

Visual Studio interface showing a C# project named `frmUser.vb` in Design mode. The code in `btnClick_Click` handles the button click event, prompting the user for hours worked and pay rate, calculating gross pay, and displaying it in `txtGrossPay`.

```
3 Private Sub btnClick_Click(sender As System.Object, e As System.EventArgs) Handles btnClick.Click
4     Dim wkHours As Decimal
5     Dim wkPayPerHour As Decimal
6     Dim wkGrossPay As Decimal
7     wkHours = InputBox("Enter hours worked", "Hours")
8     wkPayPerHour = InputBox("Enter pay per hour", "Hourly Pay Rate")
9     wkGrossPay = wkPayPerHour * wkHours
10    txtGrossPay.text = FormatCurrency(wkGrossPay)
11
12 End Sub
13 End Class
14
15
```

Setting up and using variables and taking in user input via a popup prompt.

The image also shows a screenshot of the application running, displaying a "USER INPUT" dialog box and a "Hourly Pay Rate" input box with the value 50.

Visual Studio interface showing the Autos, Call Stack, and other debugging tools.

Visual Studio interface showing a VB.NET project named `frmUser.vb` in Design view. The code in `btnClick` calculates gross pay based on hours worked and pay rate.

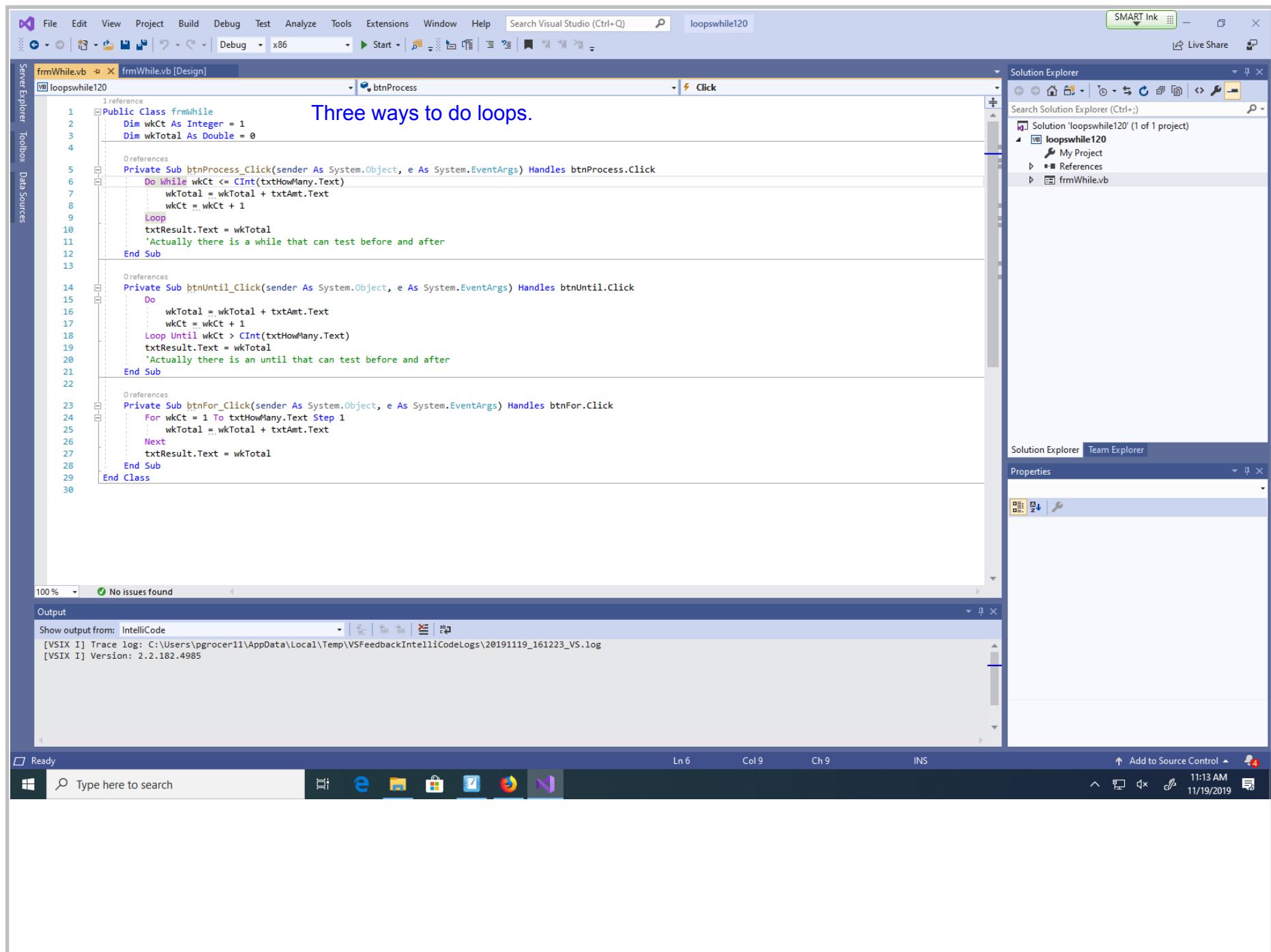
```
Private Sub btnClick_Click(sender As System.Object, e As System.EventArgs) Handles btnClick.Click
    Dim wkHours As Decimal
    Dim wkPayPerHour As Decimal
    Dim wkGrossPay As Decimal
    wkHours = InputBox("Enter hours worked", "Hours")
    wkPayPerHour = InputBox("Enter pay per hour", "Hourly Pay Rate")
    wkGrossPay = wkPayPerHour * wkHours
    txtGrossPay.text = FormatCurrency(wkGrossPay)
End Sub
End Class
```

