CONTROL STRUCTURES

This are statements that are used to control the flow of program's execution. They include: Sequence, Decision and Looping

Decision Structures

Visual Basic procedures can test conditions and then, depending on the results of that test, perform different operations. The decision structures that Visual Basic supports include:

i) If statements
ii) Select Case statements

If Statements

If statement implements branching logic based on a testable condition. The following is the general format of the **Block If** statement:

```
If <condition> Then
    (condition is true - do all of the actions
     coded in this branch)
Else
    (condition is false - do all of the actions
     coded in this branch)
End If
```

Conditions and Condition Symbols

You must learn how to write conditions by using the relational operators shown in this table.

Relational Operator	Meaning
>	greater than
<	less than
=	equal to
<>	not equal to
>=	greater than or equal to
<=	less than or equal to

Comparisons must be made comparing like data types to like data types, for example: Compare string to string (text), compare decimal to decimal, compare single to single.

If Statement (Single Structures)

The simplest **If** statement has only a **True** branch with no **Else** (**False**) branch – no action is taken if the condition is false.

```
If MedicalCheckBox.Checked Then
    'Checked the medical insurance checkbox
    BenefitsCostDecimal += MEDICAL_RATE_DECIMAL
End If
```

If Statement with Else Branch (Double Structures)

The more common **If** statement has both **True** and **False** branches as shown in this example.

```
If HoursDecimal <= 40D Then
    'Pay only regular time
    GrossPayDecimal = Decimal.Round(HoursDecimal * PayRateDecimal, 2)
Else
    'Pay regular time + overtime
    GrossPayDecimal = Decimal.Round((40D * PayRateDecimal) + _
   ((HoursDecimal - 40D) * PayRateDecimal * 1.5D), 2)
End If
```

If … Then … ElseIf Statements (multiple selections):-

If...Then…ElseIf is really just a special case of If...Then...Else. Evaluation of the If … Then … ElseIf is the construct that uses several conditions with the *ElseIf* keyword so as to make one selection from multiple selections. Notice that you can have any number of ElseIf clauses

Syntax

```
If condition1 Then
[statementblock-1]
ElseIf condition2 Then
[statementblock-2]]
...
Else
[statementblock-n]]
End If
```

Logical Operators

Logical operators are used to combine conditions. These are termed **compound conditions**. The logical operators are as follows:

- **Or** operator – If one condition or both conditions are **True**, the entire compound condition evaluates to **True**
- **And** operator – Both conditions being combined must be **True** in order for the compound condition to be **True**.
- **Not** operator – Negates a condition – if a condition is **True**, it is treated as **False**, and vice-versa
- **AndAlso** operator – Termed a short-circuit of the **And** operator. If the first condition is **False**, the compound condition is treated as **False** and the second condition is not evaluated. In this example, if the value in the **HoursTextBox** TextBox is **not numeric**, then there is no need to test the second condition – in fact, if the value is not numeric, testing the value to see if it falls within a numeric range would lead to an **exception error** due to trying to compare non-numeric data with numeric values.

```
If IsNumeric(HoursTextBox.Text) = True AndAlso
Decimal.Parse(HoursTextBox.Text, Globalization.NumberStyles.Number) <= 60D
Then
    'The data is valid, use this branch of the
    'decision structure to process the data
Else
     'Data is not valid, hours must be numeric and within allowable range
    MessageBox.Show("Hours worked must be a number between 0 and 60.",
    "Hours Numeric Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
    HoursTextBox.Focus()
    HoursTextBox.SelectAll()
End If
```

- **OrElse** operator – Termed a short-circuit of the **Or** operator. If the first condition is **True**, the compound condition is treated as **True** and the second condition is not evaluated. This is sort of a mirror-image of the **AndAlso** operator – sometimes you will use one, sometimes the other depending upon how you choose to structure your logic.

Example

In the following example, the data in the **HoursTextBox** TextBox control is first evaluated to see if it is numeric – if it is not then the condition evaluates to **True** and the second condition testing if the hours worked is less than **0D** or greater than **60D** is not tested.

```
If IsNumeric(HoursTextBox.Text) = False OrElse
Decimal.Parse(HoursTextBox.Text, Globalization.NumberStyles.Number) > 60D
Then
    'Data is not valid - hours must be numeric and within allowable range
    MessageBox.Show("Hours worked must be a number between 0 and 60.", Hours
Numeric Error",  MessageBoxButtons.OK, MessageBoxIcon.Error)
    HoursTextBox.Focus()
    HoursTextBox.SelectAll()
Else
    'The data is valid, use this branch of the
    'decision structure to process the data
End If
```

- **Xor (Exclusive Or)** operator
  If one condition is **True**, the other must be **False** in order for the compound condition to be treated as **True** overall and both conditions cannot be **True**. Useful when situations are mutually exclusive.

