Repetition

Chapter 6

Chapter 6 – Repetition

- 6.1 Do Loops
- 6.2 Processing Lists of Data with Do Loops
- 6.3 For...Next Loops
- 6.4 A Case Study: Analyze a Loan

6.1 Do Loops

- A loop is one of the most important structures in programming.
- Used to repeat a sequence of statements a number of times.
- The Do loop repeats a sequence of statements either *as long as* or *until* a certain condition is true.

Do Loop Syntax

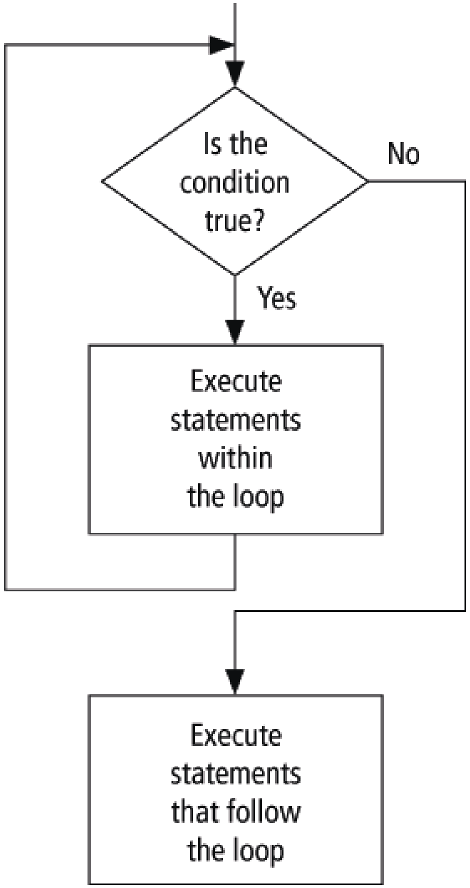
```
Do While condition
   statement(s)
Loop
```

Condition is tested,
If it is true,
the loop is run.
If it is false,
the statements
following the
Loop statement
are executed.

These statements are inside the body of the loop and are run if the condition above is true.

Pseudocode /Flow Chart for a Do Loop

```
Do While condition is true
    Processing step(s)
Loop
```


Is the condition true?
No
Yes
Execute statements within the loop
Execute statements that follow the loop

EXAMPLE 1

```
Private Sub btnDisplay_Click(...) _
                  Handles btnDisplay.Click
  'Display the numbers from 1 to 7
  Dim num As Integer = 1
  Do While num <= 7
    lstNumbers.Items.Add(num)
    num += 1 'Add 1 to the value of num
  Loop
End Sub
```

Example: Repeat Request as Long as Response is Incorrect

```
Dim passWord As String = ""
Do While passWord <> "SHAZAM"
  passWord = InputBox("What is the password?")
  passWord = passWord.ToUpper
Loop
```

passWord is the loop control variable because the value stored in passWord is what is tested to determine if the loop should continue or stop.

POST TEST LOOP

```
Do
   statement(s)
Loop Until condition
```

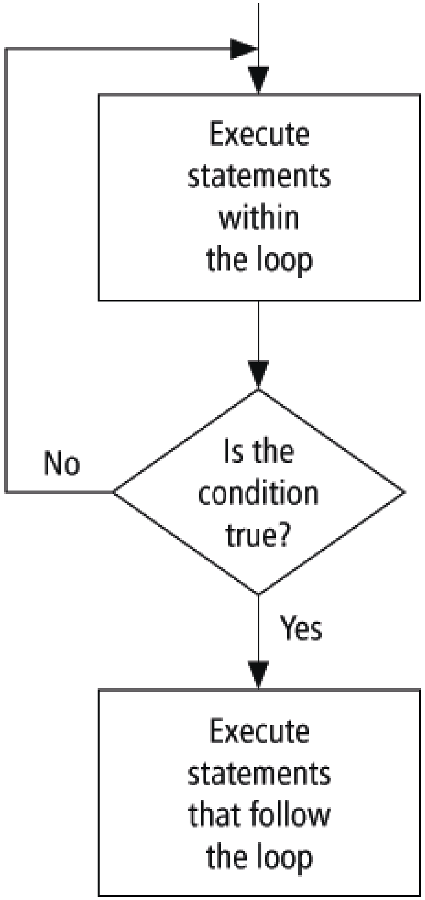
Loop is executed once and then the condition is tested. If it is false, the loop is run again. If it is frue, the statements following the Loop statement are executed.

Example: Repeat Request Until Proper Response is Given

```
Do
   passWord = InputBox("What is the password?")
   passWord = passWord.ToUpper
Loop Until passWord = "SHAZAM"
```

Pseudocode and Flowchart for a Post-Test Loop

```
Do
   statement(s)
Loop Until condition is true
```


Execute statements within the loop
Is the condition true?
No
Yes
Execute statements that follow the loop

What's the diff?

```
Do
   statement(s)
Loop Until condition
```

```
Do While condition
   statement(s)
Loop
```

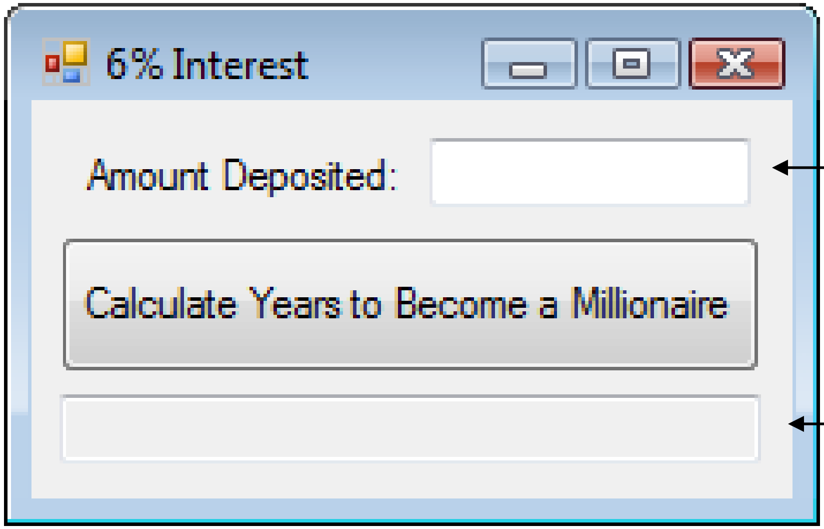
What's the difference between a

Do Until

and

Do While?