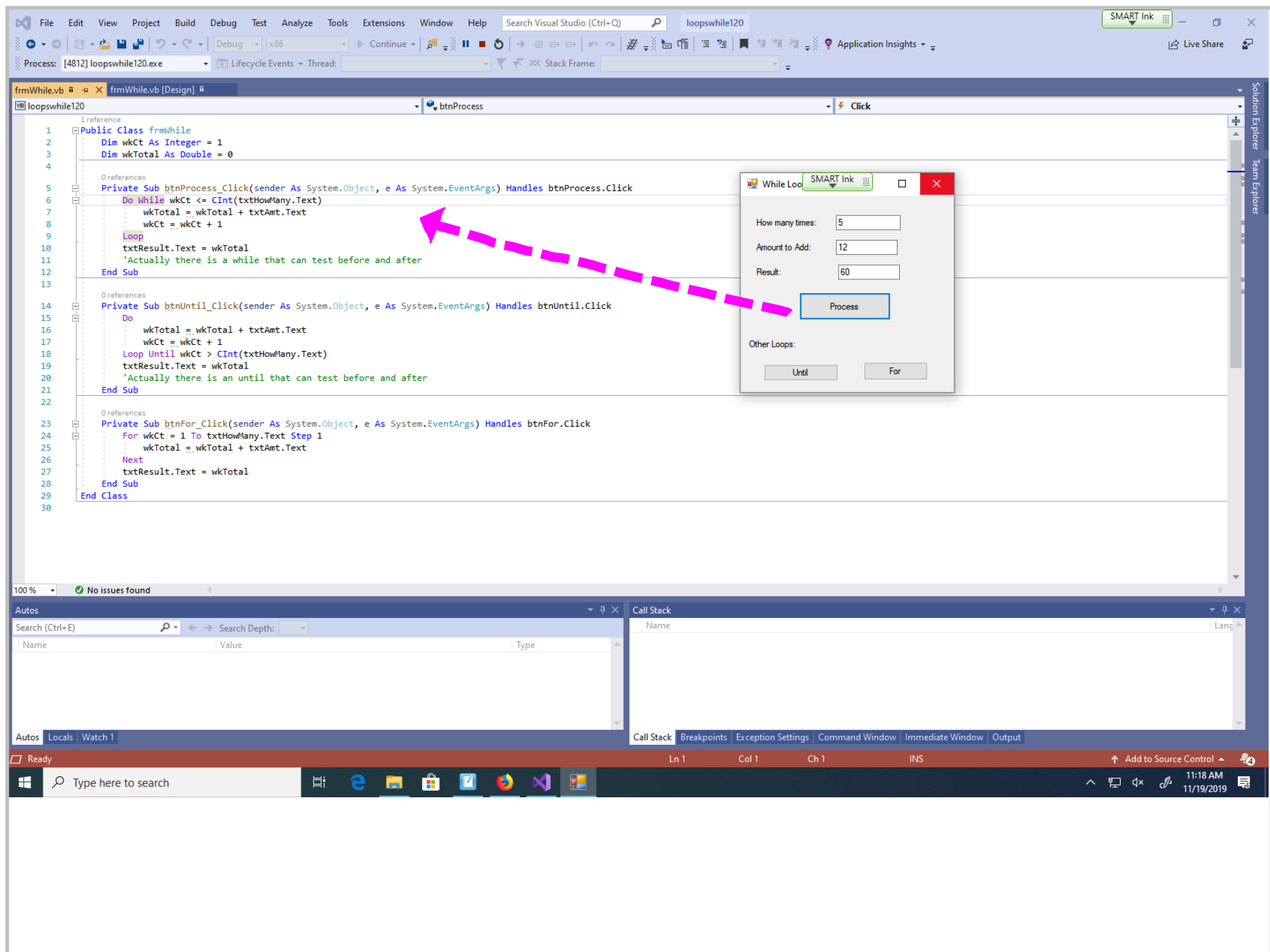
A modal dialog box titled **USER INPUT** is displayed, showing the results of the calculation:

Name:

Gross Pay:

The interface also shows the Autos, Locals, and Watch windows, and the Call Stack, Breakpoints, Exception Settings, Command Window, Immediate Window, and Output windows.





Visual Studio interface showing a VB.NET project named 'loopswhile120'. The code in 'frmWhile.vb' defines a class 'frmWhile' with three methods: 'btnProcess_Click', 'btnUntil_Click', and 'btnFor_Click'. A pink dashed arrow points from the 'Until' button in the 'While Loop' dialog to the 'Loop Until' statement in the 'btnUntil_Click' method.

```
1 reference
2 Public Class frmWhile
3     Dim wkCt As Integer = 1
4     Dim wkTotal As Double = 0
5
6     Private Sub btnProcess_Click(sender As System.Object, e As System.EventArgs) Handles btnProcess.Click
7         Do While wkCt <= CInt(txtHowMany.Text)
8             wkTotal = wkTotal + txtAmt.Text
9             wkCt = wkCt + 1
10        Loop
11        txtResult.Text = wkTotal
12        'Actually there is a while that can test before and after
13    End Sub
14
15    Private Sub btnUntil_Click(sender As System.Object, e As System.EventArgs) Handles btnUntil.Click
16        Do
17            wkTotal = wkTotal + txtAmt.Text
18            wkCt = wkCt + 1
19        Loop Until wkCt > CInt(txtHowMany.Text)
20        txtResult.Text = wkTotal
21        'Actually there is an until that can test before and after
22    End Sub
23
24    Private Sub btnFor_Click(sender As System.Object, e As System.EventArgs) Handles btnFor.Click
25        For wkCt = 1 To txtHowMany.Text Step 1
26            wkTotal = wkTotal + txtAmt.Text
27        Next
28        txtResult.Text = wkTotal
29    End Sub
30 End Class
```

The 'While Loop' dialog box shows the following settings:

- How many times: 5
- Amount to Add: 112
- Result: 560
- Process button
- Other Loops: Until (selected), For

The bottom status bar shows 'Ready' and 'No issues found'.