Input Data Validation

An application user is likely to make occasional data entry errors. A form's data must be validated against a list of **business rules** developed by a systems analyst. Example business rules include:

i) **Missing Data Test** – data values cannot be missing – this means a TextBox control cannot be empty.
ii) **Numeric Data Test** – numeric values must be tested to ensure they are numeric.
iii) **Reasonableness Test** – values (usually numeric) must fall within a specified reasonable range.

i) Missing Data Test – Employee Name

A **Missing Data** validation test is also called an **Existence Test** or **Required Field Test** – you must test a control such as a TextBox to ensure that data that must be is not missing. E.g. The **NameTextBox** control requires a name in order to process payroll.

The following code segment illustrates the coding logic for the click event of the **ComputeButton** with data validation code added to validate the **NameTextBox.**

```
Private Sub ComputeButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ComputeButton.Click
    Try
        'Declare variables and constants
        'Enforce data validation rules
        If NameTextBox.Text.Trim = String.Empty Then
            'Required employee name is missing
            MessageBox.Show("Name is required", "Name Missing Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error)
            NameTextBox.Focus()
            NameTextBox.SelectAll()
        Else
            'All data is valid—process the data
            'by use of the Input-Process-Output model
            MessageBox.Show("Data is valid.", "No Error",
            MessageBoxButtons.OK, MessageBoxIcon.Information)
        End If  'matches If statement for validating data
    Catch ex As Exception
        MessageBox.Show("Unexpected error: " & ControlChars.NewLine &
        ex.Message, "Compute Button Error", MessageBoxButtons.OK,
        MessageBoxIcon.Error)
    End Try
End Sub
```

The **Trim** method trims (deletes) leading and trailing blank characters – if the textbox only contains spaces (blank characters) the Trim method results in the empty string.

ii) Missing Data Test – Employee ID

The **EmployeeID** is a required value that is formatted as **SSN**. This is an additional example of missing data validation that uses an **OrElse** operator.

```
ElseIf EmployeeIDMaskedTextBox.Text.Trim.Length <> 11 OrElse _
    EmployeeIDMaskedTextBox.Text.IndexOf(" ", 0, _
        EmployeeIDMaskedTextBox.Text.Length) <> -1 Then
    'Required employee ID is not long enough or is not complete
    MessageBox.Show("Employee ID is not complete", _
        "Employee ID Error", MessageBoxButtons.OK, _
        MessageBoxIcon.Error)
    EmployeeIDMaskedTextBox.Focus()
    EmployeeIDMaskedTextBox.SelectAll()
```

The first half of the condition uses the **Length** method to evaluate the string length and finds if the string is 11 characters in length. The **IndexOf** method is used to search the string for the existence of any blank space within a specified portion of a string.

iii) Numeric Data Test and Reasonableness Test – Hours Worked and Pay Rate

A TextBox value that must be containing a valid numeric value is tested with the **IsNumeric** function. Hours worked must be numeric and also fall within a specific valid range – **zero to 60** hours.

```
ElseIf IsNumeric(HoursTextBox.Text) = False OrElse
(Decimal.Parse(HoursTextBox.Text, Globalization.NumberStyles.Number) <= 0D Or
Decimal.Parse(HoursTextBox.Text, Globalization.NumberStyles.Number) > 60D) Then
        'Hours must be numeric and within allowable range
        MessageBox.Show("Hours worked must be a numeric between 0 and 60",
        "Hours Value Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        HoursTextBox.Focus()
        HoursTextBox.SelectAll()
```

An alternative to using the OrElse operator is to break the compound condition into two separate, ElseIf statements.

```
ElseIf IsNumeric(HoursTextBox.Text) = False Then
    'Hours must be numeric
        MessageBox.Show("Hours worked must be a number", "Hours Not Numeric
        Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        HoursTextBox.Focus()
        HoursTextBox.SelectAll()
ElseIf Decimal.Parse(HoursTextBox.Text, Globalization.NumberStyles.Number) <= 0D
Or Decimal.Parse(HoursTextBox.Text, Globalization.NumberStyles.Number) > 60D Then
    'Hours must be within allowable range
        MessageBox.Show("Hours worked must be between 0 and 60", "Hours Value
        Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
        HoursTextBox.Focus()
        HoursTextBox.SelectAll()
```

Review the Overall Logical Structure of the Click Event:

- The **Try-Catch** block is used to catch general exceptions.
  - The **Try** branch is used to validate and process the data.
  - The **Catch** branch catches unexpected general exceptions.
- The large **If-ElseIf-Else-End If** statement is used to validate and process the data.

