 4.8
   z = ‘A’
Assignment Statement

# Syntax:

variable = expression

# = is the assignment operator

# An expression is a combination of variables, constants, numbers and operators.

# Examples:

N1 = N2  ' variable = variable
N1 = 5   ' variable = constant number
N1 = N2 * 5 +1  ' variable = expression
N1 = N1 + 5
Identifiers

# A name used in a VB program.
# Any string made up of letters, digits or the underscore.

    x, X1, ABC_123, ....

# An identifier can not start with a digit.

    123_A           ' illegal identifier

# A VB identifier can be any length.

# VB is NOT Case Sensitive.

    RATE       Rate       rate

are the same identifier.

Reserved Words

# Special class of identifiers.

# **Reserved by VB compiler** and have predefined meaning.
   (May not be redefined by the user)

# Example:

    integer, while, if, double, ...
Data Types

# In VB each variable must have a data type, or hold a particular category of data. VB provides a number of built-in data types. These include:

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>4 bytes</td>
<td>integer</td>
</tr>
<tr>
<td>Double</td>
<td>8 bytes</td>
<td>floating point or real number</td>
</tr>
<tr>
<td>Decimal</td>
<td>12 bytes</td>
<td>Similar to float but more exact</td>
</tr>
<tr>
<td>Long</td>
<td>8 bytes</td>
<td>Long integer (64 bit integer)</td>
</tr>
<tr>
<td>Char</td>
<td>2 bytes</td>
<td>unicode character</td>
</tr>
<tr>
<td>Byte</td>
<td>1 byte</td>
<td>0 to 255</td>
</tr>
<tr>
<td>Boolean</td>
<td>1 byte</td>
<td>True or False</td>
</tr>
<tr>
<td>Date</td>
<td>8 bytes</td>
<td>holds time and date</td>
</tr>
<tr>
<td>String</td>
<td>variable</td>
<td>holds variable sized strings</td>
</tr>
</tbody>
</table>

# In most programming languages each variable must be declared before it is used. Below are some examples of variable declaration in VB.

```vbnet
Dim X As Integer
Dim Y As Double
Dim C As Char
```
Declaring a variable will accomplish the following:

```
Dim X As Integer  
Dim Y As Double   
Dim C As Char
```

1) Associate the variable name with some memory location in which the data will be stored.

2) Tell the compiler how much memory is needed for each variable:

```
x ===> 4 bytes  
y ===> 8 bytes  
c ===> 2 bytes
```

3) Allow the compiler to perform type compatibility checking. For example the compiler can easily identify problems like:

```
c = 5.8;  
```

`' compiler error  (c is a character not a floating point number)

```
y = "hello"  
```

`' compiler error  (Y is double not a string)
Type Checking:

# Traditionally the BASIC language was not a strongly typed language. However, in VB .Net the programmer can explicitly tell the compiler to enforce strong type checking.

# Strong type checking can be enforced with the help of the following two statements.

```
Option Strict On   'Tell the compiler to enforce strict type checking.

Option Explicit On  'Tell the compiler that all variables must be declared before their use.
```
Type compatibilities:

# Two numbers 10 and 10.0 are not the same and therefore, have different types:

    10 is an integer number.

    10.0 is a floating point number.

# The compiler will automatically convert an integer number to its equivalent float number however, the reverse is not true.

# Example:

    Dim x as double
    x = 10 ' Will automatically convert to a float

    Dim y as integer
    y = 10.5 ' Error or warning in most compilers
Arithmetic Operators:

- # + Addition
- # - Subtraction
- # * Multiplication
- # / Real division
- # \ Integer division
- # MOD Remainder of an integer division (modula)

**Example:**

- 5.0 / 2.0 ==> 2.5
- 5 / 2 ==> 2
- 4 / 2 ==> 2
- 5 MOD 2 ==> 1
Module Module1

    Sub Main()
        Dim x, y, z As Integer

        x = 5
        y = 2
        z = x / y
        Console.WriteLine(z) ' output = 2
        Console.WriteLine(x / y) ' output = 2.5

        Console.WriteLine(x Mod y) ' output = 1
        Console.WriteLine(y Mod x) ' output = 2

        Dim a, b, c As Double

        a = 5.0
        b = 2.0
        c = a / b
        Console.WriteLine(c) ' output = 2.5
        Console.WriteLine(a / b) ' output = 2.5

        Console.ReadLine()
    End Sub

End Module
What happens if the Option EXPLICIT and STRICT are turned ON?