Visual Studio interface showing a VB.NET project named 'loopswhile120'. The code defines a class 'frmWhile' with three methods: 'btnProcess_Click', 'btnUntil_Click', and 'btnFor_Click'. The 'btnProcess_Click' method uses a 'Do While' loop to calculate a total based on user input. The 'btnUntil_Click' method uses a 'Loop Until' loop. The 'btnFor_Click' method uses a 'For' loop. A handwritten note in blue ink says 'while keeps me in' and 'Until what gets me out me'.

The 'While Loop' dialog box is open, showing the following values:

- How many times: 5
- Amount to Add: 112
- Result: 560
- Process button
- Other Loops: Until, For

The Autos window shows the following variables:

Name	Value	Type
wkCt	5	Integer
wkTotal	560	Double
txtHowMany	5	String
txtAmt	112	String
txtResult	560	String

The Call Stack window shows the following call stack:

Name	Language
frmWhile.btnProcess_Click	VB
frmWhile.btnProcess_Click	VB
frmWhile.btnProcess_Click	VB

The bottom status bar shows the current line is 6, column is 9, and the cursor is in Insert mode. The system tray shows the date and time as 11:20 AM on 11/19/2019.

The while tests to see if the processing should stay doing the loop. Deals with what keeps processing in the loop. The until tests to see when the loop is done and you should exit the loop.

Visual Studio interface showing a VB.NET project named 'loopswhile120'. The code defines a class 'frmWhile' with three methods: 'btnProcess_Click', 'btnUntil_Click', and 'btnFor_Click'. The 'btnProcess_Click' method uses a 'Do While' loop. The 'btnUntil_Click' method uses a 'Do' loop with an 'Until' condition. The 'btnFor_Click' method uses a 'For' loop. A dialog box titled 'While Loop' is open, showing input fields for 'How many times' (0), 'Amount to Add' (12), and 'Result' (12). A pink dashed arrow points from the 'Until' button in the dialog to the 'Until' loop in the code. A blue text box explains that entering 0 for the number of times will cause the 'until' loop to be executed once, while the 'while' loop would not be entered.

```
1 Public Class frmWhile
2     Dim wkCt As Integer = 1
3     Dim wkTotal As Double = 0
4
5     Private Sub btnProcess_Click(sender As System.Object, e As System.EventArgs) Handles btnProcess.Click
6         Do While wkCt <= CInt(txtHowMany.Text)
7             wkTotal = wkTotal + txtAmt.Text
8             wkCt = wkCt + 1
9         Loop
10        txtResult.Text = wkTotal
11        'Actually there is a while that can test before and after
12    End Sub
13
14    Private Sub btnUntil_Click(sender As System.Object, e As System.EventArgs) Handles btnUntil.Click
15        Do
16            wkTotal = wkTotal + txtAmt.Text
17            wkCt = wkCt + 1
18        Loop Until wkCt > CInt(txtHowMany.Text)
19        txtResult.Text = wkTotal
20        'Actually there is an until that can test before and after
21    End Sub
22
23    Private Sub btnFor_Click(sender As System.Object, e As System.EventArgs) Handles btnFor.Click
24        For wkCt = 1 To txtHowMany.Text Step 1
25            wkTotal = wkTotal + txtAmt.Text
26        Next
27        txtResult.Text = wkTotal
28    End Sub
29 End Class
30
```

While Loop

How many times: 0

Amount to Add: 12

Result: 12

Process

Other Loops:

Until For

Entering 0 for number of times will cause processing to be executed once when I use the until loop. With the while loop Result would be 0 because the loop would not be entered.

Visual Studio interface showing a VB.NET project named 'loopswwhile120'. The main window displays the code for 'frmWhile.vb [Design]'. The code defines a public class 'frmWhile' with two integer variables 'wkCt' and 'wkTotal'. It includes three event handlers: 'btnProcess_Click' (a 'Do While' loop), 'btnUntil_Click' (a 'Loop Until' loop), and 'btnFor_Click' (a 'For' loop). A pink dashed arrow points from the 'Do While' loop in the code to a 'While Loop' dialog box. The dialog box has fields for 'How many times' (0), 'Amount to Add' (12), and 'Result' (0), with a 'Process' button and 'Until'/'For' radio buttons.

```
1 reference
2 Public Class frmWhile
3     Dim wkCt As Integer = 1
4     Dim wkTotal As Double = 0
5
6     Private Sub btnProcess_Click(sender As System.Object, e As System.EventArgs) Handles btnProcess.Click
7         Do While wkCt <= CInt(txtHowMany.Text)
8             wkTotal = wkTotal + txtAmt.Text
9             wkCt = wkCt + 1
10        Loop
11        txtResult.Text = wkTotal
12        'Actually there is a while that can test before and after
13    End Sub
14
15    Private Sub btnUntil_Click(sender As System.Object, e As System.EventArgs) Handles btnUntil.Click
16        Do
17            wkTotal = wkTotal + txtAmt.Text
18            wkCt = wkCt + 1
19        Loop Until wkCt > CInt(txtHowMany.Text)
20        txtResult.Text = wkTotal
21        'Actually there is an until that can test before and after
22    End Sub
23
24    Private Sub btnFor_Click(sender As System.Object, e As System.EventArgs) Handles btnFor.Click
25        For wkCt = 1 To txtHowMany.Text Step 1
26            wkTotal = wkTotal + txtAmt.Text
27        Next
28        txtResult.Text = wkTotal
29    End Sub
30 End Class
```