- The **If** and **ElseIf** branches are used to test if the business rule is **INVALID**.
- If a business rule is violated, display a message box, set the focus to the control with invalid data, select any data within that control, then transfer overall program control to the last **End If** statement.

- The **Else** branch is used to store all of the code used for **Input-Process-Output-Other Tasks**.
- A correct solution will almost always have **ABSOLUTELY NO CODE** between the last **End If** and the **Catch** statements. Also there is no code after the End Try statement except for the End Sub.

Select Case Structure
A **Select Case** block statement in Visual Basic is a form of decision structure. It can be used in place of an **If** statement when a **single variable** value, **single expression** value, **single control property**, or similar object is to be evaluated and different actions are taken depending on the value of the expression.
The advantage of a **Select Case** instead of multiple **If** statements is that a **Select Case** is often easier to read and code than multiple **If** statements.

Syntax:

```
Select Case testexpression
      Case Value 1
            Statements block 1
      Case Value 2
            Statements block 2
      Case Value n
            Statements block n
              .
              .
      Case Else
            Statements block n+1
  End Select
```

The **Case Else** branch is optional and is used when the various Case statements are not completely exhaustive of all possible values for the object being evaluated.

There are several rules to learn about coding a Select Case:
You must use the keyword **Is** when using a comparison operator such as =, >=, or <=; examples:

i). The following is a comparison between **Select Case** and the equivalent **If** statement:

```
If ValueDecimal < 150 Then
    'Do one thing
ElseIf ValueDecimal >= 2580 Then
    'Do a second thing
Else
    'Do some alternative thing
End If

Select Case ValueDecimal
    Case Is < 150
        'Do one thing
    Case Is >= 2580
        'Do a second thing
    Case Else
        'Do some alternative thing
End Select
```

ii). To test for a value that falls within a range of constants, the keyword **To** is used; example:

```
Case 25 To 72
```

Compare the **Select Case** to the equivalent **If** statement code:

```
If ValueDecimal >= 25 And ValueDecimal <= 72 Then
    'Do one thing
ElseIf ValueDecimal >= 90 And ValueDecimal <= 100 Then
    'Do a second thing
Else
    'Do some alternative thing
End If

Select Case ValueDecimal
    Case 25 To 72
        'Do one thing
    Case 90 To 100
        'Do a second thing
    Case Else
        'Do some alternative thing
End Select
```

iii).A list can combine individual values and ranges of values; example:

```
Case 15, 17, 25 To 72, 79, Is > 150
```

Compare the **Select Case** to the equivalent **If** statement code:

```
If ValueDecimal = 15 Or ValueDecimal = 17 Or _
    (ValueDecimal >= 25 And ValueDecimal <= 72) Or _
    ValueDecimal  = 79 Or ValueDecimal  > 150 Then _
    'Do something
Else
    'Do alternative
End If

Select Case ValueDecimal
    Case 15, 17, 25 To 72, 79, Is > 150
        'Do something
    Case Else
        'Do alternative
End Select
```

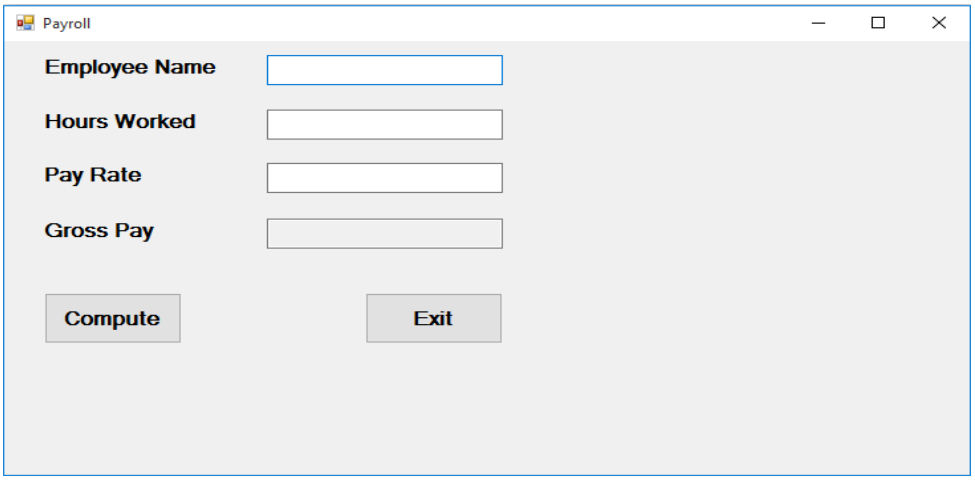
Sample Application:

A program will be required to determine the gross pay for each of several employees of a certain company. The company pays "straight-time" for the first 40 hours worked by each employee and pays "one-and –a half time" for all hours worked in excess of 40 hours. You are given a list of employees of the company, the number of hours each employee worked and the hourly rate of each employee. Your program should input this information for each employee, and then compute and display the employee's gross pay. Use textboxes for both inputs and outputs.