Option Strict On  'Tell the compiler to enforce strict type checking.
Option Explicit On  'Tell the compiler that all variables must be
declared before their use.

Module Module1

    Sub Main()
    Dim x, y, z As Integer

            x = 5
            y = 2

            z = x / y  'The compiler will cause an error here!!

            Console.WriteLine(z)    ' output = 2
            Console.WriteLine(x / y) ' output = 2.5

            Console.WriteLine(x Mod y)  ' output = 1
            Console.WriteLine(y Mod x)  ' output = 2

            Console.ReadLine()

    End Sub

End Module

The compiler will cause an error to indicate that the operation x/y will
result in a floating point number, and since OPTION STRICT is ON, the result
can not be implicitly converted to an integer and placed into ‘Z’.

You may solve this problem by explicitly converting the result of the
division into an integer by using a type conversion function (CInt):

        z = CInt(x / y)

CInt() will convert the result of (x / y) into an integer, before placing it into
variable “z”.

(More on type conversion functions later..)
Spacing:

# In VB, spacing before and after arithmetic operators is optional.

# Spaces are used to improve readability. (However, the VB editor reformats the program according to its own standard.)

Precedence

# Operator precedence in most programming languages is the same as operator precedence in algebra.

* , / , \ , MOD          higher precedence
+ , -                   ::::
< , <= , = , <> , > , >= lower precedence

Parentheses  (have the highest precedence)

# Parentheses can be used to determine the order of operations.

x + y * z              ===>              x + ( y * z)
(x + y) * z

# Using parentheses in equations improves readability and reduce the potential for errors.
Char type:

# Char - is the type used to represent a single letter or symbol.

Dim ch AS char
ch = 'X'
ch = '5'
ch = '?'

Integer type:

# integer - is the data-type used to represent an integer number. In VB integers are 4 bytes long.

Other compilers may have integers that are 2, 4, or even 8 bytes long.

Largest number = (2**32) -1
Smallest number = -(2**32)

NOTE: Long int in VB is 8 bytes long.

Dim value AS Integer
value = 339

Double type:

# Double - is the type used to represent a floating point numbers. In VB floating point numbers are 8 bytes long.

Dim value AS Double
value = 339.55
Constants

# Similar to a variable, a constant can store a value, however unlike a variable, the value of a constant cannot be changed during the execution of the program.

# Syntax:

CONST Constant_Name As data_type = value

# Example:

CONST PI As Double = 3.14159

# Proper use of constants will make your programs easier to read and easier to modify.
Displaying Output in a Console Application:

# Output can be sent to the display by using the Write() or WriteLine() procedure.

# Syntax:
System.Console.WriteLine(argument)

Arguments can be:

constant (string or numeric)
or
variable

# Example 1:

```vbnet
Option Strict On
Option Explicit On

Module Module1
    Sub Main()
        '-----------------------
        ' Writing to the Console
        System.Console.WriteLine(3)
        System.Console.WriteLine(3 * 5)
        System.Console.WriteLine("HELLO")
    End Sub
End Module
```

3
15
HELLO
# Example 2:

```vbnet
Option Strict On
Option Explicit On

Module Module1
    Sub Main()
        '-----------------------
        ' Writing to the Console
        Dim temp As Integer = 75
        Console.WriteLine("The temperature is {0} degrees", temp)

        Dim A, B As Integer
        A = 10
        B = 20
        Console.WriteLine("A is = {0} and B is = {1}", A, B)
    End Sub
End Module
```

The temperature is 75 degrees
A is = 10 and B is = 20
**Input** from the Keyboard (Console Applications)

# Input can be read from the keyboard by using the **ReadLine()** function.

# Syntax: 

```
Variable = System.Console.ReadLine()
```

# ReadLine() will return a STRING as the result. So, if a value other than a string is desired the programmer must perform the conversion.

```vbnet
Option Strict On
Option Explicit On

Module Module1

Sub Main()

    '--------------------------
    ' Reading from the Keyboard
    Dim Name As String
    Console.Write("What is your name? ")
    Name = Console.ReadLine()
    Console.WriteLine("Good day {0}.", Name)

    Dim Age As Double
    Console.Write("How old are you? ")
    Age = CDbl(Console.ReadLine())
    Console.WriteLine("{0} you are approximately {1} days old.",
                      Name, Age * 365)

    Console.ReadLine()

End Sub

End Module
```
# Syntax:

```vbnet
if Condition then
    Statement
end if

if Condition then
    The_true_part
else
    The_false_part
end if
```