While Loop SMART Ink

How many times: 0

Amount to Add: 12

Result: 0

Process

Other Loops:

Until For

Autos

Search (Ctrl+E)

Name Value Type

Call Stack

Name Language

Autos Locals Watch 1

Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Ready

Type here to search

Ln 6 Col 9 Ch 9 INS

Add to Source Control

11:22 AM 11/19/2019

Visual Studio interface showing a VB.NET project named 'loopswhile120'. The code in 'frmWhile.vb' defines a class 'frmWhile' with three methods: 'btnProcess_Click', 'btnUntil_Click', and 'btnFor_Click'. The 'btnProcess_Click' method uses a 'Do While' loop. The 'btnUntil_Click' method uses a 'Loop Until' loop. The 'btnFor_Click' method uses a 'For' loop. A red handwritten note 'Start end condition increments' is written across the code. A 'While Loop' dialog box is open, showing 'How many times: 5', 'Amount to Add: 12', and 'Result:'. The 'Process' button is highlighted. The 'Autos' window is empty. The 'Call Stack' window is empty. The status bar shows 'Ln 6 Col 9 Ch 9 INS'.

```
1 reference
2 Public Class frmWhile
3     Dim wkCt As Integer = 1
4     Dim wkTotal As Double = 0
5
6     Private Sub btnProcess_Click(sender As System.Object, e As System.EventArgs) Handles btnProcess.Click
7         Do While wkCt <= CInt(txtHowMany.Text)
8             wkTotal = wkTotal + txtAmt.Text
9             wkCt = wkCt + 1
10        Loop
11        txtResult.Text = wkTotal
12        'Actually there is a while that can test before and after
13    End Sub
14
15    Private Sub btnUntil_Click(sender As System.Object, e As System.EventArgs) Handles btnUntil.Click
16        Do
17            wkTotal = wkTotal + txtAmt.Text
18            wkCt = wkCt + 1
19        Loop Until wkCt > CInt(txtHowMany.Text)
20        txtResult.Text = wkTotal
21        'Actually there is an until that can test before and after
22    End Sub
23
24    Private Sub btnFor_Click(sender As System.Object, e As System.EventArgs) Handles btnFor.Click
25        For wkCt = 1 To txtHowMany.Text Step 1
26            wkTotal = wkTotal + txtAmt.Text
27        Next
28        txtResult.Text = wkTotal
29    End Sub
30 End Class
```

While Loop

How many times: 5

Amount to Add: 12

Result:

Process

Other Loops:

Until For

Autos

Search (Ctrl+E)

Name Value Type

Call Stack

Name Lang

Ready

Ln 6 Col 9 Ch 9 INS

Add to Source Control

Type here to search

11:24 AM 11/19/2019

Visual Studio interface showing a VB.NET project named 'loopswhile120'. The code defines a class 'frmWhile' with three methods: 'btnProcess_Click', 'btnUntil_Click', and 'btnFor_Click'. The 'btnProcess_Click' method uses a 'Do While' loop. The 'btnUntil_Click' method uses a 'Do Loop Until' loop. The 'btnFor_Click' method uses a 'For' loop. A red handwritten note 'Start + end condition increments' is written across the code. A 'While Loop' dialog box is open, showing input values: 'How many times: 5', 'Amount to Add: 12', and 'Result: 60'. The dialog also has buttons for 'Process', 'Until', and 'For'.

```
1 reference
2 Public Class frmWhile
3     Dim wkCt As Integer = 1
4     Dim wkTotal As Double = 0
5
6     Private Sub btnProcess_Click(sender As System.Object, e As System.EventArgs) Handles btnProcess.Click
7         Do While wkCt <= CInt(txtHowMany.Text)
8             wkTotal = wkTotal + txtAmt.Text
9             wkCt = wkCt + 1
10        Loop
11        txtResult.Text = wkTotal
12        'Actually there is a while that can test before and after
13    End Sub
14
15    Private Sub btnUntil_Click(sender As System.Object, e As System.EventArgs) Handles btnUntil.Click
16        Do
17            wkTotal = wkTotal + txtAmt.Text
18            wkCt = wkCt + 1
19        Loop Until wkCt > CInt(txtHowMany.Text)
20        txtResult.Text = wkTotal
21        'Actually there is an until that can test before and after
22    End Sub
23
24    Private Sub btnFor_Click(sender As System.Object, e As System.EventArgs) Handles btnFor.Click
25        For wkCt = 1 To txtHowMany.Text Step 1
26            wkTotal = wkTotal + txtAmt.Text
27        Next
28        txtResult.Text = wkTotal
29    End Sub
30 End Class
```

While Loop dialog box:

- How many times: 5
- Amount to Add: 12
- Result: 60
- Process
- Other Loops: Until, For