Example 4: Form



txtAmount

txtWhen

Example 4: Code

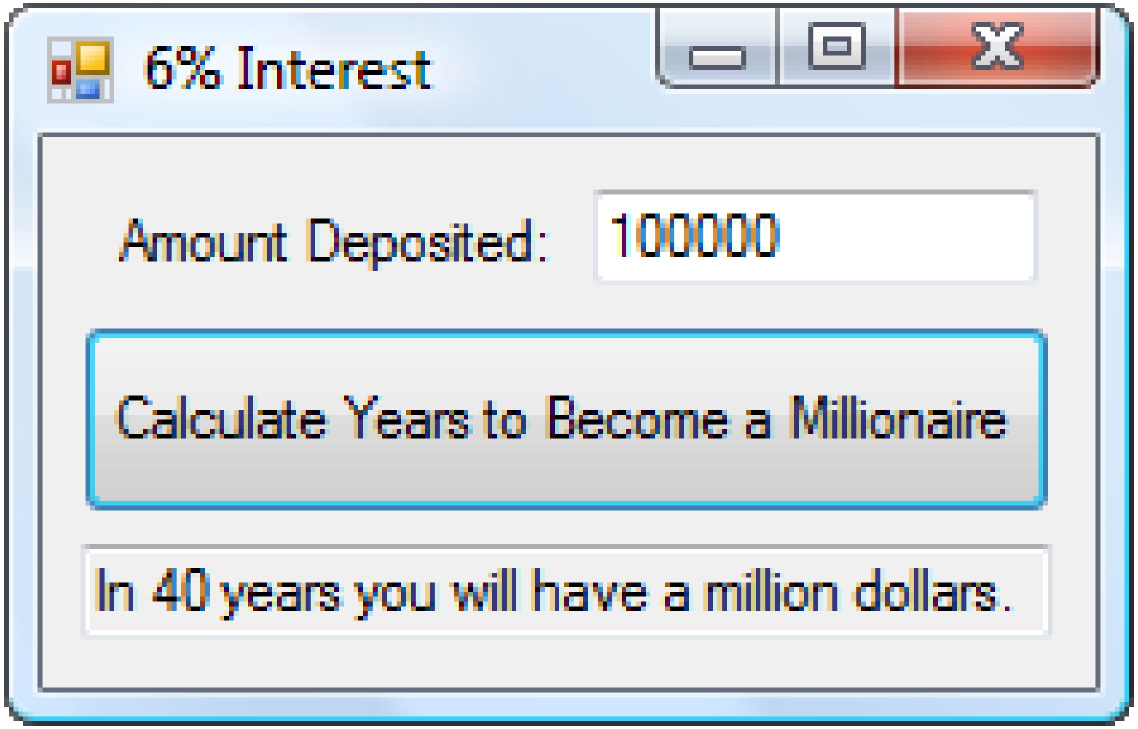
```
Private Sub btnCalculate_Click(...) Handles
                                btnCalculate.Click
  Dim balance As Double, numYears As Integer
  balance = CDbl(txtAmount.Text)
  Do While balance < 1000000
    balance += 0.06 * balance
    numYears += 1
  Loop
  txtWhen.Text = "In " & numYears & _
     " years you will have a million dollars."
End Sub
```

See how bad this code is without comments?

Example 4: Code

```
'calculate how long it'll take the balance to reach $1m
Private Sub btnCalculate_Click(...) Handles btnCalculate.Click
  Dim balance As Double, numYears As Integer

  'ask what the current balance is
  balance = CDbl(txtAmount.Text)

  'loop until the balance reaches $1m
  Do While balance < 1000000
    balance += 0.06 * balance
    numYears += 1
  Loop

  'display a message
  txtWhen.Text = "In " & numYears & " years you will have a million dollars."
End Sub
```

Example 4: Output



Amount Deposited: 100000

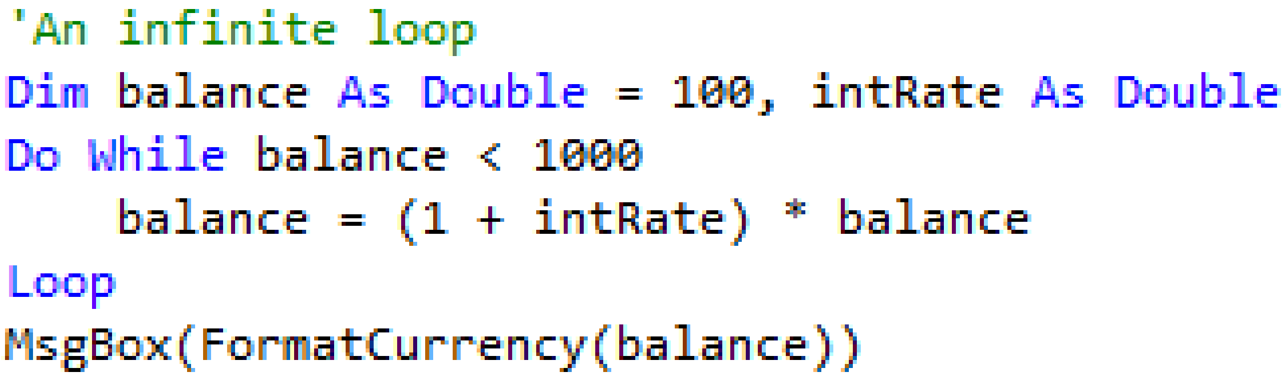
In 40 years you will have a million dollars.

