Chapter 4: Control Structures: Part 1

Outline

4.1 Introduction
4.2 Algorithms
4.3 Pseudocode
4.4 Control Structures
4.5 If/Then Selection Structure
4.6 If/Then/Else Selection Structure
4.7 While Repetition Structure
4.8 Do While/Loop Repetition Structure
4.9 Do Until/Loop Repetition Structure
4.10 Assignment Operators
4.11 Formulating Algorithms: Case Study 1 (Counter-Controlled Repetition)
4.12 Formulating Algorithms with Top-Down, Stepwise Refinement: Case Study 2 (Sentinel-Controlled Repetition)

4.13 Formulating Algorithms with Top-Down, Stepwise Refinement: Case Study 3 (Nested Control Structures)

4.14 Formulating Algorithms with Top-Down, Stepwise Refinement: Case Study 4 (Nested Repetition Structures)

4.15 Introduction to Windows Application Programming

4.1 Introduction

- Structured programming
  - Control structures
    - Helpful in building and manipulating objects

4.2 Algorithms

- Algorithms
  - A procedure for solving a problem, in terms of
    - The actions to be executed and
    - The order in which these actions are to be executed

4.3 Pseudocode

- Pseudocode
  - Informal language to helps programmers develop algorithms
  - Not executed on computers
  - Helps conceptualize a program during the program-design process
  - Describes only executable statements

4.4 Control Structures

- Transfer of control
  - **GoTo** statement
    - It causes programs to become quite unstructured and hard to follow
- Bohm and Jacopini
  - All programs could be written in terms of three control structures
    - Sequence structure
    - Selection structure
    - Repetition structure

4.4 Control Structures

- Flowcharts
  - Graphical representation of an algorithm
  - Drawn using certain special-purpose symbols
    - Rectangles
    - Diamonds
    - Ovals
    - Small circles

4.4 Control Structures

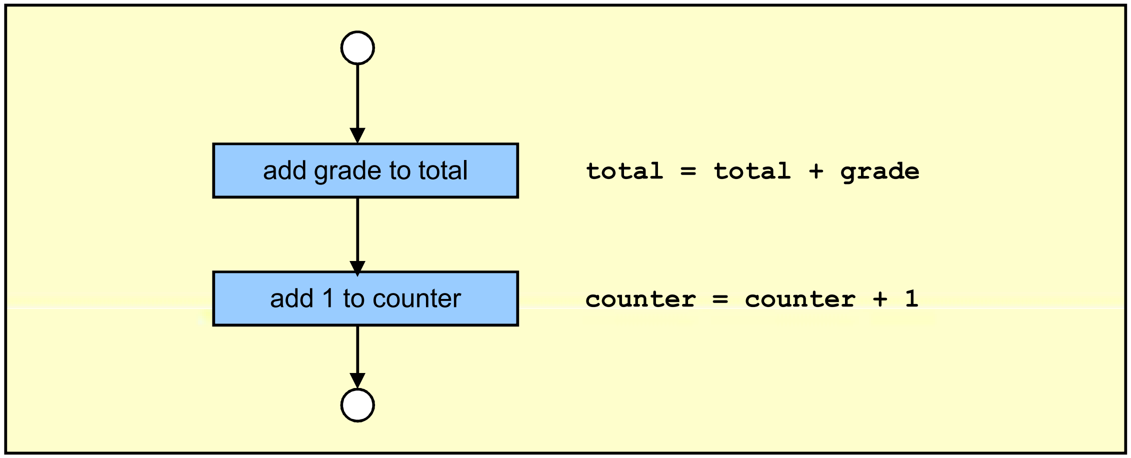


Fig. 4.1 Flowcharting Visual Basic's sequence structure.

4.4 Control Structures

- Selection Structures
  - `If/Then`
    - Single-selection structure
  - `If/Then/Else`
    - Double-selection structure
  - `Select Case`
    - Multiple-selection structure

4.4 Control Structures

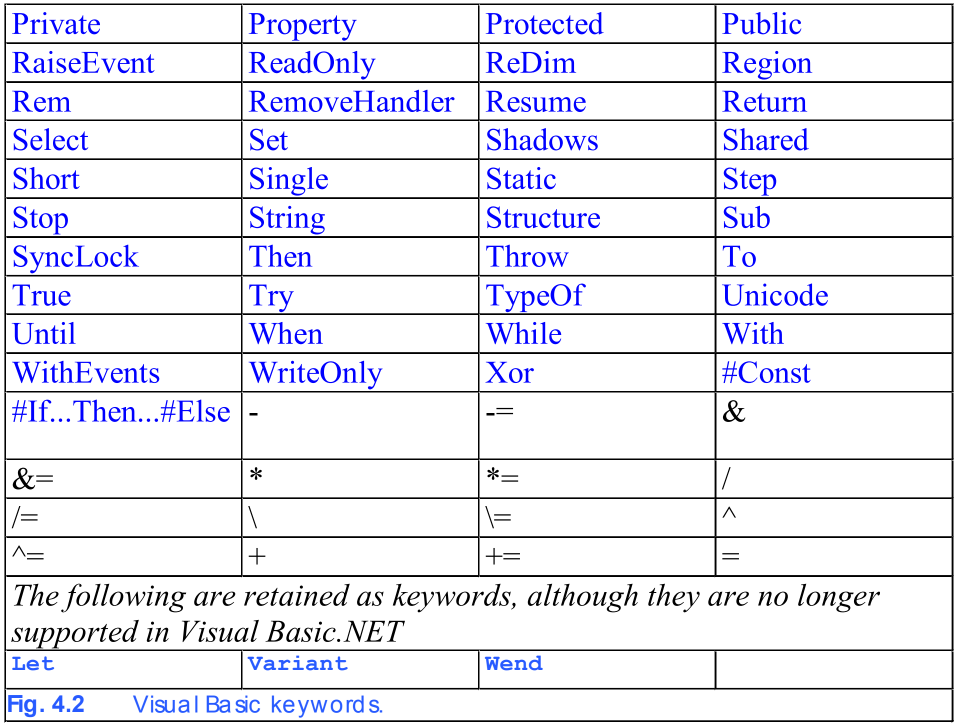
- Repetition Structures
  - **`While`**
  - **`Do While/Loop`**
  - **`Do/Loop While`**
  - **`Do Until/Loop`**
  - **`Do/Loop Until`**
  - **`For/Next`**
  - **`For Each/Next`**

4.4 Control Structures

Visual Basic Keywords			
AddHandler	AddressOf	Alias	And
AndAlso	Ansi	As	Assembly
Auto	Boolean	ByRef	Byte
ByVal	Call	Case	Catch
CBool	CByte	CChar	CDate
CDec	CDbl	Char	CInt
Class	CLng	CObj	Const
CShort	CSng	CStr	CType
Date	Decimal	Declare	Default
Delegate	Dim	Do	Double
Each	Else	ElseIf	End
Enum	Erase	Error	Event
Exit	ExternalSource	False	Finally
For	Friend	Function	Get
GetType	GoTo	Handles	If
Implements	Imports	In	Inherits
Integer	Interface	Is	Lib
Like	Long	Loop	Me
Mod	Module	MustInherit	MustOverride
MyBase	MyClass	Namespace	New
Next	Not	Nothing	NotInheritable
NotOverridable	Object	On	Option
Optional	Or	OrElse	Overloads
Overridable	Overrides	ParamArray	Preserve

Fig. 4.2 Visual Basic keywords.

4.4 Control Structures



Private	Property	Protected	Public
RaiseEvent	ReadOnly	ReDim	Region
Rem	RemoveHandler	Resume	Return
Select	Set	Shadows	Shared
Short	Single	Static	Step
Stop	String	Structure	Sub
SyncLock	Then	Throw	To
True	Try	TypeOf	Unicode
Until	When	While	With
WithEvents	WriteOnly	Xor	#Const
#If...Then...#Else	-	-=	&
&=	*	*=	/
/=	\	\=	^
^=	+	+=	=
The following are retained as keywords, although they are no longer supported in Visual Basic.NET			
Let	Variant	Wend	

Fig. 4.2 Visual Basic keywords.

Fig. 4.2 Visual Basic keywords.