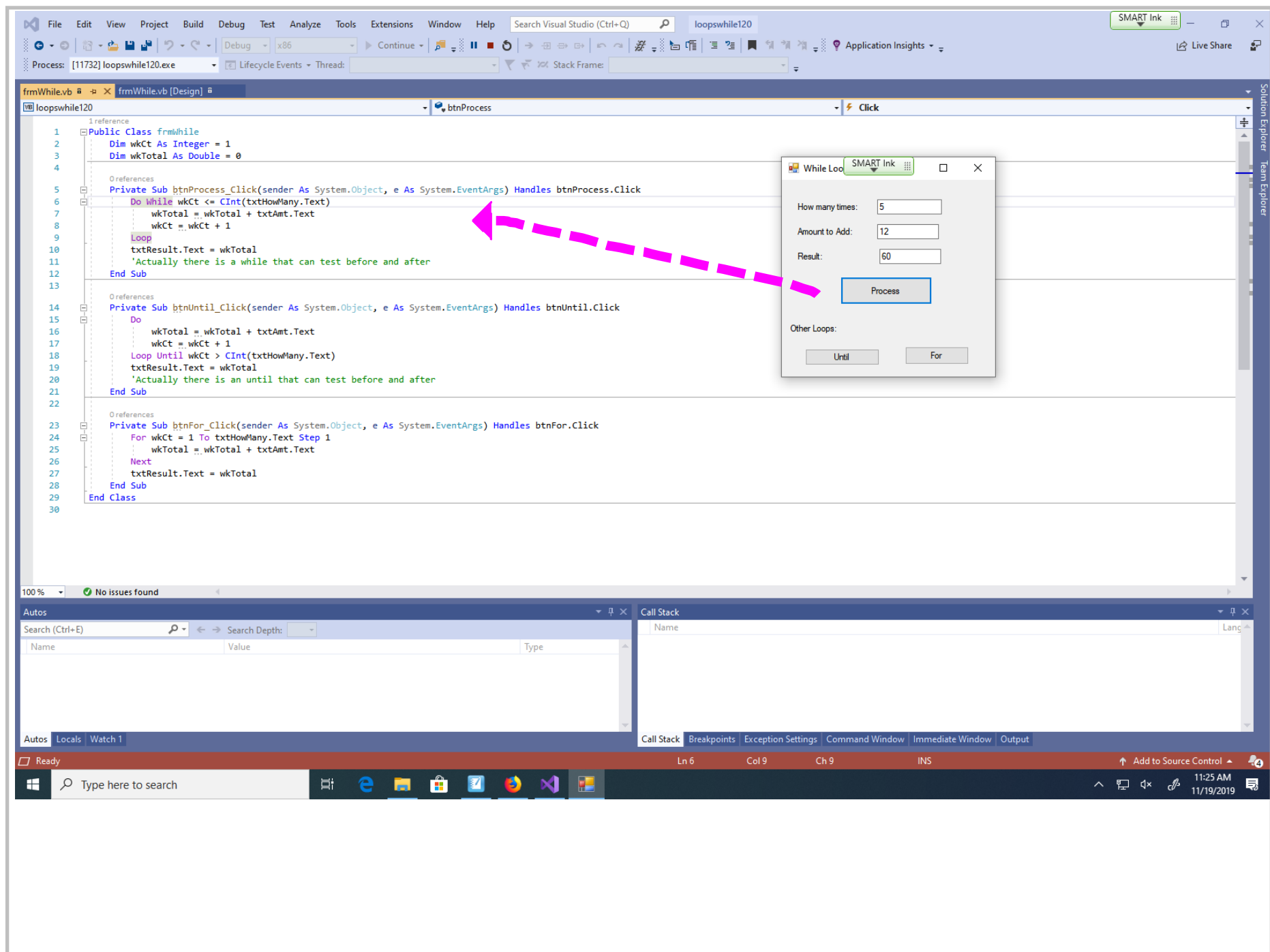
Autos, Locals, Watch 1

Call Stack, Breakpoints, Exception Settings, Command Window, Immediate Window, Output

Ln 6 Col 9 Ch 9 INS

Ready Type here to search

11:24 AM 11/19/2019



Visual Studio interface showing a VB.NET project named 'loopswwhile120'. The code in 'frmWhile.vb' defines a class 'frmWhile' with three event handlers: 'btnProcess_Click', 'btnUntil_Click', and 'btnFor_Click'. The 'btnProcess_Click' handler uses a 'Do While' loop to calculate a total based on user input. A 'While Loop' dialog box is open, showing 'How many times: 5', 'Amount to Add: 12', and 'Result: 72'. The 'Process' button is highlighted. The 'btnUntil_Click' handler uses a 'Do Until' loop. The 'btnFor_Click' handler uses a 'For' loop. The bottom status bar shows 'Ready' and the date/time '11:25 AM 11/19/2019'.

```
1 reference
2 Public Class frmWhile
3     Dim wkCt As Integer = 1
4     Dim wkTotal As Double = 0
5
6     Private Sub btnProcess_Click(sender As System.Object, e As System.EventArgs) Handles btnProcess.Click
7         Do While wkCt <= CInt(txtHowMany.Text)
8             wkTotal = wkTotal + txtAmt.Text
9             wkCt = wkCt + 1
10        Loop
11        txtResult.Text = wkTotal
12        'Actually there is a while that can test before and after
13    End Sub
14
15    Private Sub btnUntil_Click(sender As System.Object, e As System.EventArgs) Handles btnUntil.Click
16        Do
17            wkTotal = wkTotal + txtAmt.Text
18            wkCt = wkCt + 1
19        Loop Until wkCt > CInt(txtHowMany.Text)
20        txtResult.Text = wkTotal
21        'Actually there is an until that can test before and after
22    End Sub
23
24    Private Sub btnFor_Click(sender As System.Object, e As System.EventArgs) Handles btnFor.Click
25        For wkCt = 1 To txtHowMany.Text Step 1
26            wkTotal = wkTotal + txtAmt.Text
27        Next
28        txtResult.Text = wkTotal
29    End Sub
30 End Class
```

While Loop

How many times: 5

Amount to Add: 12

Result: 72

Process

Other Loops:

Until For

I did the while loop in btnProcess and then I clicked on Until to do the until loop. It went through it once and increased the total by 12 and the wkCt by 1. When it checked, wkCt was greater than txtHowMany so the loop ended after one pass.