COMMENTS

- Be careful to avoid **infinite** loops – loops that never end
- Visual Basic allows for the use of either the **While** keyword or the **Until** keyword at the top or the bottom of a loop
- This textbook will use only **While** at the top and only **Until** at the bottom

INFINITE LOOP

Why?



```
'An infinite loop
Dim balance As Double = 100, intRate As Double
Do While balance < 1000
    balance = (1 + intRate) * balance
Loop
MsgBox(FormatCurrency(balance))
```

6.2 Processing Lists of Data with Do Loops

- Peek Method
- Counters and Accumulators
- Flags
- Nested Loops

Processing Lists of Data with Do Loops

- Display all or selected items from lists
- Search lists for specific items
- Perform calculations on the numerical entries of a list

Peek Method

- Data to be processed are often retrieved from a file by a Do loop
- To determine if we have reached the end of the file from which we are reading, we use the Peek method.

Peek Example

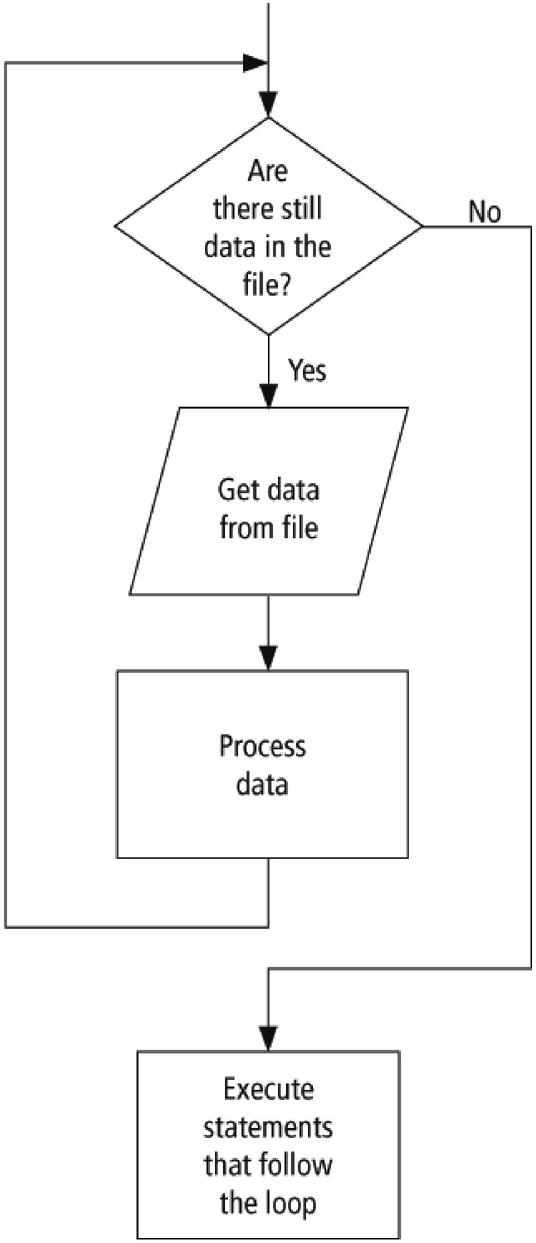
- Suppose a file has been opened as a StreamReader object named *sr*.
- **`sr.Peek`** is the ANSI value of the first character of the line about to be read with ReadLine. If the end of the file has been reached, the value of **`sr.Peek`** is -1

Example 1: Display the Total Contents of a File

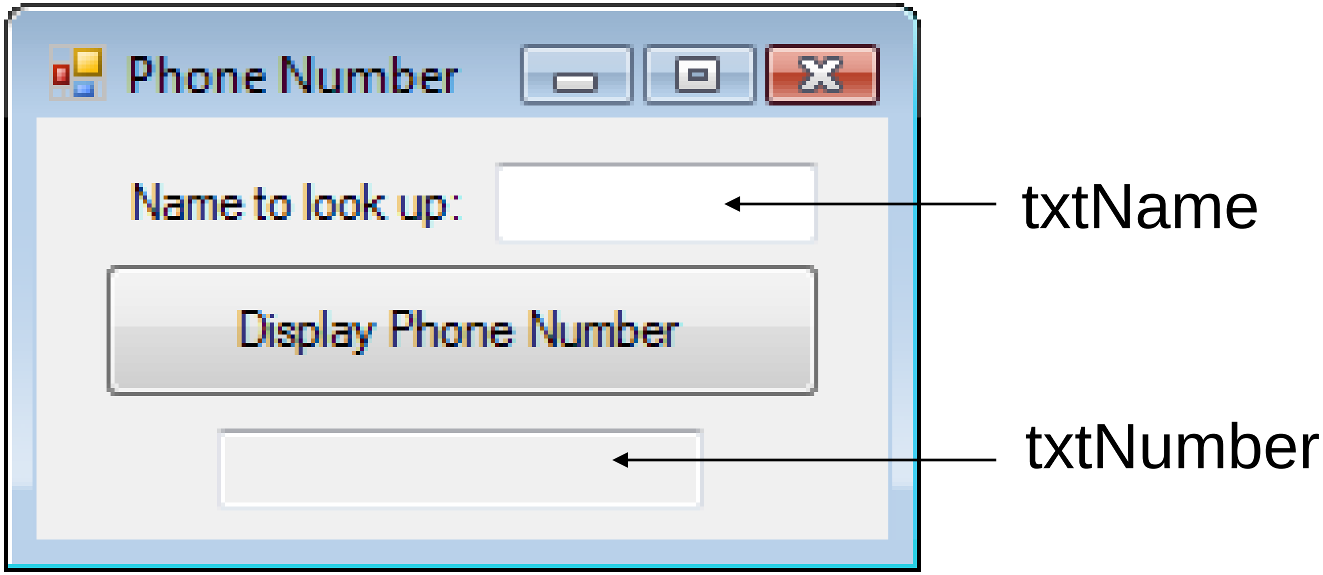
```
Dim sr As IO.StreamReader = _

  IO.File.OpenText("PHONE.TXT")
lstNumbers.Items.Clear()
Do While sr.Peek <> -1
  name = sr.ReadLine
  phoneNum = sr.ReadLine
  lstNumbers.Items.Add(name & "  " _
                      & phoneNum)
Loop
sr.Close()
```