4.5 If/Then Selection Structure

- A selection structure chooses among alternative courses of action.
- It is a single-entry/single-exit structure
- Example

```
If studentGrade >= 60 Then
   Console.WriteLine(“Passed”)
End If
```

4.5 If/Then Selection Structure

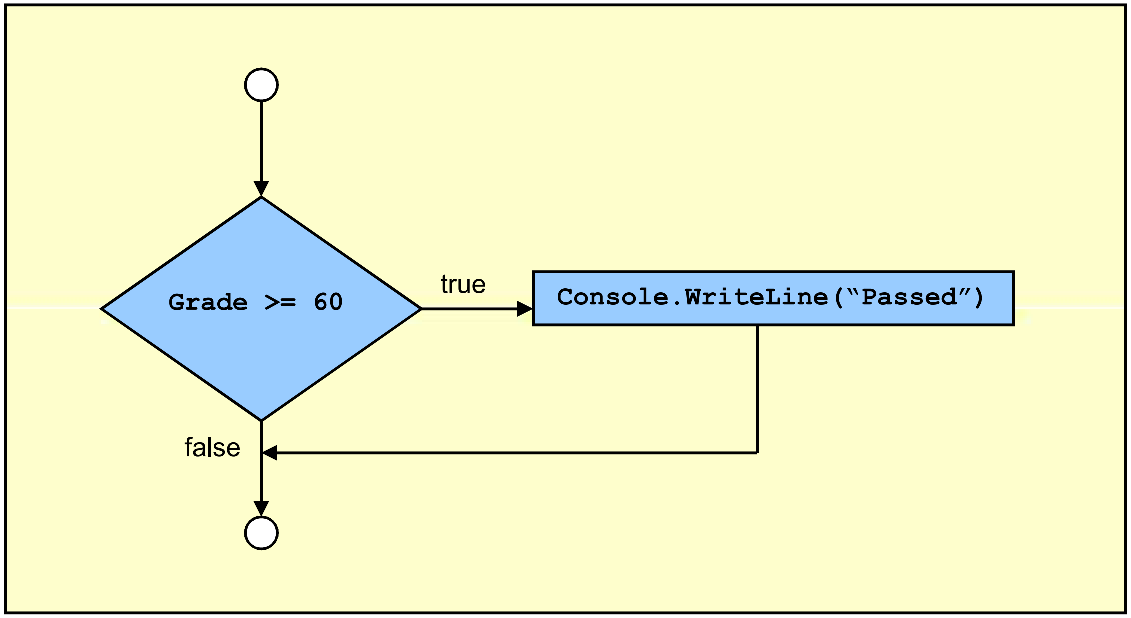


Fig. 4.3 Flowcharting a single-selection If/Then structure.

4.6 **If/Then/Else** Selection Structure

- Example

```
If studentGrade >= 60 Then
   Console.WriteLine(“Passed”)
Else
   Console.WriteLine(“Failed”)
End If
```

- Nested **If**/**Then**/**Else** structures
  - Test for multiple conditions by placing one structure inside the other.
  - **ElseIf** keyword

4.6 If/Then/Else Selection Structure

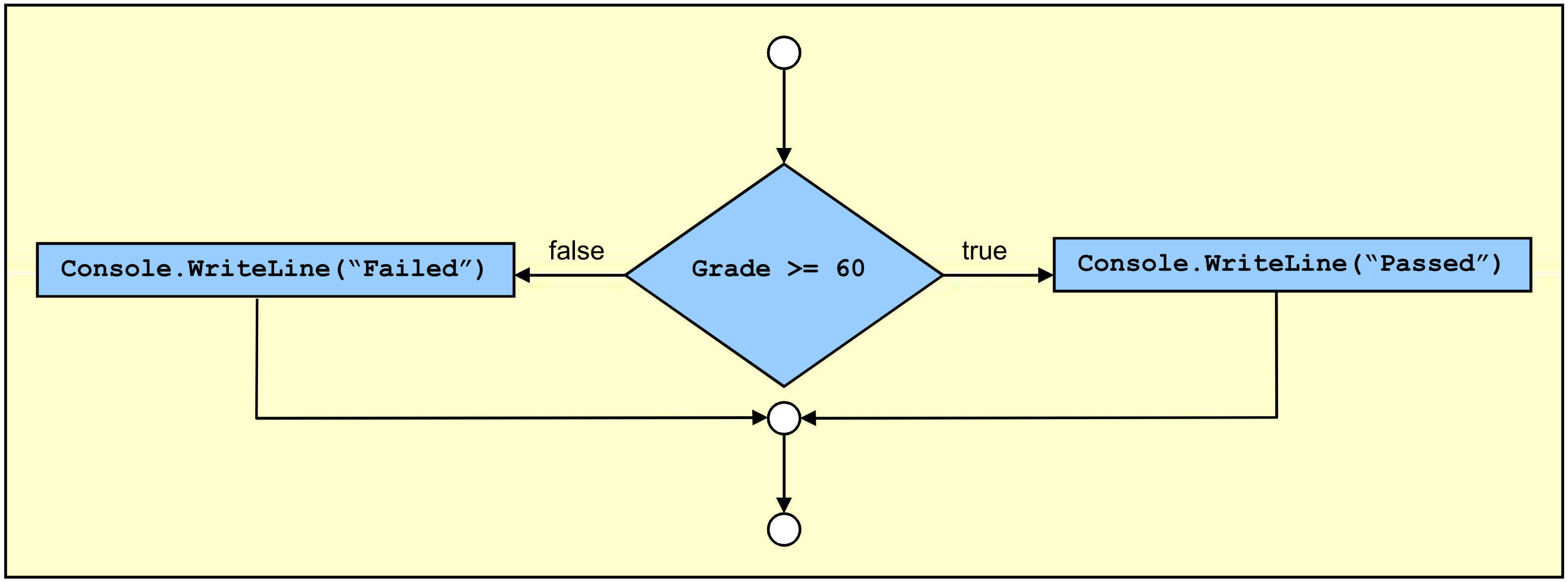


Fig. 4.4 Flowcharting a double-selection If/Then/Else structure.

4.7 While Repetition Structure

- Repetition structure
  - Allows the programmer to specify that an action should be repeated, depending on the value of a condition
- Example (pseudocode)

```
While there are more items on my shopping list
   Purchase next item
   Cross it off my list
```

While.vb

```
' Fig. 4.5: While.vb
' Demonstration of While structure.

Module modWhile

   Sub Main()
      Dim product As Integer = 2

      ' structure multiplies and displays product
      ' while product is less than 10
      While product <= 1000
         Console.Write("{0}   ", produ
         product = product * 2
      End While

      Console.WriteLine() ' write a blank line

      ' print result
      Console.WriteLine("Smallest power of 2 " & _
         "greater than 1000 is {0}", product)
      Console.ReadLine() ' prevents window from closing
   End Sub ' Main

End Module ' modWhile
```

The decision is tested each time the loop iterates

Program Output

```
2  4  8  16  32  64  128  256  512
Smallest power of 2 greater than 1000 is 1024
```

4.4 Control Structures

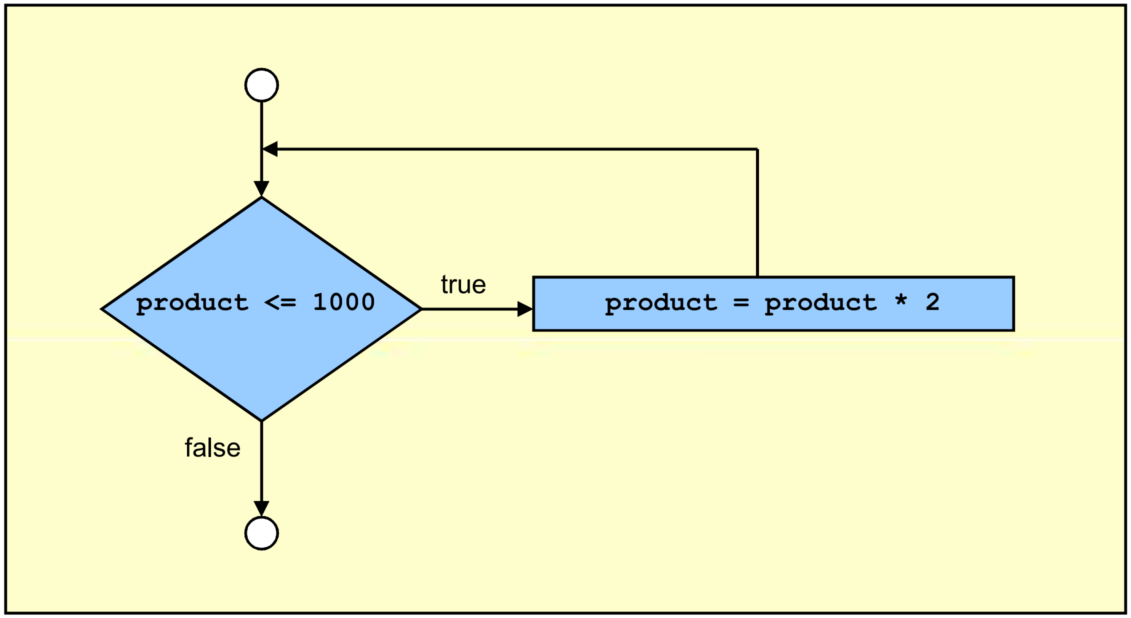


Fig. 4.6 Flowchart of the While repetition structure.