Autos

Search (Ctrl+E)

Name Value Type

Autos Locals Watch 1

Call Stack

Name Lang

Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Ready

Ln 6 Col 9 Ch 9 INS

Type here to search

11:25 AM 11/19/2019

Visual Studio interface showing a VB.NET project named 'loopswWhile120'. The code in 'frmWhile.vb' defines a class 'frmWhile' with three event handlers: 'btnProcess_Click', 'btnUntil_Click', and 'btnFor_Click'. The 'btnProcess_Click' handler uses a 'Do While' loop to calculate the sum of numbers from 1 to 'txtHowMany.Text'. The 'btnUntil_Click' handler uses a 'Do Until' loop for the same calculation. The 'btnFor_Click' handler uses a 'For' loop. A 'While Loop' dialog box is open, showing 'How many times: 5', 'Amount to Add: 12', and 'Result: 84'. The 'Process' button is highlighted. The 'Autos' window is empty, and the 'Call Stack' window is also empty. The status bar at the bottom shows 'Ready' and the date/time '11:26 AM 11/19/2019'.

```
1 reference
2 Public Class frmWhile
3     Dim wkCt As Integer = 1
4     Dim wkTotal As Double = 0
5
6     Private Sub btnProcess_Click(sender As System.Object, e As System.EventArgs) Handles btnProcess.Click
7         Do While wkCt <= CInt(txtHowMany.Text)
8             wkTotal = wkTotal + txtAmt.Text
9             wkCt = wkCt + 1
10        Loop
11        txtResult.Text = wkTotal
12        'Actually there is a while that can test before and after
13    End Sub
14
15    Private Sub btnUntil_Click(sender As System.Object, e As System.EventArgs) Handles btnUntil.Click
16        Do
17            wkTotal = wkTotal + txtAmt.Text
18            wkCt = wkCt + 1
19        Loop Until wkCt > CInt(txtHowMany.Text)
20        txtResult.Text = wkTotal
21        'Actually there is an until that can test before and after
22    End Sub
23
24    Private Sub btnFor_Click(sender As System.Object, e As System.EventArgs) Handles btnFor.Click
25        For wkCt = 1 To txtHowMany.Text Step 1
26            wkTotal = wkTotal + txtAmt.Text
27        Next
28        txtResult.Text = wkTotal
29    End Sub
30 End Class
```

I did the until loop one more time.

Visual Studio IDE showing a VB.NET project named `loopswhile120`. The code defines a class `frmWhile` with three methods: `btnProcess_Click`, `btnUntil_Click`, and `btnFor_Click`. The `btnProcess_Click` method contains a `Do While` loop that increments `wkTotal` and `wkCt` until `wkCt` is greater than `CInt(txtHowMany.Text)`. The `btnUntil_Click` method contains a `Do` loop that increments `wkTotal` and `wkCt` until `wkCt` is greater than `CInt(txtHowMany.Text)`. The `btnFor_Click` method contains a `For` loop that increments `wkTotal` from 1 to `CInt(txtHowMany.Text)`.

Handwritten blue notes in the code editor indicate a fix for the `btnUntil_Click` method: `wkCt = 1` and `wkTotal = 0` should be added to the beginning of the loop.