Pseudocode and Flowchart for Processing Data from a File

```
Do While there are still data in the file
    Get an item of data
    Process the item
Loop
```


Are there still data in the file?
No
Yes
Get data from file
Process data
Execute statements that follow the loop

Example 2: Form



Example 2: Partial Code

```
Do While (name <> txtName.Text) _
                And (sr.Peek <> -1)
   name = sr.ReadLine
   phoneNum = sr.ReadLine
Loop
```

As long as the name being searched for has not been found AND the end of the file has not been reached, the loop will continue

Counters and Accumulators

- A **counter** is a numeric variable that keeps track of the number of items that have been processed.
- An **accumulator** is a numeric variable that totals numbers.

File COINS.TXT

1

1

5

10

10

25

Count the number of coins and determine the total value

Example 3: Partial Code

```
Dim numCoins As Integer = 0
Dim sum As Integer = 0
Dim coin As String
Do While sr.Peek <> -1
  coin = sr.ReadLine
  numCoins += 1
  sum += CDbl(coin)
Loop
```

sum is an accumulator. It is used to total up the values of the coins.

numCoins is a counter, it increases by 1 each time through the loop

Flags

- A **flag** is a variable that keeps track of whether a certain situation has occurred.
- The data type most suited to flags is **Boolean**.

More About Flags

When *flagVar* is a variable of Boolean type, the statements

```
If flagVar = True Then
```

and

```
If flagVar = False Then
```

can be replaced by

```
If flagVar Then
```

and

```
If Not flagVar Then
```

Flags continued

The statements

```
  Do While flagVar = True
```

and

```
  Do While flagVar = False
```

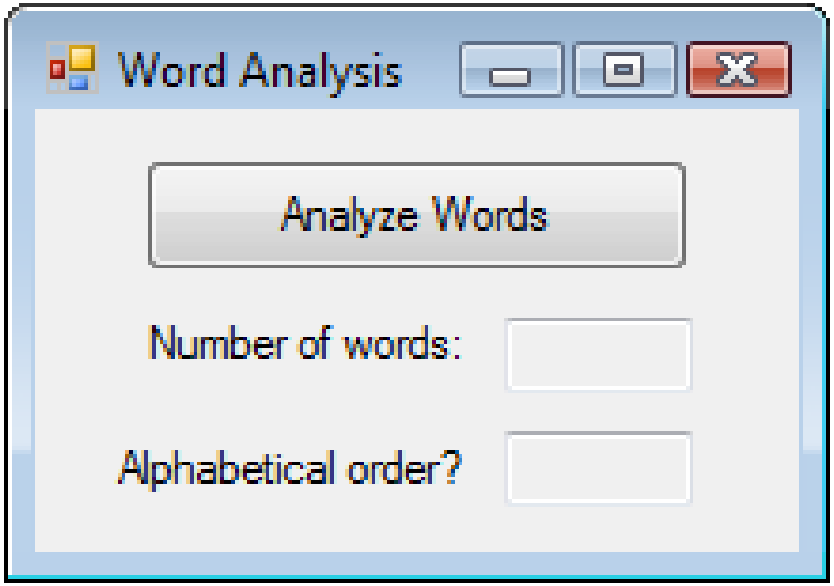
can be replaced by

```
  Do While flagVar
```

and

```
  Do While Not flagVar
```

Example 4: Form



The file WORDS.TXT contains words from a spelling bee, one word per line. Count the words and determine whether they are in alphabetical order.

Example 4: Partial Code

```
Dim word1 As String = ""
Dim orderFlag As Boolean = True
Do While (sr.Peek <> -1)
   word2 = sr.ReadLine
   wordCounter += 1
   If word1 > word2 Then
      orderFlag = False
   End If
   word1 = word2
Loop
```

Nested Loops

Statements inside a loop can contain another loop.

6.3 For…Next Loops

- Nested For … Next Loops
- Local Type Inference

For…Next Loops

- Used when we know how many times we want the loop to execute
- A counter controlled loop

Sample

```
For i As Integer = 1 To 5
  lstTable.Items.Add(i & " " & i ^ 2)
Next
```