4.8 Do While/Loop Repetition Structure

- This structure behaves like the **`While`** repetition structure

DoWhile.vb

```
' Fig. 4.7: DoWhile.vb
' Demonstration of the Do While/Loop structure.

Module modDoWhile

   Sub Main()
      Dim product As Integer = 2

      ' structure multiplies and displays
      ' product while product is less tha
      Do While product <= 1000
         Console.Write("{0}  ", product)
         product = product * 2
      Loop

      Console.WriteLine() ' write a blank line

      ' print result
      Console.WriteLine("Smallest power of 2 " & _
         "greater than 1000 is {0}", product)
      Console.ReadLine() ' prevent window from closing
   End Sub

End Module ' modDoWhile
```

Failure to provide the body of the structure with an action that causes the condition to become false creates an infinite loop

Program Output

```
2  4  8  16  32  64  128  256  512
Smallest power of 2 greater than 1000 is 1024
```

4.8 Do While/Loop Repetition Structure

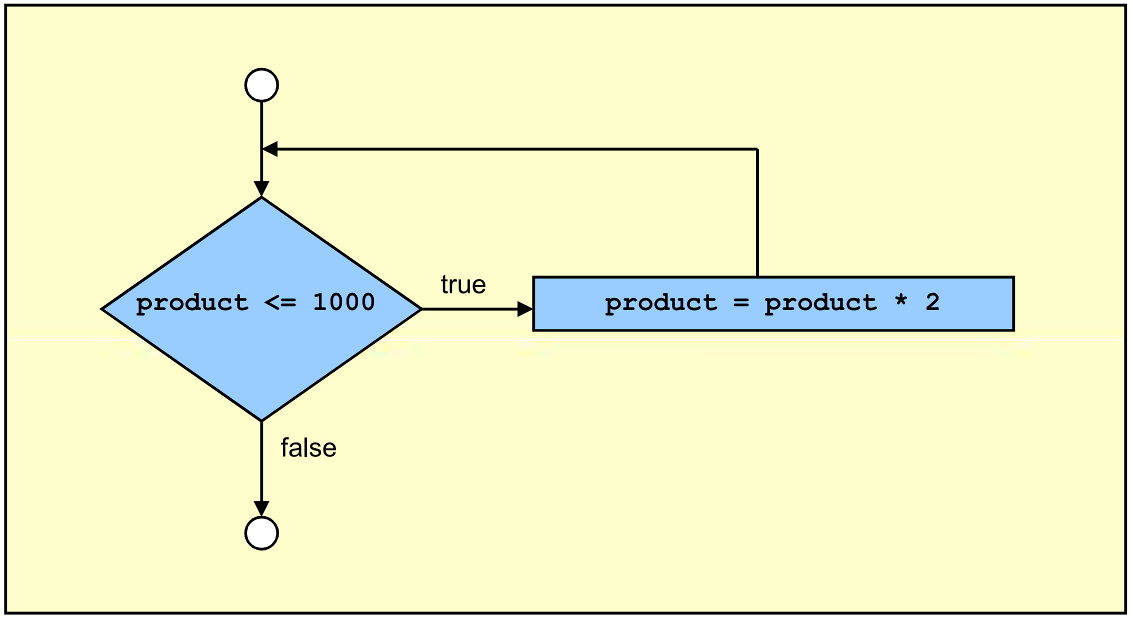


Fig. 4.8 Flowchart of a Do While/Loop repetition structure.

4.9 Do Until/Loop Repetition Structure

- It tests a condition for falsity for repetition to continue.

DoUntil.vb

```
' Fig. 4.9: DoUntil.vb
' Demonstration of the Do Until/Loop Structure.

Module modDoUntil

   Sub Main()
      Dim product As Integer = 2

      ' find first power of 2 greater than 1000
      Do Until product > 1000
         Console.Write("{0}  ", product)
         product = product * 2
      Loop

      Console.WriteLine() ' write a blank line

      ' print result
      Console.WriteLine("Smallest power of 2 " & _
         "greater than 1000 is {0}", product)
    End Sub ' Main

End Module ' modDoUntil
```

The loop ends when the condition becomes true

Program Output

```
2  4  8  16  32  64  128  256  512
Smallest power of 2 greater than 1000 is 1024
```

4.9 Do Until/Loop Repetition Structure

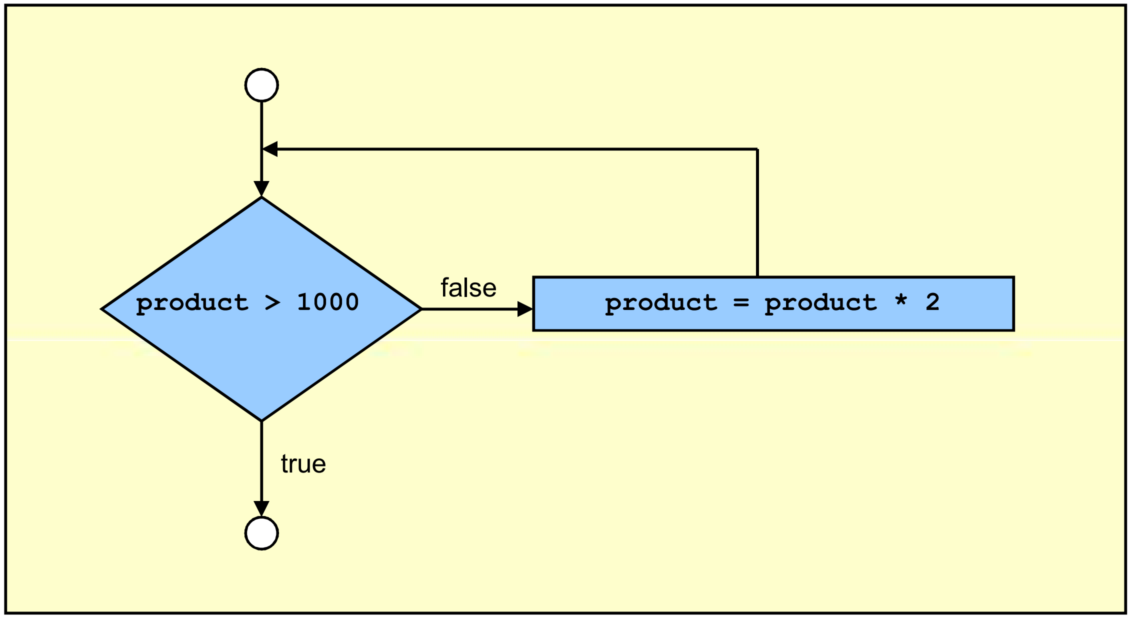


Fig. 4.10 Flowcharting the Do Until/Loop repetition structure.