Required:

i) Design your graphical user interface

ii) Write the program code to compute gross pay. Your program should validate the inputs in such a way that the names of the employee must be input, the number of hours worked and pay rate must be numeric, greater than zero but less than or equal to 60 and greater than zero respectively. Display an appropriate message in case of violation.



Properties

- The first three TextBox controls are used for data entry – use these names: **EmployeeNameTextBox, HoursWorkedTextBox** and **PayRateTextBox**
- The last TextBox control is used to display the output. Set these properties:
    - **ReadOnly** property = **True**,
    - Name propertie = **GrossPayTextBox**.
- Name the buttons **ComputeButton** and **ExitButton**.

Program Code

```
Public Class PayrollForm
    Private Sub ComputeButton_Click(sender As Object, e As EventArgs) Handles
    ComputeButton.Click
        Try
            Dim HoursDecimal, PayRateDecimal, GrossPayDecimal As Decimal
            If EmployeeNameTextBox.Text.Trim = String.Empty Then
                MessageBox.Show("Name is required", "Name Missing Error",
                MessageBoxButtons.OK, MessageBoxIcon.Error)
                EmployeeNameTextBox.Focus()
                EmployeeNameTextBox.SelectAll()
            ElseIf IsNumeric(HoursWorkedTextBox.Text) = False OrElse
            (Decimal.Parse(HoursWorkedTextBox.Text,
             Globalization.NumberStyles.Number) <= 0D Or
            Decimal.Parse(HoursWorkedTextBox.Text,
            Globalization.NumberStyles.Number) > 60D)  Then
                'Hours must be numeric and within allowable range
                MessageBox.Show("Hours worked must be numeric between 0 and 60",
                "Hours Value Error", MessageBoxButtons.OK,
                 MessageBoxIcon.Error)
                HoursWorkedTextBox.Focus()
                HoursWorkedTextBox.SelectAll()
            ElseIf IsNumeric(PayRateTextBox.Text) = False OrElse
            Decimal.Parse(PayRateTextBox.Text,
            Globalization.NumberStyles.Currency) <= 0D Then
                'Pay rate must be numeric and greater than zero
                MessageBox.Show("Pay rate worked must be greater than zero.",
                        "Pay Rate Value Error", MessageBoxButtons.OK,
                        MessageBoxIcon.Error)
                PayRateTextBox.Focus()
                PayRateTextBox.SelectAll()
```

```
            Else
                HoursDecimal = Decimal.Parse(HoursWorkedTextBox.Text,
                Globalization.NumberStyles.Number)
                PayRateDecimal = Decimal.Parse(PayRateTextBox.Text,
                Globalization.NumberStyles.Currency)
                'Compute gross pay
                If HoursDecimal <= 40D Then 'pay only regular time
                    GrossPayDecimal = Decimal.Round(HoursDecimal *
                    PayRateDecimal, 2)
                Else 'pay regular + overtime
                    GrossPayDecimal = Decimal.Round((40D * PayRateDecimal) _
                    + ((HoursDecimal - 40D) * PayRateDecimal * 1.5D), 2)
                End If
                GrossPayTextBox.Text = GrossPayDecimal.ToString("C")
            End If
        Catch ex As Exception
           MessageBox.Show("Unexpected error: " & ControlChars.NewLine &
           ex.Message, "Compute Button Error", MessageBoxButtons.OK,
           MessageBoxIcon.Error)
        End Try
    End Sub

    Private Sub ExitButton_Click(sender As Object, e As EventArgs) Handles
    ExitButton.Click
        Me.Close()
    End Sub
End Class
```

Exercise

Write a Visual Basic program that will be used in a school to calculate the average mark out of four subjects. The average mark is used to determine the grade the student gets. The grade is displayed by use of a message box dialog. The grading system is shown in the table below. Sketch your GUI

MARK	GRADE
0-49	Fail
50-64	C
65-74	B
75-100	A