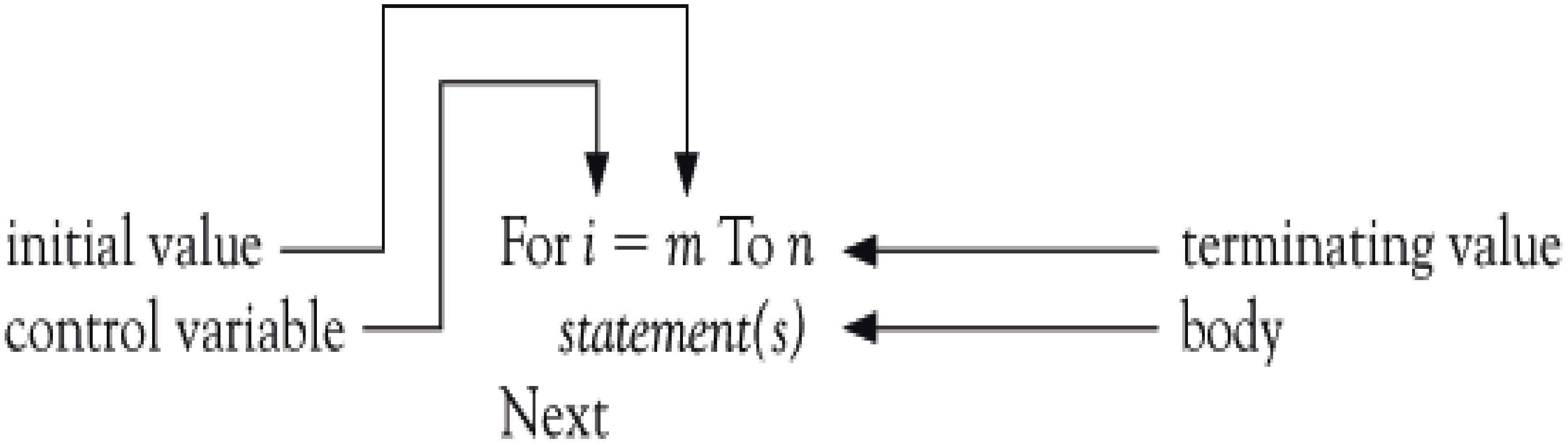
The loop control variable, i, is

- initialized to 1
- tested against the stop value, 5
- incremented by 1 at the Next statement

Similar Do While Loop

```
i = 1
Do While i <= 5
   lstTable.Items.Add(i & " " & i ^ 2)
   i += 1
Loop
```

For…Next Loop Syntax



initial value

control variable

For *i* = *m* To *n*

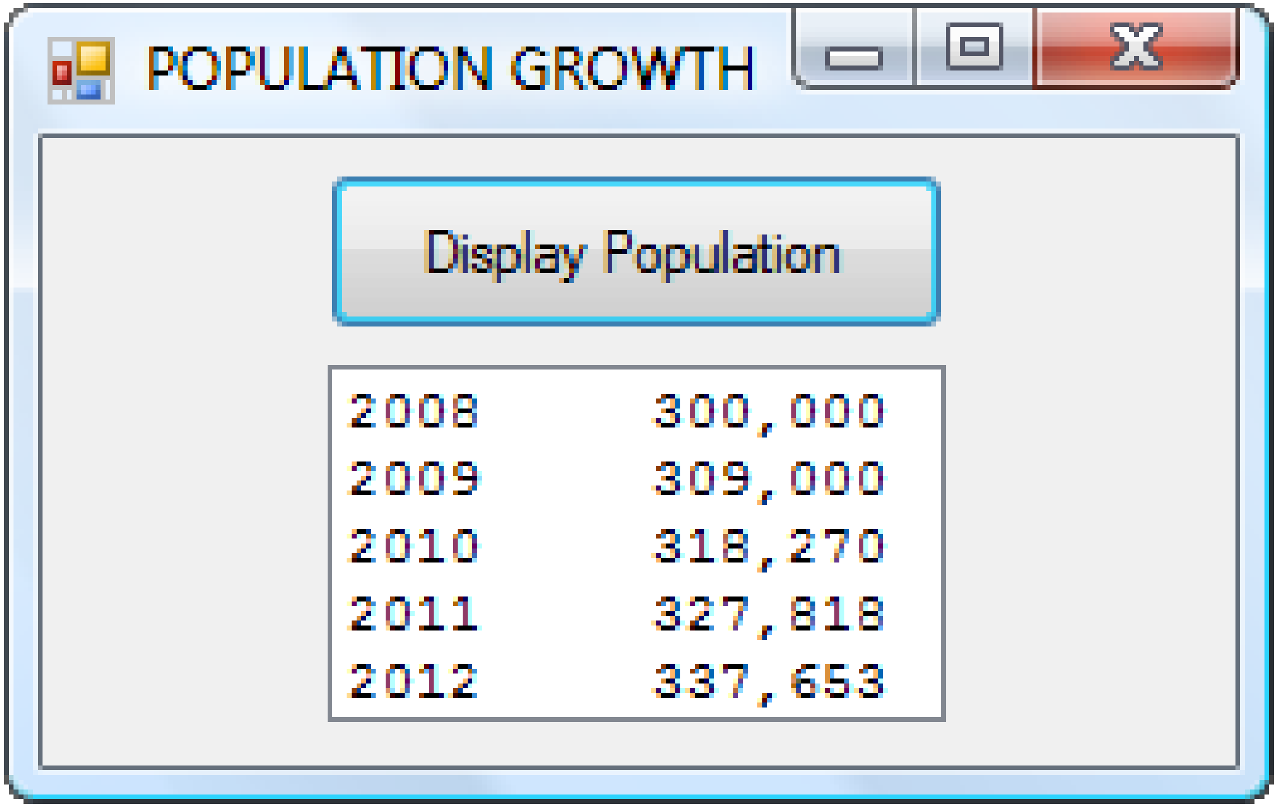
statement(s)