4.10 Assignment Operators

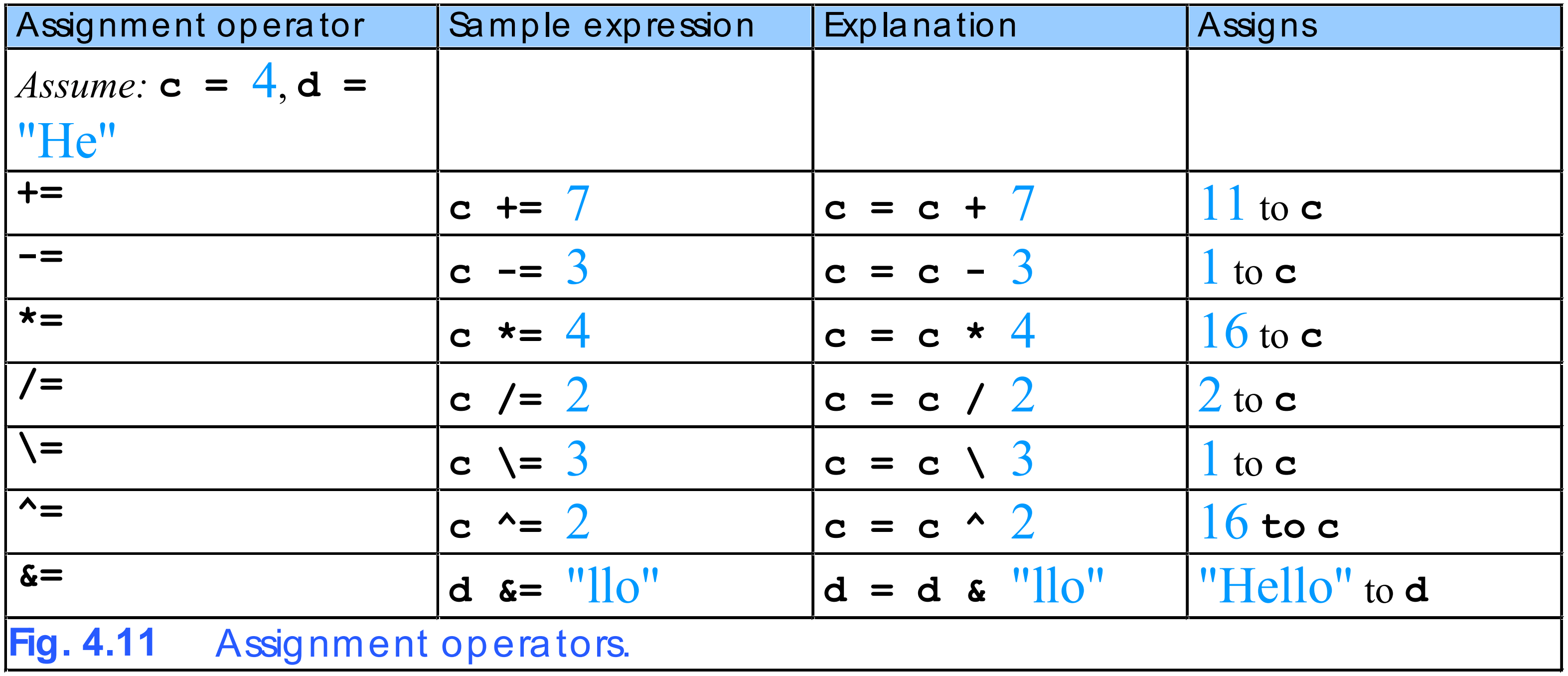
- Binary operators
  - **+**, **-**, *****, **^**, **&**, **/** or \

    variable **=** variable operator expression

    variable operator**=** expression
- Example
  - Addition assignment operator, **+=**

```
value = value + 3
value += 3
```

4.10 Assignment Operators



Assignment operator	Sample expression	Explanation	Assigns
Assume: `c = 4`, `d = "He"`			
`+=`	`c += 7`	`c = c + 7`	11 to `c`
`-=`	`c -= 3`	`c = c - 3`	1 to `c`
`*=`	`c *= 4`	`c = c * 4`	16 to `c`
`/=`	`c /= 2`	`c = c / 2`	2 to `c`
`\=`	`c \= 3`	`c = c \ 3`	1 to `c`
`^=`	`c ^= 2`	`c = c ^ 2`	16 `to c`
`&=`	`d &= "llo"`	`d = d & "llo"`	"Hello" to `d`

Fig. 4.11 Assignment operators.

Fig. 4.11 Assignment operators.

Assignment.vb

```
' Fig. 4.12: Assignment.vb
' Using an assignment operator to calculate a power of 2.

Module modAssignment

   Sub Main()
      Dim exponent As Integer ' power input by user
      Dim result As Integer = 2 ' number to raise to a power

      ' prompt user for exponent
      Console.Write("Enter an integer exponent: ")
      result = Console.ReadLine()

      result ^= exponent ' same as result = result ^ exponent
      Console.WriteLine(“result ^= exponent: {0}", result)

      result = 2 ' reset base value
      result = result ^ exponent
      Console.WriteLine(“result = result ^ exponent: {0}", result)

   End Sub ' Main

End Module ' modAssignment
```

Same effect on the variable **result**

Program Output

```
Enter an integer exponent: 8
result ^= exponent: 256
result = result ^ exponent: 256
```

4.11 Formulating Algorithms: Case Study 1 (Counter-Controlled Repetition)

- Counter-controlled repetition
  - Counter
    - Variable that specifies the number of times that a set of statements will execute

4.11 Formulating Algorithms: Case Study 1 (Counter-Controlled Repetition)

```
Set total to zero
Set grade counter to one

While grade counter is less than or equal to 10
    Input the next grade
    Add the grade to the total
    Add one to the grade counter

Set the class average to the total divided by 10
Print the class average
```

Fig. 4.13 Pseudocode algorithm that uses counter-controlled repetition to solve the class-average problem.

Outline

age1.vb

```
' Fig. 4.14: Average1.vb
' Using counter-controlled repeti

Module modAverage

   Sub Main()
      Dim total As Integer           ' sum of grades
      Dim gradeCounter As Integer    ' number of grades input
      Dim grade As Integer           ' grade input by user
      Dim average As Double          ' class average

      ' initialization pha
      total = 0
      gradeCounter = 1

      ' processing phase
      While gradeCounter <= 10

         ' prompt for input and read grade from user
         Console.Write("Enter
         grade = Console.ReadI

         total += grade     ' a

         gradeCounter += 1 ' add 1 to gradeCounter
      End While

      ' termination phase
      average = total / 10

      ' write a blank line and display class average
      Console.WriteLine()
      Console.WriteLine("Class average is {0}", average)

```

total accumulates the sum of the grades

gradeCounter counts the number of grades entered

The **While** structure iterates while the value of **gradeCounter** is less than or equal to 10.

gradeCounter is incremented to indicate that a grade has been processed. The condition eventually becomes false, terminating the loop

```
   End Sub ' Main

End Module ' modAverage
```

```
Enter integer grade: 89
Enter integer grade: 70
Enter integer grade: 73
Enter integer grade: 85
Enter integer grade: 64
Enter integer grade: 92
Enter integer grade: 55
Enter integer grade: 57
Enter integer grade: 93
Enter integer grade: 67

Class average is 74.5
```

Outline

`Average1.vb`

Program Output

4.12 Formulating Algorithms with Top-Down, Stepwise Refinement: Case Study 2 (Sentinel-Controlled Repetition)

- Sentinel value
  - Indicates “end of data entry”
  - Choosing a sentinel value that is also a legitimate data value could result in a logic error
- Top-down, stepwise refinement
  - The top is a single statement that conveys the overall function of the program
  - Each refinement is a complete specification of the algorithm; only the level of detail in each refinement varies

4.12 Formulating Algorithms with Top-Down, Stepwise Refinement: Case Study 2 (Sentinel-Controlled Repetition)

- Algorithms (three phases)
  - Initialization phase
    - Initializes the program variables
  - Processing phase
    - Inputs data values and adjusts program variables accordingly
  - Termination phase
    - Calculates and prints the results

4.12 Formulating Algorithms with Top-Down, Stepwise Refinement: Case Study 2 (Sentinel-Controlled Repetition)

```
Initialize total to zero
Initialize counter to zero

Input the first grade (possibly the sentinel)

While the user has not as yet entered the sentinel
    Add this grade to the running total
    Add one to the grade counter
    Input the next grade (possibly the sentinel)

If the counter is not equal to zero
    Set the average to the total divided by the counter
    Print the average
Else
    Print “No grades were entered”
```

Fig. 4.15 Pseudocode algorithm that uses sentinel-controlled repetition to solve the class-average problem.