# The Condition is expressed as follows:

\[
<\text{expression}> <\text{relational operator} > <\text{expression}>
\]

\[
X = Y
\]

# The Relational Operators are:

- `=` equal
- `<>` not equal
- `<`
- `>`
- `<=`
- `>=`
# Examples:

If score >= 90 then
    Console.WriteLine("Excellent work!")
End if

If age >= 21 then
    Console.WriteLine("drink beer..")
else
    Console.WriteLine("drink root beer..")
End if

# What’s wrong with this example?

If age <> 21 then
    Console.WriteLine("drink root beer..")
else
    Console.WriteLine("drink beer..")
End if
Module Module1

    Sub Main()
        Dim X, Y As Integer
        X = 5
        Y = 10

        If X = 5 Then
            Console.WriteLine(X) 'Output = 5
        End If
        '---------------------

        Dim score As Double = 91
        If score >= 90 Then
            Console.WriteLine("Excellent work!") ' Output Excellent Work
        End If
        '---------------------

        Dim age As Integer = 21
        If age >= 21 Then
            Console.WriteLine("drink beer..") ' Drink Beer
        Else
            Console.WriteLine("drink root beer..")
        End If
        '---------------------

        'What is wrong with this code? (hint: try setting the age to 22
        'and run the program.)

        Dim age As Integer = 22
        If age <> 21 Then
            Console.WriteLine("drink root beer..")
        Else
            Console.WriteLine("drink beer..") ' Drink Beer
        End If

        Console.ReadLine()

    End Sub
End Module
Nested IF Statement

# Used to handle complex conditions.

# Example:

```vbnet
Module Module1

Sub Main()

    '---------------------
    ' Nested If
    Dim temp As Integer = 32

    If temp <= 32 Then
        Console.WriteLine("Warning! Ice on road")
        If temp = 32 Then
            Console.WriteLine("Temp exactly freezing, beware of water.")
        Else
            Console.WriteLine("Watch for black ice! Temp: {0}", temp)
        End If   'Temp = 32
    End If   'Temp <= 32

    Console.ReadLine()

End Sub

End Module
```
ElseIf Statement

# Used to handle related sequence of Ifs without nesting.

# Example:

```vbnet
'---------------------
' ElseIf

Module Module1

Sub Main()

    Dim score As Integer
    Console.Write("Please enter the test score: ")
    score = CInt(Console.ReadLine())   'Input the score

    If score >= 90 Then
        Console.WriteLine("A")
    ElseIf score >= 80 Then
        Console.WriteLine("B")
    ElseIf score >= 70 Then
        Console.WriteLine("C")
    ElseIf score >= 60 Then
        Console.WriteLine("D")
    Else
        Console.WriteLine("Failing Grade")
    End If

    Console.ReadLine()

End Sub

End Module
```
Select Case Statement

# An alternative to a series of ElseIf statements.
# A long sequence of ElseIf statements is hard to read, and debug.
# An alternative is to replace the if-then-elseif statements with a “Select Case” statement.

# Syntax:

```
Select Case testExpression
    [ Case expressionList
        [ statements] ]
    [ Case Else
        [ else-statements] ]
End Select
```

# Example:

See next page.
Example 1:

` ' Select Case Statement
Option Strict On
Option Explicit On

Module Module1

Sub Main()
    Dim value As Integer

    Console.WriteLine("CASE Example 1:")
    Console.Write("Please enter an integer:")
    value = CInt(Console.ReadLine())

    Select Case value
        Case 1
            Console.WriteLine("The value is 1")
        Case 2, 3
            Console.WriteLine("The value {0} is 2 or 3", value)
        Case 4, 5, 6
            Console.WriteLine("The value {0} is 4, 5 or 6", value)
        Case Else ' Default case
            Console.WriteLine("The value {0} is something other than 1,2,3,4,5 or 6", value)
    End Select

    Console.ReadLine()
End Sub

End Module`
Example 2:

`Select Case Statement`  
Option Strict On  
Option Explicit On

Module Module1

    Sub Main()
        Dim value As Integer
        '--------------------------------------
        ' Testing against a range of numeric values