Next

terminating value

body

Example 1: Output

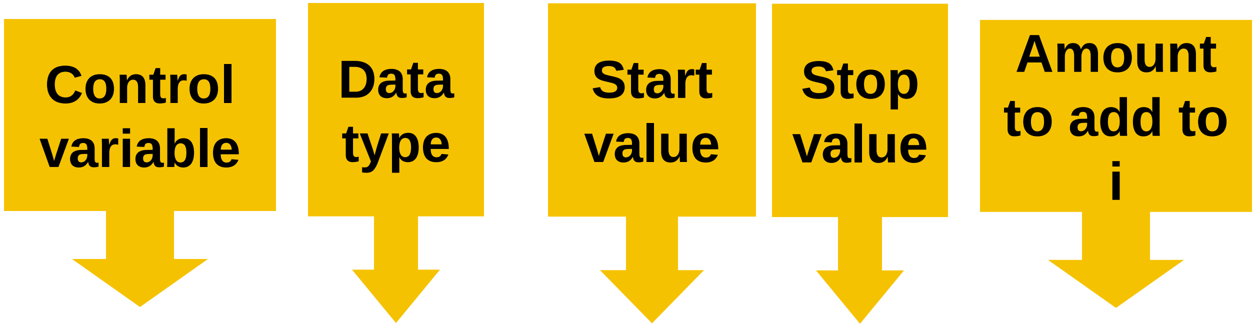


```
2008        300,000
2009        309,000
2010        318,270
2011        327,818
2012        337,653
```

Example 1: Code

```
Dim pop as Double = 300000
Dim fmtStr As String = "{0,4}{1,12:N0}"
For yr As Integer = 2008 To 2012
 lstPop.Items.Add(String.Format( _
                    fmtStr, yr, pop)
  pop += 0.03 * pop
Next
```

Example 2



Control variable

Data type

Start value

Stop value

Amount to add to i

```
For i As Integer = 0 To n Step s
  lstValues.Items.Add(i)
Next
```

Example with Negative Step

```
For j As Integer = 10 To 1 Step -1
   lstBox.Items.Add(j)
Next
lstBox.Items.Add("Blastoff")
```

Example: Nested Loops

Outer loop
Inner loop

```
For i As Integer = 65 To 70
   For j As Integer = 1 To 25
      lstBox.Items.Add(Chr(i) & j)
   Next
Next
```

OUTPUT:

```
A1
A2
A3
:
```

For and Next Pairs

- For and Next statements must be paired.
- If one is missing, the automatic syntax checker will complain with a wavy underline and a message such as

“A ‘For’ must be paired with a ‘Next’.”

Start, Stop, and Step values

- Consider a loop beginning with

  `For i As Integer = m To n Step s.`

- The loop will be executed exactly once if *m* equals *n* no matter what value *s* has.

- The loop will not be executed at all if *m* is greater than *n* and *s* is positive, or if *m* is less than *n* and *s* is negative.

Altering the Control Variable

- The value of the control variable should not be altered within the body of the loop.

- Doing so might cause the loop to repeat indefinitely or have an unpredictable number of repetitions.

Non-integer Step Values

- Can lead to round-off errors with the result that the loop is not executed the intended number of times.
- We will only use Integers for all values in the header.

Comments

```
For i As Integer = 1 To 1 Step 10
    (some statements)
Next
```

How many times of loops?

Comments

```
For i As Integer = 2 To 1 Step 2
    (some statements)
Next
```

How many times of loops?

Comments

```
For i As Integer = 1 To 5 Step -1
   (some statements)
Next
```

How many times of loops?

Comments

- The value of the control variable should not be altered within the body of the loop (For … Next).

- To skip an iteration in a For .. Next loop: Continue For

- To skip an iteration in a Do .. While loop:

  Continue Do

COMMENTS

```
For i As Integer = 1 To 5
   (some statements)
    Continue For
   (some statements)
Next
```

What will happen?

Comments

- To break out of a For .. Next loop:
  Exit For

- To break out of a Do .. While loop:
  Exit Do

Practice

- Why won’t the following lines of code work as intended?

```
For i As Integer = 15 To 1
    lstBox.Items.Add(i)
Next
```

Practice

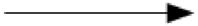
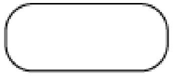
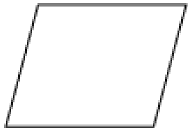
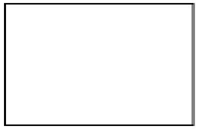
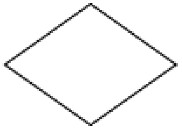
- When is a For … Next loop more appropriate than a Do loop?

Review

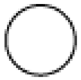
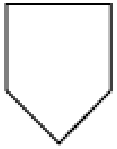
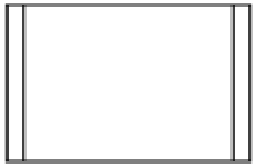
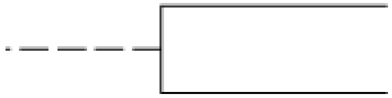
Performing a Task on the Computer

- Determine Output
- Identify Input
- Determine process necessary to turn given Input into desired Output