◀ ▶

ClassAverage2.vb

```
' Fig. 4.16: ClassAverage2.vb
' Using sentinel-controlled repetition to
' display a class average.

Module modClassAverage

   Sub Main()
      Dim total As Integer          ' sum of grades
      Dim gradeCounter As Integer   ' number of grades input
      Dim grade As Integer          ' grade input by user
      Dim average As Double         ' average of all grades

      ' initialization phase
      total = 0
      gradeCounter = 0

      ' processing phase
      ' prompt for input and read grade from user
      Console.Write("Enter integer grade, -1
      grade = Console.ReadLine()

      ' sentinel-controlled loop where -1 is
      While grade <> -1

         total += grade    ' add gradeValue to total
         gradeCounter += 1 ' add 1 to gradeCounter

         ' prompt for input and read grade from user
         Console.Write("Enter integer grade, -1 to Quit: ")
         grade = Console.ReadLine()
      End While

```

In sentinel-controlled repetition, a value is read before the program reaches the **`While`** structure

In a sentinel-controlled loop, the prompts requesting data entry should remind the user of the sentinel value

Outline

ClassAverage2.vb

```
      ' termination phase
      If gradeCounter <> 0 Then
         average = total / gradeCounter

         ' display class average
         Console.WriteLine()
         Console.WriteLine("Class average is {0:F}", average)
      Else ' if no grades were entered
         Console.WriteLine("No grades were entered")
      End If

   End Sub ' Main

End Module ' modClassAverage

```

Program Output

```
Enter Integer Grade, -1 to Quit: 97
Enter Integer Grade, -1 to Quit: 88
Enter Integer Grade, -1 to Quit: 72
Enter Integer Grade, -1 to Quit: -1

Class average is 85.67
```

4.13 Formulating Algorithms with Top-Down, Stepwise Refinement: Case Study 3 (Nested Control Structures)

```
Initialize passes to zero
Initialize failures to zero
Initialize student to one

While student counter is less than or equal to ten
    Input the next exam result

    If the student passed
        Add one to passes
    Else
        Add one to failures

    Add one to student counter

Print the number of passes
Print the number of failures

If more than eight students passed
    Print "Raise tuition"
```

Fig. 4.17 Pseudocode for examination-results problem.

Analysis.vb

```
' Fig. 4.18: Analysis.vb
' Using counter-controlled repetition to display exam results.

Module modAnalysis

   Sub Main()
      Dim passes As
      Dim failures A                                       es
      Dim student As Integer = 1      ' student counter
      Dim result As String            ' one exam result

      ' process 10 exam results; counter-controlled loop
      While student <=
         Console.Write                                         ")
         result = Cons

         ' nested control structures
         If result = "P" Then
            passes += 1    ' increment number of passes
         Else
            failures += 1  ' increment number of failures
         End If

         student
      End While

      ' display exam results
      Console.WriteLine("Passed: {0}{1}Failed: {2}", passes, _
         vbCrLf, failures)

      ' raise tuition if than 8 students pass
      If passes > 8 Then
         Console.WriteLine("Raise Tuition")
      End If

```

The **While** loop inputs and processes the 10 examination results

The **If**/**Then**/**Else** structure is a nested control. It is enclosed inside the **While**.

Identifier **vbCrLf** is the combination of the carriage return and linefeed characters

```
      End Sub ' Main

   End Module ' modAnalysis
```

Analysis.vb

Program Output

```
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
F
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
P
Passed: 9
Failed: 1
Raise Tuition
```

```
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
F
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
F
Enter result (P = pass, F = fail)
F
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
P
Enter result (P = pass, F = fail)
F
Enter result (P = pass, F = fail)
P
Passed:  6
Failed:  4
```

`Analysis.vb`

Program Output

4.14 Formulating Algorithms with Top-Down, Stepwise Refinement: Case Study 4 (Nested Repetition Structures)

```
Initialize side to the value input
Initialize row to 1

If side is less than or equal to 20

    While row is less than or equal to side
        Set column to one

        While column is less than or equal to side
            Print *
            Increment column by one

        Print a line feed/carriage return
        Increment row by one

Else
    Print "Side is too large"
```

Fig. 4.19 Second refinement of the pseudocode.

PrintSquare.vb

```
1    ' Fig. 4.20: PrintSquare.vb
2    ' Program draws square of $.
3
4    Module modPrintSquare
5
6       Sub Main()
7          Dim side As Integer        ' square side
8          Dim row As Integer = 1     ' current row
9          Dim column As Integer      ' current column
10
11         ' obtain side from user
12         Console.Write("Enter side length (must be 20 or less): ")
13         side = Console.ReadLine()
14
15         If side <= 20 Then         ' If true, while is tested
16
17            ' this while is nested inside the If
18            While row <= side       ' controls row
19               column = 1
20
21               ' this loop prints one row of * characters
22               ' and is nested inside the While in line 18
23               While (column <= side)
24                  Console.Write("* ")  ' print * characters
25                  column += 1          ' increment column
26               End While
27
28               Console.WriteLine()  ' position cursor on next line
29               row += 1             ' increment row
30            End While
31         Else ' condition (side <= 20) is false
32            Console.WriteLine("Side too large")
33         End If
34
```

Three levels of nesting

Each iteration of the inner loop prints a single *

Outline

```
End Sub ' Main

End Module ' modPrintSquare
```

```
Enter side length (must be 20 or less): 8
* * * * * * * *
* * * * * * * *
* * * * * * * *
* * * * * * * *
* * * * * * * *
* * * * * * * *
* * * * * * * *
* * * * * * * *
```

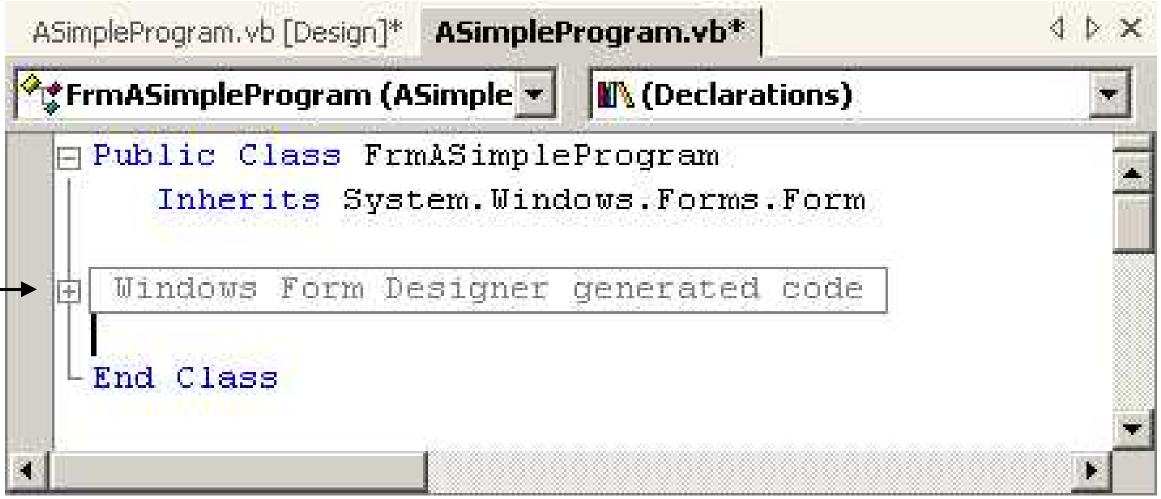
PrintSquare.vb

Program Output

4.15 Introduction to Windows Application Programming

- Windows application
  - Consists of at least one class
    - **`Inherits`** from class **`Form`**
    - **`Form`** is called the superclass or base class
  - Keyword **`Class`**
    - Begins a class definition and is followed by the class name
  - Keyword **`Inherits`**
    - Indicates that the class inherits existing pieces from another class

4.15 Introduction to Windows Application Programming



```
Public Class FrmASimpleProgram
    Inherits System.Windows.Forms.Form

 Windows Form Designer generated code 

End Class
```