        Console.WriteLine("CASE  Example 2:")
        Console.Write("Please enter an integer:")
        value = CInt(Console.ReadLine())

        Select Case value
            Case Is < 10
                Console.WriteLine("The value \{0\} is less than 10", value)
            Case 10 To 14
                Console.WriteLine("The value \{0\} is between 10 to 14", value)
            Case 15
                Console.WriteLine("The value \{0\} is 15", value)
            Case Else  ' Default case
                Console.WriteLine("The value \{0\} is not found", value)
        End Select

        Console.ReadLine()
    End Sub

End Module
Example 3:

' Select Case Statement
Option Strict On
Option Explicit On

Module Module1

Sub Main()
'--------------------------------------
' Testing against a range of alphanumeric values
    Dim Name As String
    Console.WriteLine("CASE  Example 3:")
    Console.Write("Please enter a name:")
    Name = Console.ReadLine()

    Select Case Name
    Case "Adam" To "John"
        Console.WriteLine("The name {0} is between Adam and John", Name)
    Case "Kim" To "Mary"
        Console.WriteLine("The name {0} is between Kim and Mary", Name)
    Case "Paul", "Quincy", "Zima"
        Console.WriteLine("The name {0} is in the list Paul, Quincy, Zima", Name)
    Case Else ' Default case
        Console.WriteLine("The name {0} is not found", Name)
    End Select

    Console.ReadLine()
End Sub

End Module
Loops
(do loops)

A loop is a portion of a program that repeats the execution of one or more statements.

Do Loops:
# The semantics of the DO loop is:

‘Do this work while a condition is true’
Or
‘Do this work until a condition becomes true’

# Syntax:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>do while condition</td>
<td>‘condition is check at the beginning</td>
</tr>
<tr>
<td>statements</td>
<td></td>
</tr>
<tr>
<td>Loop</td>
<td></td>
</tr>
<tr>
<td>do until condition</td>
<td>‘condition is check at the beginning</td>
</tr>
<tr>
<td>statements</td>
<td></td>
</tr>
<tr>
<td>Loop</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>statements</td>
<td></td>
</tr>
<tr>
<td>Loop while condition</td>
<td>‘condition is check at the end</td>
</tr>
<tr>
<td>do</td>
<td></td>
</tr>
<tr>
<td>statements</td>
<td></td>
</tr>
<tr>
<td>Loop until condition</td>
<td>‘condition is check at the end</td>
</tr>
</tbody>
</table>


# The **Condition** is expressed as follows:

\[ \langle \text{expression} \rangle \ \langle \text{operator} \rangle \ \langle \text{expression} \rangle \]

\[ X \quad = \quad Y \]

# The **Operators** are:

\[ =, \neq, <, >, \leq, \geq \]

**And some other ones:** AND, OR, NOT

**Necessary Components of a do while loop:**

1) The loop condition must be true at one point! (therefore, the loop variable must be initialized.)

2) There must be a way to exit a loop
Examples:

' Do Loops
Option Strict On

Module Module1

Sub Main()

    Dim counter As Integer = 0
    '---------------------------------
    ' DO WHILE
    Console.WriteLine("Let's try counting up")

    Do While (counter < 10)
        Console.WriteLine("Counter = {0}", counter)
        counter = counter + 1
    Loop

    '---------------------------------
    ' DO UNTIL
    Console.WriteLine("Now lets try counting down. Note: At this point
    counter should be 10")

    Do Until counter = 0
        Console.WriteLine("Counter = {0}", counter)
        counter = counter - 1
    Loop
    Console.ReadLine()

End Sub

End Module
Examples:  (explain the result of the following loops)

```vbnet
Dim X As Integer

    X = 0
    Do While (X > 0)    'a loop that never executes
        Console.WriteLine("hello")
    Loop

'+++++++++++++++++

X = 0
Do While (X < 10)    'an infinite loop (never stops)
        Console.WriteLine("hello")
    Loop

'+++++++++++++++++
```

#  Note:

In a do-while or do-until loop:

#  if the condition is checked at the **beginning**, the body of the loop is executed **zero or more times**.
Placing the loop condition at the end:

# Syntax:

```
do
   one or more statements
Loop while (condition-is-true)

do
   one or more statements
Loop until (condition-is-true)
```

Examples:

```
counter = 0
Do
   Console.WriteLine("This line will execute at least once. ")
Loop While counter > 0
```

```
counter = 5  ' How many hello’s does this loop print?
Do
   Console.WriteLine("This line will execute at least once. ")
Loop Until (counter >= 0)  '0, 1, 5 or 6 time? Desk check it!
```

# Note:

In a do-while or do-until loop:

# if the condition is checked at the end, the body of the loop is executed one or more times.
Methods for terminating a loop:

1) **Count controlled loop** (by knowing how many times the loop should execute.)