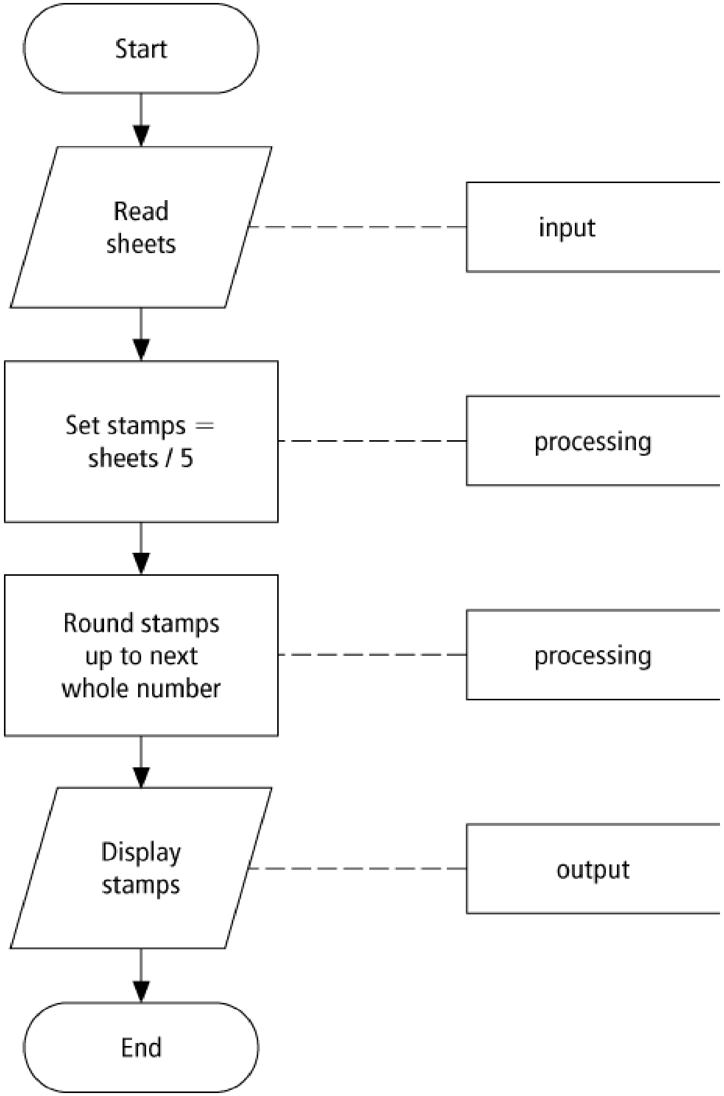
FLOWCHART SYMBOLS

Symbol	Name	Meaning
	*Flowline*	Used to connect symbols and indicate the flow of logic.
	*Terminal*	Used to represent the beginning (Start) or the end (End) of a task.
	*Input/Output*	Used for input and output operations, such as reading and displaying. The data to be read or displayed are described inside.
	*Processing*	Used for arithmetic and data-manipulation operations. The instructions are listed inside the symbol.
	*Decision*	Used for any logic or comparison operations. Unlike the input/ouput and processing symbols, which have one entry and one exit flowline, the decision symbol has one entry and two exit paths. The path chosen depends on whether the answer to a question is "yes" or "no."

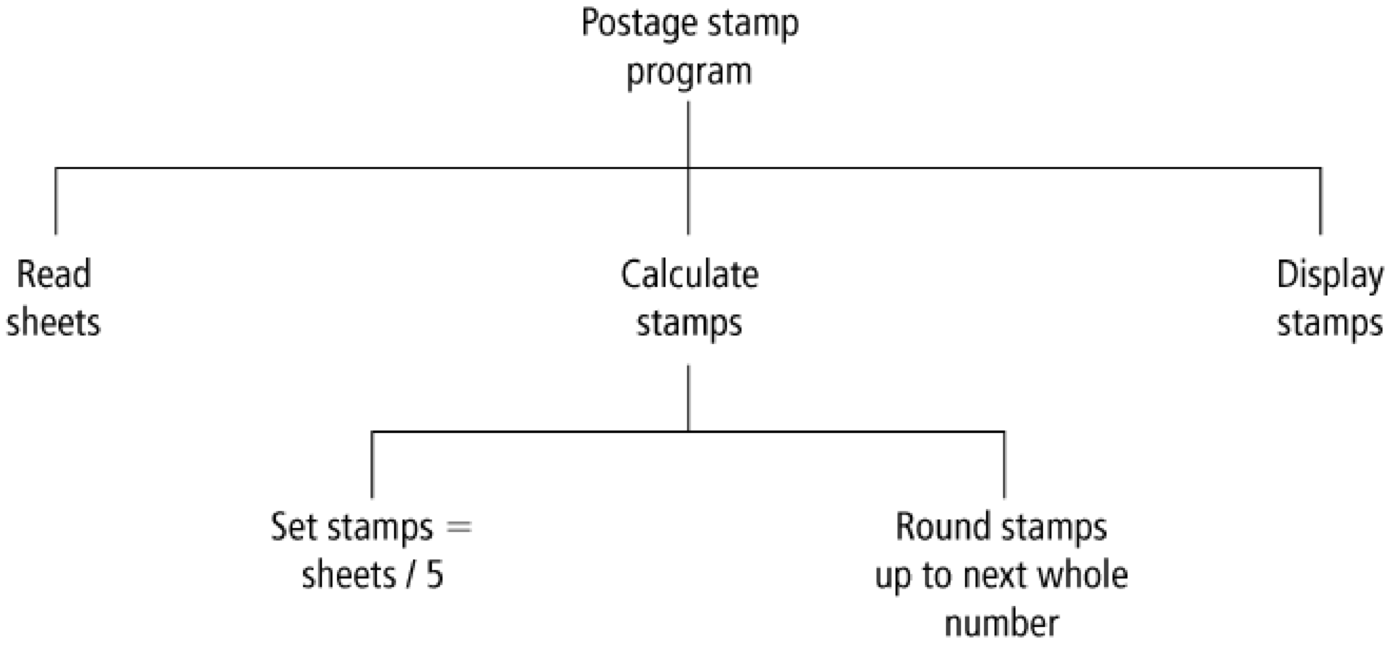
FLOWCHART SYMBOLS CONTINUED

	***Connector***	Used to join different flowlines.
	***Offpage Connector***	Used to indicate that the flowchart continues to a second page.
	***Predefined Process***	Used to represent a group of statements that perform one processing task.
	***Annotation***	Used to provide additional information about another flowchart symbol.