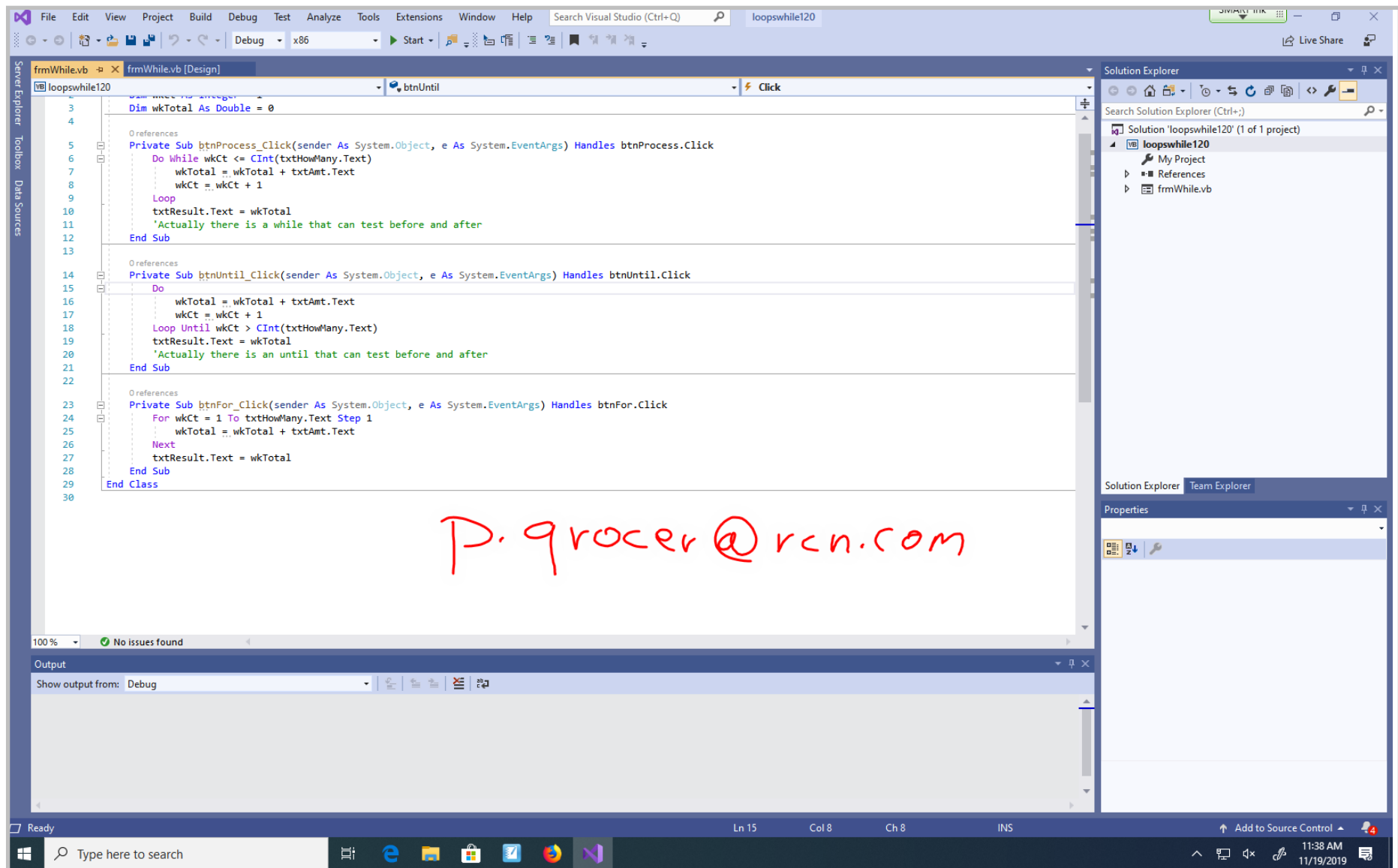
A dialog box titled "While Loop" is shown, displaying the following values:

- How many times: 5
- Amount to Add: 12
- Result: 84

The dialog box also has a "Process" button and a section for "Other Loops" with "Until" and "For" options.

The bottom of the IDE shows the "Autos" window with a search bar and a table with columns "Name", "Value", and "Type". The "Call Stack" window is also visible.

Windows taskbar at the bottom shows the time as 11:29 AM on 11/19/2019.



Visual Studio interface showing a VB.NET project named 'firstIF'. The code defines two event handlers: `btnCheck_Click` and `btnAnother_Click`. The `btnCheck_Click` handler calculates the difference between `txtBudget.Text` and `txtSpent.Text` and updates `txtResult.Text` based on whether the difference is greater than 1000. The `btnAnother_Click` handler checks if `txtDept.Text` is 'CI' and updates `txtResult.Text` based on whether `txtBudget.Text` is greater than 5000.

```
3 Private Sub btnCheck_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCheck.Click
4     Dim wkDiff As Integer
5     wkDiff = txtBudget.Text - txtSpent.Text
6     If wkDiff > 1000 Then
7         txtResult.Text = wkDiff & " is what you have to spend"
8     Else
9         txtResult.Text = "You have 1000 or less to spend"
10    End If
11 End Sub
12
13 References
14 Private Sub btnAnother_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnAnother.Click
15     If txtDept.Text = "CI" Then
16         If txtBudget.Text > 5000 Then
17             txtResult.Text = "Your budget will not change"
18         Else
19             txtResult.Text = "You can request an increase of 10%"
20         End If
21     Else
22         txtResult.Text = "Not the CI Department"
23     End If
24 End Sub
25 End Class
```

A handwritten flowchart is overlaid on the code, illustrating the logic of the `btnAnother_Click` handler:

- Decision diamond: `dept CI`. If 'N' (No), it leads to a process box `not CI`, which then leads to a process box `Increase`.
- If 'Y' (Yes), it leads to a decision diamond `budget > 5000`.
- From `budget > 5000`: If 'N' (No), it leads to a process box `Increase`. If 'Y' (Yes), it leads to a process box `same`.
- Both `Increase` and `same` process boxes lead to a final output box.

The Output window shows the program execution trace, indicating that the program exited with code 0 (0x0).

Visual Studio interface showing a C# project named `firstIF`. The code in `frmFirst.vb` implements two click events for buttons `btnCheck` and `btnAnother`.

```
Private Sub btnCheck_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCheck.Click
    Dim wkDiff As Integer
    wkDiff = txtBudget.Text - txtSpent.Text
    If wkDiff > 1000 Then
        txtResult.Text = wkDiff & " is what you have to spend"
    Else
        txtResult.Text = "You have 1000 or less to spend"
    End If
End Sub

Private Sub btnAnother_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnAnother.Click
    If txtDept.Text = "CI" Then
        If txtBudget.Text > 5000 Then
            txtResult.Text = "Your budget will not change"
        Else
            txtResult.Text = "You can request an increase of 10%"
        End If
    Else
        txtResult.Text = "Not the CI Department"
    End If
End Sub
```

A hand-drawn flowchart in red ink is overlaid on the code, illustrating the logic for the `btnAnother_Click` event:

- Decision diamond: `dept CI`
- If `Y` (Yes): Decision diamond: `budget > 5000`
- If `Y` (Yes) to budget: Terminal box: `same`
- If `N` (No) to budget: Terminal box: `increase`
- If `N` (No) to dept: Terminal box: `not CI`
- Arrows connect the flow from the `dept CI` diamond to the `budget > 5000` diamond, and from the `budget > 5000` diamond to the `increase` and `same` boxes. The `not CI` box also has an arrow pointing to the `increase` box.