2) **Ask before iteration.** (after or before every iteration ask the user if they want to continue)

3) Exit on a **flag condition or terminating value**. (e.g. if the number entered is negative, or the name entered is "dummy", etc.)

4) **Running out of values. (EOF)** Typically used when reading from files.

Questions to ask yourself when writing VB programs?

# When should you use a **Do-While** loop?
# When should you use a **Do-Until** loop?

# When should you use a **Do-loop-While** loop?
# When should you use a **Do-loop-Until** loop?
(For Loops)

For Loop:

# Syntax:

\[
\text{For Loop}\_\text{variable} = \text{starting\_point} \text{ to } \text{ending\_point} \ [ \text{step} ] \\
\text{statement} \\
\text{Next} \ [\text{Loop\_variable}] \\
\]

# Examples:

```vbnet
Module Module1

Sub Main()
    Dim LoopCounter As Integer
    For LoopCounter = 0 To 10 Step 1
        Console.WriteLine("hello: {0}", LoopCounter)
    Next LoopCounter

    Console.WriteLine()
    '------------------------

    For LoopCounter = 10 To 0 Step -1
        Console.WriteLine("Goodbye: {0}", LoopCounter)
    Next LoopCounter

    Console.WriteLine()
    '------------------------

    Console.ReadLine()
    End Sub

End Module
```
'FOR LOOPS

Module Module1
    Sub Main()
        Dim Row, Col As Integer
        For Row = 1 To 10
            For Col = 1 To 10
                Console.Write("{0,4}", Row * Col) ' {0,4} Right
                'Justify the value in 4 spaces.
            Next Col
            Console.WriteLine()
        Next Row
        Console.WriteLine()
        '------------------------
        For Row = 10 To 1 Step -1
            For Col = 10 To 1 Step -1
                Console.Write("{0,-6}", Row * Col) ' {0,-6} Left
                'Justify the value in 6 spaces.
            Next Col
            Console.WriteLine()
        Next Row
        Console.ReadLine()
    End Sub
End Module
Boolean Expressions, Variables, Operators

Boolean Expressions

# An expression which evaluates to either TRUE or FALSE.

(x > y)
(x = y)
(x <= 48)

# A boolean expression has the following syntax:

<operand> relational-operator <operand>

# Operands can be variables or constants:

<variable> relational-operator <variable>
          or
<variable> relational-operator <constant>

# Relational Operators are:

=, <, >, <=, >=, <>

# Constants may be:

literal constants  (i.e. 1, 5.2, “hello”, 'x')
          or
constant identifiers (i.e. TAX_RATE, PI)
Boolean Type

Variables of type boolean can only hold two values.

(true or false)

```vbnet
Dim DONE As Boolean
DONE = False
Console.WriteLine(DONE) ' outputs "false"

DONE = True
Console.WriteLine(DONE) ' outputs "true"
```

BOOLEAN variables, constants and functions can make your programs more readable.
Boolean Constants and Variables

# Constants of Type BOOLEAN.

```vbnet
const DEBUG_MODE As Boolean = true
```

# Variables of type BOOLEAN.

```vbnet
Dim DONE As Boolean
```

# Use of Boolean Constants in the body of the program:

```vbnet
Module Module1

Sub Main()

    Const DEBUG_MODE As Boolean = True

    ' program statements...
    If DEBUG_MODE = True Then
        Console.WriteLine("This line will only print if the program is in DEBUG MODE")
    End If

    Console.ReadLine()

End Sub

End Module
```
Use of Boolean Variables in the body of the program:

```vbnet
Module Module1
    Sub Main()

        Dim DONE As Boolean
        DONE = False
        Console.WriteLine(DONE) ' outputs "false"

        '-----------------------
        Dim value As Integer

        Do Until DONE
            Console.WriteLine(value)
            value = value + 1
            If value = 5 Then DONE = True
        Loop
        Console.WriteLine(DONE) ' outputs "true"

        '-----------------------
        DONE = False
        value = 0
        Do While Not DONE
            Console.WriteLine(value)
            value = value + 1
            If value = 5 Then DONE = True
        Loop
        Console.WriteLine(DONE) ' outputs "true"

        Console.ReadLine()
    End Sub
End Module
```