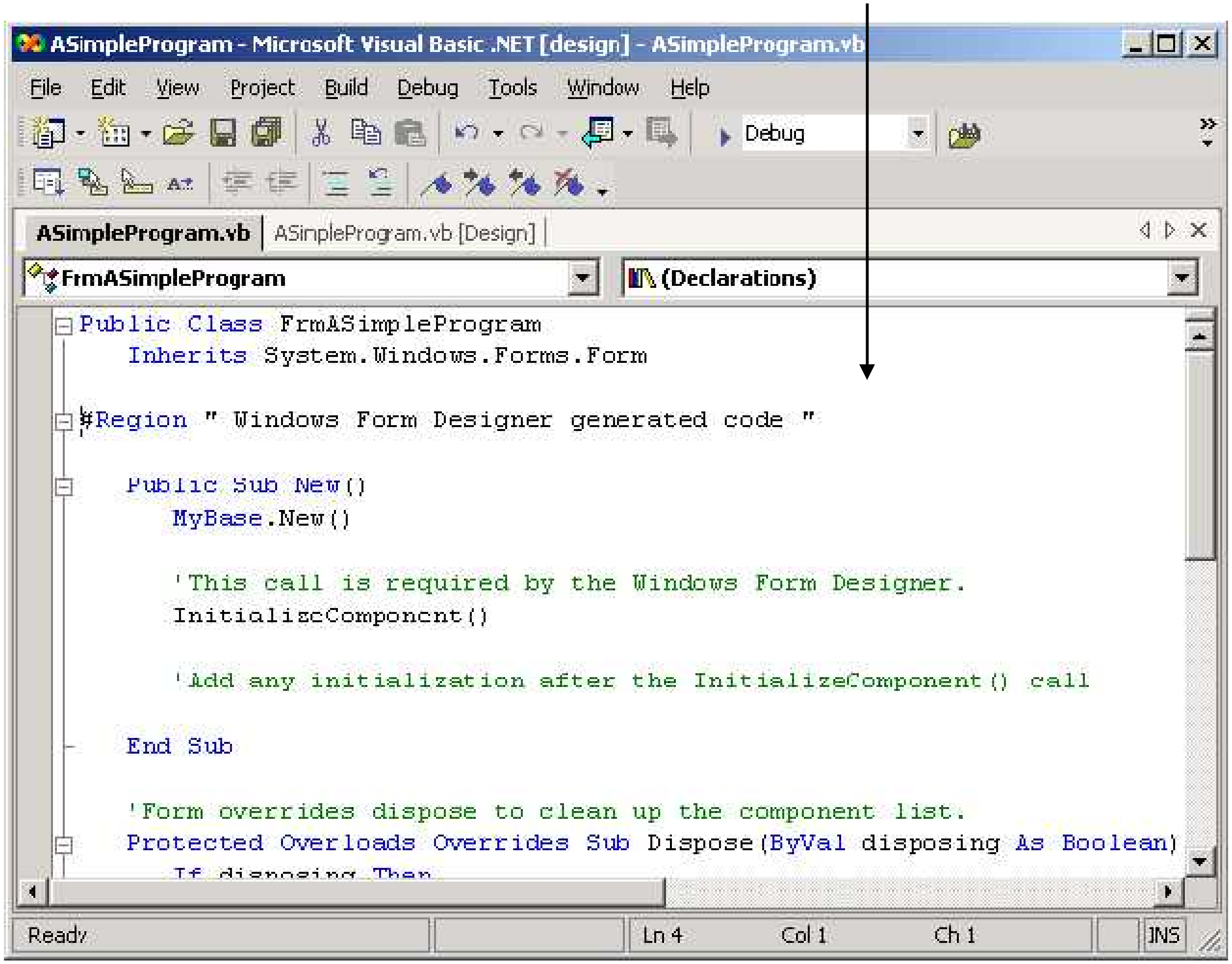
Fig. 4.21 IDE showing program code for Fig. 2.15.

4.15 Introduction to Windows Application Programming

- **`Windows Form Designer generated code`**
  - Collapsed by default
  - The code is created by the IDE and normally is not edited by the programmer
  - Present in every Windows application

4.15 Introduction to Windows Application Programming

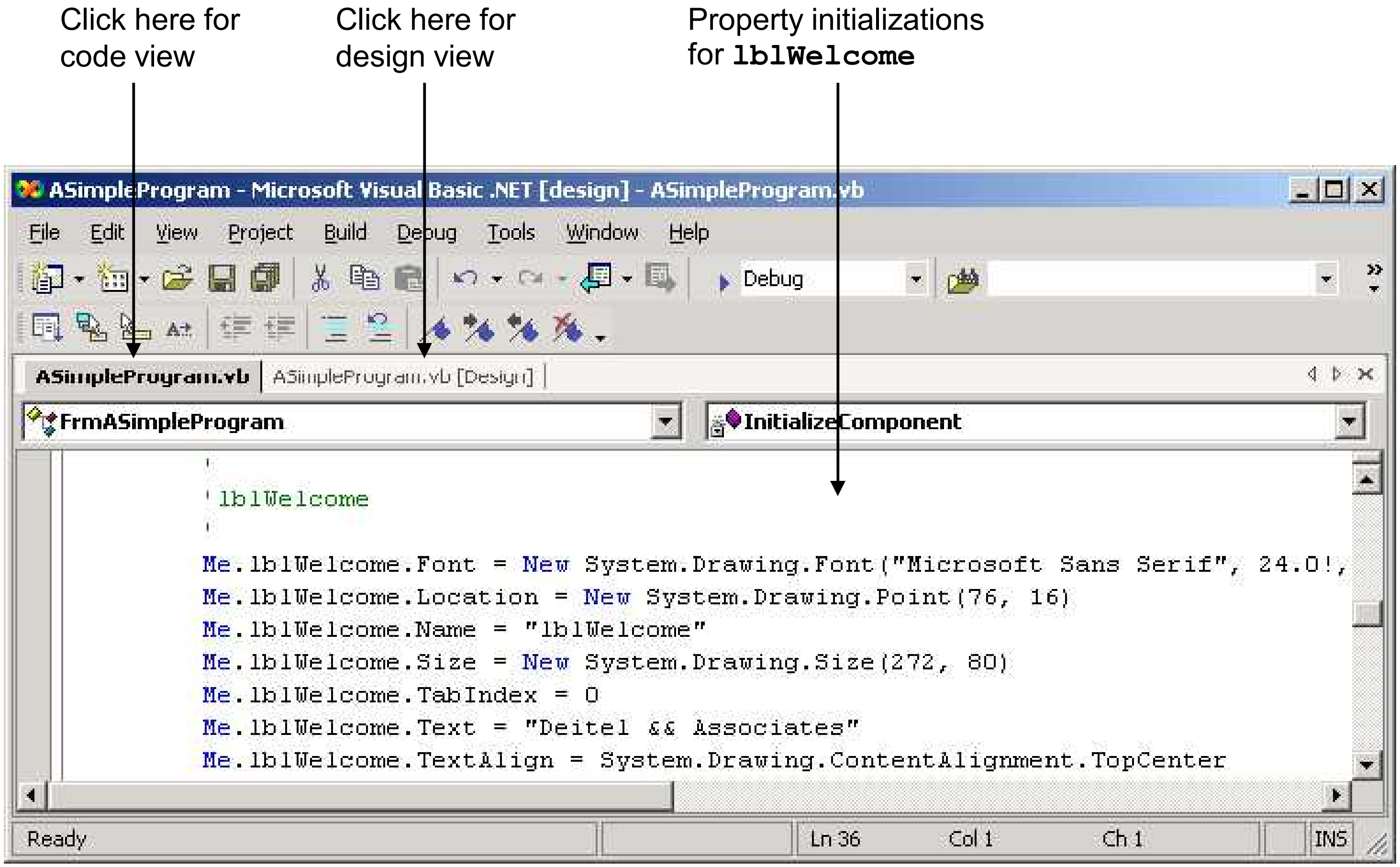
Expanded code


ASimpleProgram - Microsoft Visual Basic .NET [design] - ASimpleProgram.vb
File Edit View Project Build Debug Tools Window Help
Debug
ASimpleProgram.vb
ASimpleProgram.vb [Design]
FrmASimpleProgram
(Declarations)
Ready
Ln 4
Col 1
Ch 1
INS

```
Public Class FrmASimpleProgram
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
```

Fig. 4.22 Windows Form Designer generated code when expanded.

4.15 Introduction to Windows Application Programming



Click here for code view

Click here for design view

Property initializations for **`lblWelcome`**

ASimpleProgram - Microsoft Visual Basic .NET [design] - ASimpleProgram.vb
File Edit View Project Build Debug Tools Window Help
Debug
ASimpleProgram.vb
ASimpleProgram.vb [Design]
FrmASimpleProgram
InitializeComponent
Ready
Ln 36
Col 1
Ch 1
INS

```
'
'lblWelcome
'
Me.lblWelcome.Font = New System.Drawing.Font("Microsoft Sans Serif", 24.0!,
Me.lblWelcome.Location = New System.Drawing.Point(76, 16)
Me.lblWelcome.Name = "lblWelcome"
Me.lblWelcome.Size = New System.Drawing.Size(272, 80)
Me.lblWelcome.TabIndex = 0
Me.lblWelcome.Text = "Deitel && Associates"
Me.lblWelcome.TextAlign = System.Drawing.ContentAlignment.TopCenter
```

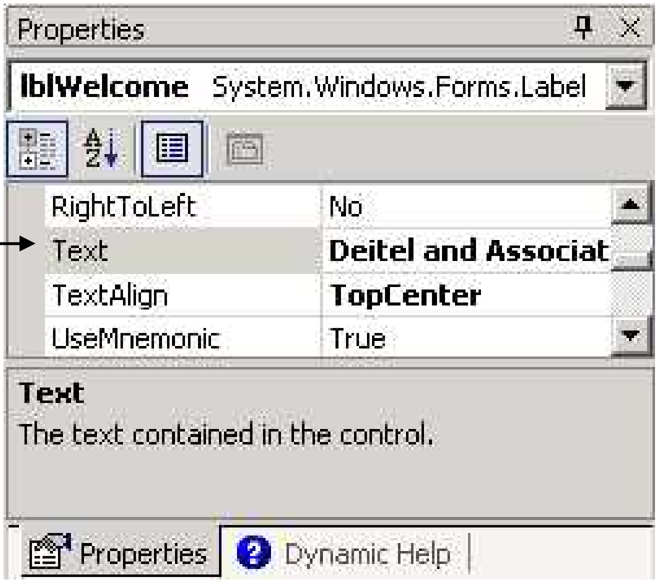
Fig. 4.23 Code generated by the IDE for lblWelcome.