A dialog box titled `IF` is open, showing input fields for `Dept:` (12), `Budget:` (6000), and `Spent:` (2000). It has buttons for `Check` and `Another Check`, and a text box at the bottom containing `Not the CI Department`.

The bottom of the screen shows the Windows taskbar with the time 11:45 AM on 11/19/2019.

Visual Studio interface showing a VB.NET project named `firstIF`. The code defines a class `frmFirst` with two event handlers: `btnCheck_Click` and `btnAnother_Click`.

```
1 Public Class frmFirst
2
3     Private Sub btnCheck_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCheck.Click
4         Dim wkDiff As Integer
5         wkDiff = txtBudget.Text - txtSpent.Text
6         If wkDiff > 1000 Then
7             txtResult.Text = wkDiff & " is what you have to spend"
8         Else
9             txtResult.Text = "You have 1000 or less to spend"
10        End If
11    End Sub
12
13    Private Sub btnAnother_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnAnother.Click
14        If txtDept.Text = "CI" Then
15            If txtBudget.Text > 5000 Then
16                txtResult.Text = "Your budget will not change"
17            Else
18                txtResult.Text = "You can request an increase of 10%"
19            End If
20        Else
21            txtResult.Text = "Not the CI Department"
22        End If
23    End Sub
24 End Class
```

Handwritten notes and a flowchart are overlaid on the code:

- Problem I asked the class to include in this program:
 CI and either $budget > 10000$ or $spent > budget$
- Flowchart logic:
 - Decision: $dept = CI$
 - If Yes (Y): Decision: $budget > 5000$
 - If Yes (Y): Action: $same$
 - If No (N): Action: $increase$
 - If No (N): Action: $not CI$
 - If Yes (Y) from $not CI$: Action: $increase$

Visual Studio IDE showing a VB.NET project named `firstIF`. The code in `frmFirst.vb` defines two event handlers: `btnCheck_Click` and `btnAnother_Click`. The `btnCheck_Click` handler calculates the difference between budget and spent amounts and updates `txtResult.Text` based on whether the difference is greater than 1000. The `btnAnother_Click` handler checks if the department is "CI" and updates the result text based on budget and spent amounts.

Handwritten notes and diagrams are present:

- Red handwritten notes:** "either budget > 10000 OR spent budget".
- Blue handwritten notes:** "IF TextDept.text = 'CI' Then If +txtBudget.text > 10000 then +txtResult.text = 'OK' else If +txtSpent.text > +txtResult.text = 'Problem' endif".
- Flowchart:** A flowchart starting with a decision diamond "CI". If "Yes", it goes to a decision diamond "budget > 10000". If "Yes", it goes to a box "OK". If "No", it goes to a decision diamond "spent > budget". If "Yes", it goes to a box "OK". If "No", it goes to a box "Not the CI Department".
- Green handwritten note:** "Since different cannot be compound".

The bottom of the image shows the Windows taskbar with the system clock at 11:59 AM on 11/19/2019.

Visual Studio IDE showing a VB.NET project named 'firstIF'. The code in 'frmFirst.vb' defines two event handlers: 'btnCheck_Click' and 'btnAnother_Click'. The 'btnCheck_Click' handler calculates the difference between budget and spent amounts and updates the 'txtResult' text based on whether the difference is greater than 1000. The 'btnAnother_Click' handler checks if the department is 'CI' and then checks the budget and spent amounts to update the 'txtResult' text.

```

Private Sub btnCheck_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnCheck.Click
    Dim wkDiff As Integer
    wkDiff = txtBudget.Text - txtSpent.Text
    If wkDiff > 1000 Then
        txtResult.Text = wkDiff & " is what you have to spend"
    Else
        txtResult.Text = "You have 1000 or less to spend"
    End If
End Sub

Private Sub btnAnother_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnAnother.Click
    If txtDept.Text = "CI" Then
        If txtBudget.Text > 5000 Then
            txtResult.Text = "Your budget will not change"
        Else
            txtResult.Text = "You can request an increase of 10%"
        End If
    Else
        txtResult.Text = "Not the CI Department"
    End If
End Sub

```

Handwritten notes and diagrams are present on the code and in the right margin:

- Red handwritten note:** "IF txtDept.text = 'CI' AND (either budget > 10000 OR spent > budget)"
- Blue handwritten note:** "IF TextDept.text = 'CI' Then If txtBudget.text > 10000 then txtResult.text = 'OK' else If txtSpent.text > txtBudget.text then txtResult.text = 'Problem' endif"
- Flowchart:** A flowchart starting with a decision diamond 'CI'. If 'Yes', it goes to a decision diamond 'budget > 10000'. If 'Yes', it goes to a box 'OK'. If 'No', it goes to a decision diamond 'spent > budget'. If 'Yes', it goes to a box 'Problem'. If 'No', it goes to a box 'OK'. If 'No' from the initial 'CI' decision, it goes to a box 'Not CI'.
- Green handwritten note:** "Convert is important in comparisons especially if different number of digits. CInt()" with a small window titled 'another Check' showing a '10%' value.
- Call Stack:** Shows the current method 'firstIF'.