Selection (Decision Structures)

Many times in programming, you want to make a decision based on the values of the variables and select different courses of action based on that.

If Statements

The first type of decision structure allows the selection of either one or two alternatives based on a condition.

The one alternative version is the If-Then statement. If the comparison is true, then the statements in [some action] are executed. If the comparison is false, no actions are executed.

```
IF (number or variable) (comparison) (number or  variable) THEN
        [some action]
END IF
```

Example:

```
If total > 50.0 Then
        DiscountRate = .10
End If
```

The two alternative version is the If-Then-Else statement. If the comparison is true, then the statements in [some action] are executed. If the comparison is false, then [some different actions] are executed.

```
IF (number or variable) (comparison) (number or  variable) THEN
        [some actions]
ELSE
        [some different actions]
END IF
```

Example:

```
If total > 50.0 Then
        DiscountRate = .10
Else
        DiscountRate = .05
End If
```

Comparison Operators

	Operator	Example	Meaning
Equal to	=	EmpCode = “S”	Is employee code equal to “S”?
Greater than	>	Hours > 40	Are the hours greater than 40?
Greater than	>=	Revenue >= Costs	Is Revenue greater than or equal to

or equal to			costs?
Less than	<	thisyear < lastyear	Is this year’s revenue less than last year’s?
Less than or equal to	<=	Total <= 1000.0	Is the total value less than or equal to 1000 (dollars)?
Not equal to	<>	Number <> 0	Is the number not equal to zero?

Select Case Statement

This statement works for the situation where you have a single variable and you want to make decisions based on the values of that variable. The Select Case statement will allow you to program alternatives for each value of the variable or for a range of values.

In the Select Case statement only one alternative is ever executed.

The first form of the select statement has alternative actions based on individual values of the **case statement variable**:

```
Select Case <variable of any type>
        Case <value1> : [some action 1]
        Case <value2> : [some action 2]
        …
        Case <valueN> : [some action N]
        ‘ the else case is optional
        Case else:  [some more action ]
End Select
```

If the value of the variable is value1, then the first some action is executed and all the rest are skipped. If the value of the variable is value2, then the second some action is executed and all the rest are skipped. If there is an else case and the value of the variable is not listed in any of the other cases, then this action is executed.

Example:

```
Select Case grade
      Case "A" : message = "Great!"
      Case "B" : message = "Good!"
      Case "C" : message = "Okay, you made it"
      Case "D" : message = "Not so good"
      Case "F" : message = "Disaster!"
      Case Else : message = "  "
  End Select
```

The second form of the select statement has alternative actions based on ranges of values of the variable:

```
Select Case <variable of any type>
        Case <value1> to <endvalue1> : [some action 1]
        Case <value2> to <endvalue2> : [some action 2]
        …
        Case <valueN> to <endvalueN> : [some action N]
        ‘ the else case is optional
        Case else:  [some more action ]
End Select
```

In this form, each alternative action is executed if the value of the variable falls within the range of the two values given in the case.

Example:
```
      Select Case gradeaverage
            Case 90 To 100 : grade = "A"
            Case 80 To 89 : grade = "B"
            Case 60 To 79 : grade = "C"
            Case 40 To 59 : grade = "D"
            Case Else : grade = "F"
        End Select
```

Note that Select Case statements can also have a mix of value and value ranges in a single statement.

Nested If Statements

In the actions of an If statement, there can be more If statements. These can be used to test more conditions.

Example: Suppose that there is a String variable called PayStatus that is either “Hourly” or “Salaried”, and that we want to compute the Pay of an employee using a variable Hours that tells how many hours they worked that week, and the variable PayRate that tells how much per hour that they make. Salaried employees are always paid as if they worked 40 hours, no matter how many hours they actually worked. If hourly employees work more than 40 hours in a week, then they get 1.5 times their pay rate for any hours over 40.

```
If  PayStatus = “Hourly”  Then
        If Hours > 40 Then   ‘ pay regular plus overtime
                Pay = (PayRate * 40) + (1.5 * PayRate * (Hours – 40))
        Else   ‘ regular pay
                Pay = PayRate * Hours
        End If
Else   ‘ for salaried workers
        Pay = PayRate * 40
End If
```

Data Type Boolean

So far, we have used types Integer, Single, and String. Now we introduce another data type called Boolean.

You can declare variables of type Boolean.

```
        Dim isValid as Boolean
```

There are only 2 values of type Boolean: True and False. These can be assigned to variables of type Boolean.

```
isValid = False
isValid = True
```

Boolean values can be used in conditions of If statements

```
If isValid Then
    Message = “OK”
Else
    Message = “Try again.”
End If
```

More About Conditions

If you have an If statement and you want to test if two conditions are both true, you can use an And operator:

```
If ((PayStatus = “Hourly”) And (Hours > 40)) Then
    <actions>
End If
```

If you have an If statement and you want to test if at least one of two conditions is true, you can use an Or operator:

```
If (( StudentYear = “Junior”) Or (StudentYear = “Senior”)) Then
    Message = “Invite to Prom”
Else
    Message = “No Prom”
End If
```