Dim DONE As Boolean
DONE = False
Do While (NOT DONE)
    // one or more statements
    if (some condition arises)
        DONE = True // stop the loop
    Loop
Boolean Operators

# Boolean operators are used to form more complicated boolean expressions.

# Syntax:

<boolean expr> boolean operator <boolean expr>

# There are three boolean operators:

<table>
<thead>
<tr>
<th>AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>OR</td>
</tr>
<tr>
<td>NOT</td>
</tr>
</tbody>
</table>

Complex Boolean Expressions:

# Is formed by combining simple boolean expressions with the AND, OR, NOT operators.

# Examples:

(Salary < MinSal) OR (NumDependents > 5)
(hours_worked > 40) AND (hours_worked <= 60)
(hours_worked > 40) AND (NOT SALARY_EMP)
Truth Table:

<table>
<thead>
<tr>
<th>AND</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>
Evaluate the following Boolean Expressions:

Assume:

\[
\begin{align*}
\text{Dim } & X, Y, Z \text{ As Double} \\
\text{Dim FLAG As Boolean} \\
X & = 3.0 \\
Y & = 4.0 \\
Z & = 2.0 \\
Flag & = \text{false}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>((X &gt; Z) \text{ AND } (Y &gt; Z))</td>
<td></td>
</tr>
<tr>
<td>((X + Y / Z) \leq 3.5)</td>
<td></td>
</tr>
<tr>
<td>((Z &gt; X) \text{ OR } (Z &gt; Y))</td>
<td></td>
</tr>
<tr>
<td>(\text{NOT Flag})</td>
<td></td>
</tr>
<tr>
<td>((X = 1.0) \text{ OR } (X = 3.0))</td>
<td></td>
</tr>
<tr>
<td>((Z &lt; X) \text{ AND } (X &lt; Y))</td>
<td></td>
</tr>
<tr>
<td>((X \leq Z) \text{ OR } (X \geq Y))</td>
<td></td>
</tr>
<tr>
<td>((\text{NOT Flag}) \text{ OR } ((Y + Z) \geq (X - Z)))</td>
<td></td>
</tr>
<tr>
<td>(\text{NOT (Flag OR ((Y + Z) \geq (X - Z)))})</td>
<td></td>
</tr>
</tbody>
</table>
Module Module1

Sub Main()

'-----------------------
'Boolean Expressions

Dim X, Y, Z As Double
Dim FLAG As Boolean
X = 3.0
Y = 4.0
Z = 2.0
FLAG = False

Console.WriteLine("(X > Z) AND (Y > Z)  = {0}", ((X > Z) And (Y > Z)))
Console.WriteLine("(X + Y / Z) <= 3.5  = {0}", ((X + Y / Z) <= 3.5))
Console.WriteLine("(Z > X) OR (Z > Y)  = {0}", ((Z > X) Or (Z > Y)))
Console.WriteLine("NOT FLAG            = {0}", (Not FLAG))
Console.WriteLine("(X = 1.0) OR (X = 3.0) = {0}", ((X = 1.0) Or (X = 3.0)))
Console.WriteLine("(Z < X) AND (X < Y)    = {0}", ((Z < X) And (X < Y)))
Console.WriteLine("(Z <= X) OR ( X >= Y)  = {0}", ((Z <= X) Or ( X >= Y)))
Console.WriteLine("(NOT FLAG) OR ((Y + Z) >= (X - Z)) = {0}",
                      ((Not FLAG) Or ((Y + Z) >= (X - Z))))
Console.WriteLine("NOT (FLAG OR ((Y + Z) >= (X - Z))) = {0}",
                      (Not (FLAG Or ((Y + Z) >= (X - Z)))))

Console.ReadLine()

End Sub

End Module

Output:

(X > Z) AND (Y > Z)  = True
(X + Y / Z) <= 3.5  = False
(Z > X) OR (Z > Y)  = False
NOT FLAG            = True
(X = 1.0) OR (X = 3.0) = True
(Z < X) AND (X < Y)    = True
(X <= Z) OR ( X >= Y)  = False
(NOT FLAG) OR ((Y + Z) >= (X - Z)) = True
NOT (FLAG OR ((Y + Z) >= (X - Z))) = False