4.15 Introduction to Windows Application Programming

- How IDE updates the generated code
  1. Modify the file name
     - Change the name of the file to **`ASimpleProgram.vb`**
  2. Modify the label control's **`Text`** property using the **Properties** window
     - Change the property of the label to "**`Deitel and Associates`**"
  3. Examine the changes in the code view
     - Switch to code view and examine the code
  4. Modifying a property value in code view
     - Change the string assigned to **`Me.lblWelcome.Text`** to "**`Visual Basic .NET`**"

4.15 Introduction to Windows Application Programming

Text property



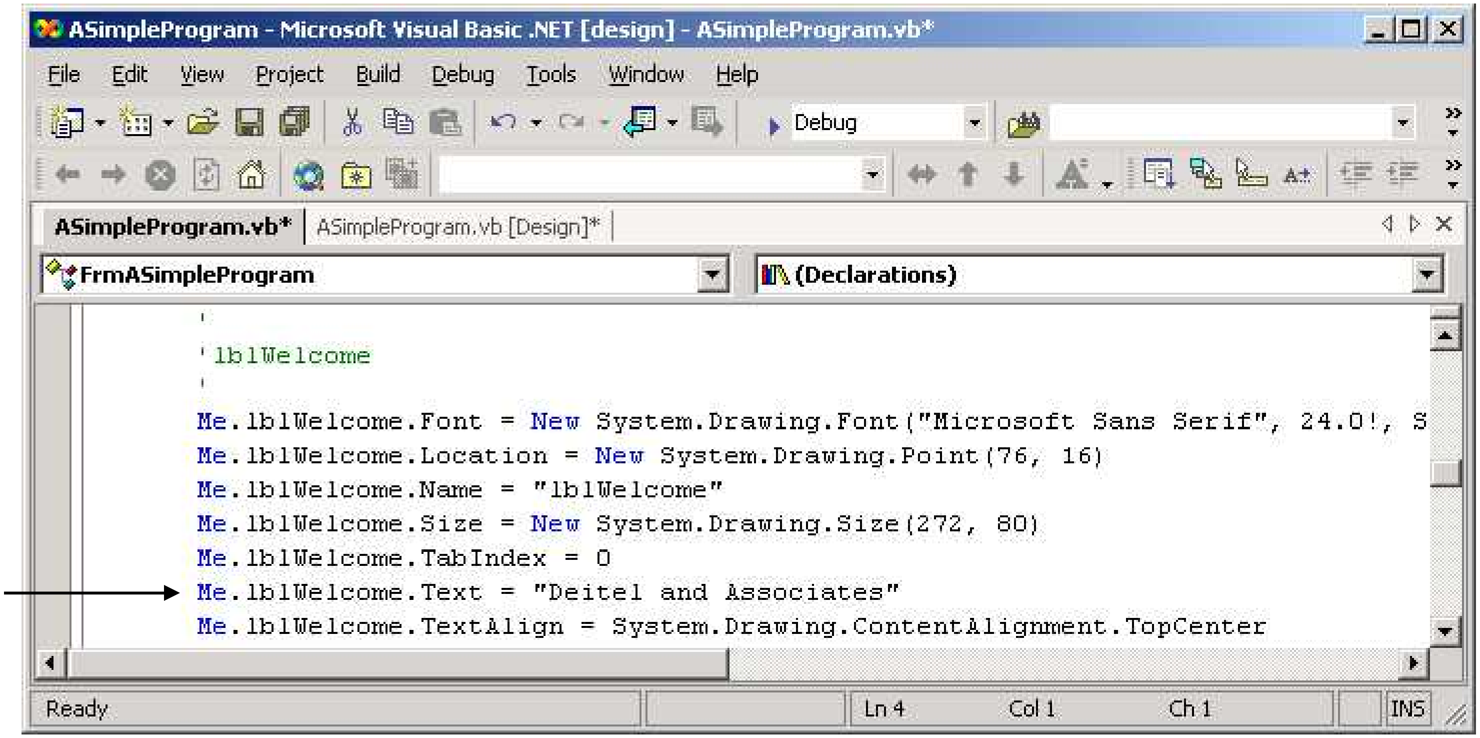
lblWelcome System.Windows.Forms.Label

RightToLeft	No
Text	Deitel and Associat
TextAlign	TopCenter
UseMnemonic	True

Text
The text contained in the control.

Fig. 4.24 Using the Properties window to set a property value.

4.15 Introduction to Windows Application Programming

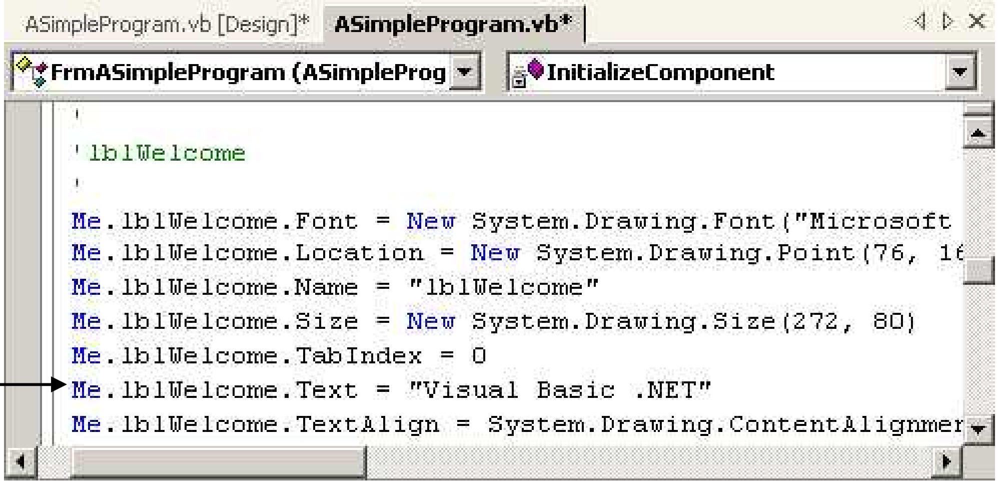

ASimpleProgram - Microsoft Visual Basic .NET [design] - ASimpleProgram.vb*
File Edit View Project Build Debug Tools Window Help
Debug
ASimpleProgram.vb*
ASimpleProgram.vb [Design]*
FrmASimpleProgram
(Declarations)
Ready
Ln 4
Col 1
Ch 1
INS

```
'
'lblWelcome
'
Me.lblWelcome.Font = New System.Drawing.Font("Microsoft Sans Serif", 24.0!, S
Me.lblWelcome.Location = New System.Drawing.Point(76, 16)
Me.lblWelcome.Name = "lblWelcome"
Me.lblWelcome.Size = New System.Drawing.Size(272, 80)
Me.lblWelcome.TabIndex = 0
Me.lblWelcome.Text = "Deitel and Associates"
Me.lblWelcome.TextAlign = System.Drawing.ContentAlignment.TopCenter
```

Text property

Fig. 4.25 Windows Form Designer generated code reflecting new property values.

4.15 Introduction to Windows Application Programming



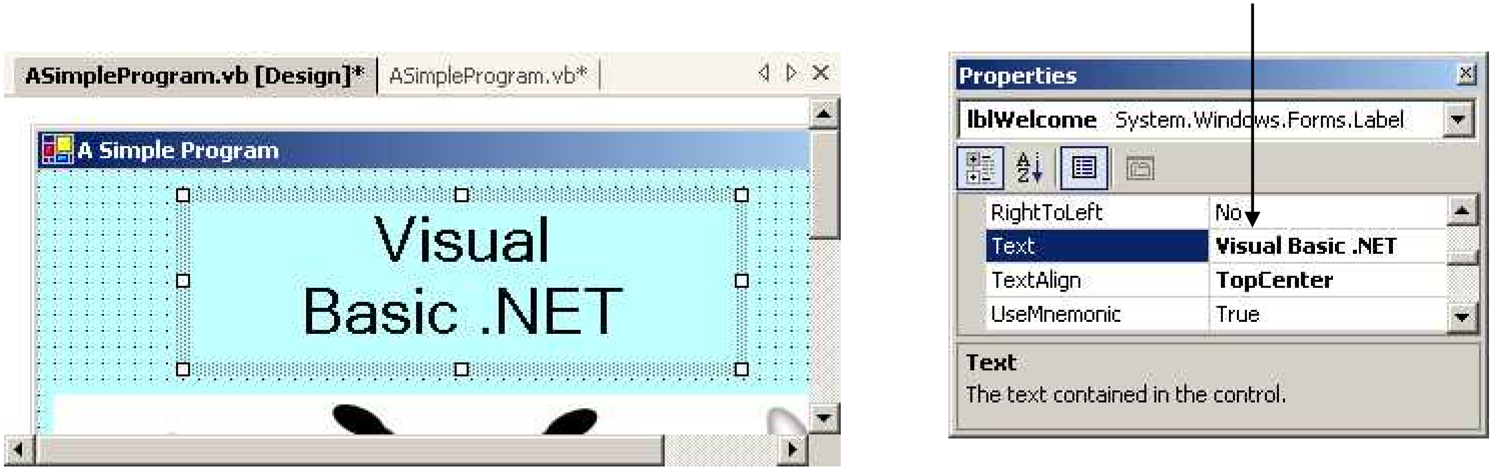
```
'
'lblWelcome
'
Me.lblWelcome.Font = New System.Drawing.Font("Microsoft
Me.lblWelcome.Location = New System.Drawing.Point(76, 16
Me.lblWelcome.Name = "lblWelcome"
Me.lblWelcome.Size = New System.Drawing.Size(272, 80)
Me.lblWelcome.TabIndex = 0
Me.lblWelcome.Text = "Visual Basic .NET"
Me.lblWelcome.TextAlign = System.Drawing.ContentAlignmer
```

`Text` property

Fig. 4.26 Changing a property in the code view editor.

4.15 Introduction to Windows Application Programming

`Text` property value



ASimpleProgram.vb [Design]* | ASimpleProgram.vb*

A Simple Program

Visual Basic .NET

Properties

lblWelcome System.Windows.Forms.Label

RightToLeft	No
Text	**Visual Basic .NET**
TextAlign	**TopCenter**
UseMnemonic	True

Text

The text contained in the control.

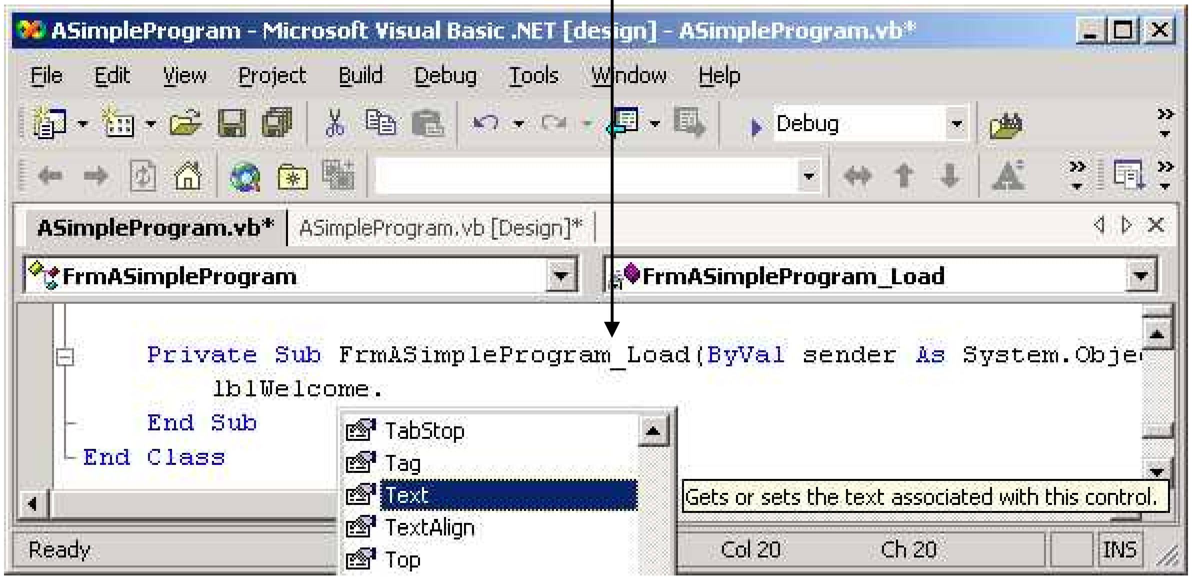
Fig. 4.27 New Text property value reflected in design mode.

4.15 Introduction to Windows Application Programming

5. Change the label's Text Property at runtime
   - Add a method named **`FrmASimpleProgram_Load`** to the class
   - Add the statement **`lblWelcome.Text = "Visual Basic"`** in the body of the method definition
6. Examine the results of the **`FrmASimpleProgram_Load`** method
   - Select **Build > Build Solution** then **Debug > Start**

4.15 Introduction to Windows Application Programming

FrmASimpleProgram_Load method

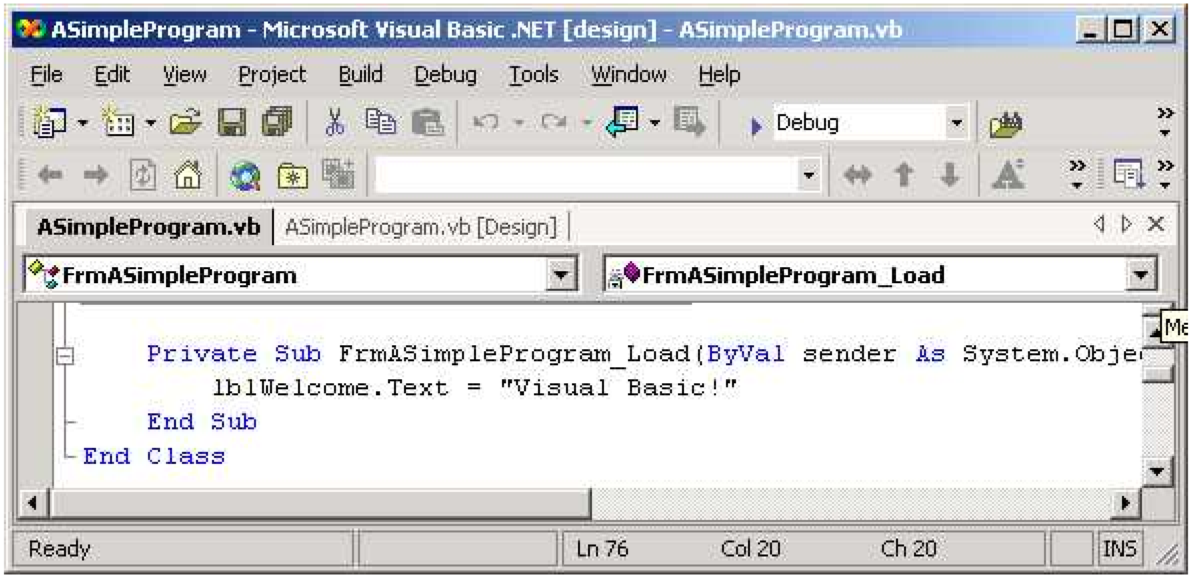


```
Private Sub FrmASimpleProgram_Load(ByVal sender As System.Obje
    lblWelcome.
End Sub
End Class
```

Gets or sets the text associated with this control.

Fig. 4.28 Adding program code to FrmASimpleProgram_Load.

4.15 Introduction to Windows Application Programming



```
    Private Sub FrmASimpleProgram_Load(ByVal sender As System.Obje
        lblWelcome.Text = "Visual Basic!"
    End Sub
End Class
```

Fig. 4.29 Method FrmASimpleProgram_Load containing program code.

4.15 Introduction to Windows Application Programming

7. Terminate program execution
    - Click the close button to terminate program execution