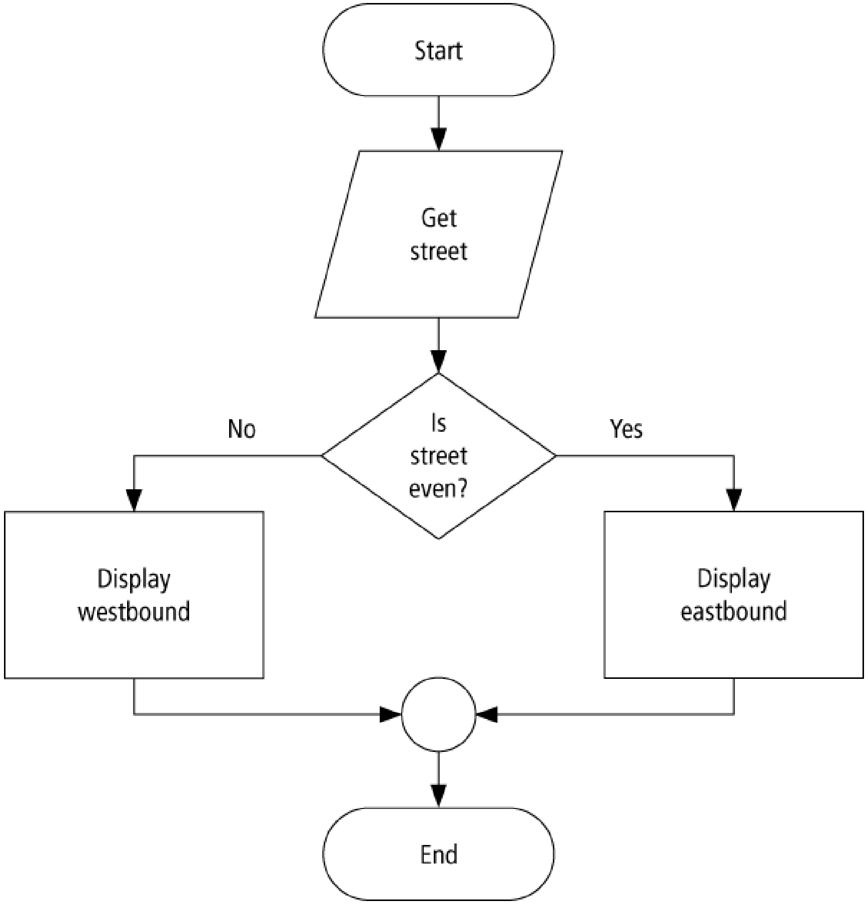
FLOWCHART EXAMPLE



HIERARCHY CHARTS EXAMPLE



FLOWCHART


Start
Get street
Is street even?
No
Yes
Display westbound
Display eastbound
End

Control Name Prefixes

Control	Prefix	Example
button	btn	btnCompute
label	lbl	lblAddress
text box	txt	txtAddress
list box	lst	lstOutput

VARIABLES

- Declaration:

```
Dim speed As Double
```

Variable name

Data type

- **Assignment:**

```
speed = 50
```

Variables

Visual Basic type	structure Storage size	Value range
Boolean	4 bytes	True or False
Byte	1 byte	0 to 255 (unsigned)
Char	2 bytes	0 to 65535 (unsigned)
Date	8 bytes	January 1, 1 CE to December 31, 9999
Decimal	12 bytes	+/-79,228,162,514,264,337,593,543,950,335 with no decimal point;
Double	8 bytes	-1.79769313486231E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values

VARIABLES

Visual Basic type	structure Storage size	Value range
Integer	4 bytes	-2,147,483,648 to 2,147,483,647
Long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Object	4 bytes	Any type can be stored in a variable of type Object
Short	2 bytes	-32,768 to 32,767
Single	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values
String	10 bytes + (2 * string length)	0 to approximately two billion Unicode characters

Some Types of Syntax Errors

- Misspellings

```
lstBox.Itms.Add(3)
```

- Omissions

```
lstBox.Items.Add(2 + )
```

- Incorrect punctuation

```
Dim m; n As Integer
```

Displayed as blue underline in VS

A Type of Run-time Error

```
Dim numVar As Integer = 1000000
numVar = numVar * numVar
```

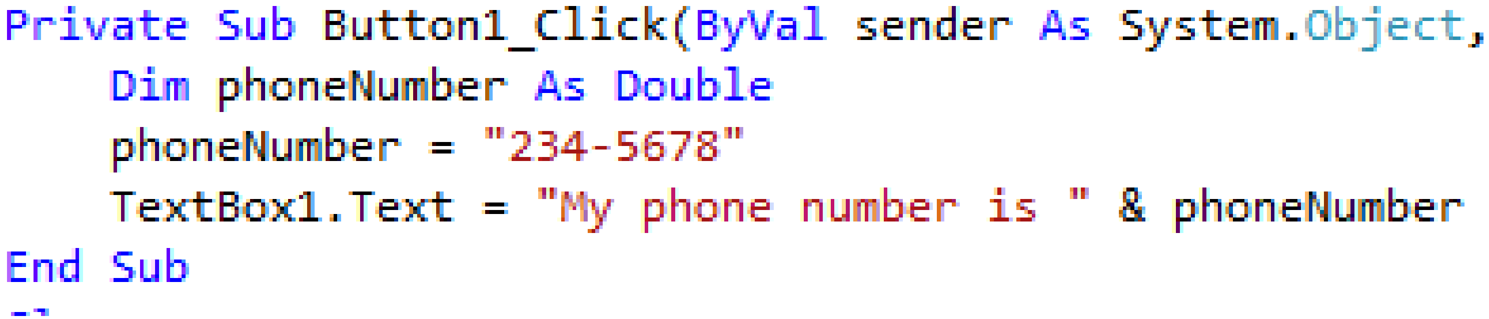
What’s wrong with the above?

A Logical Error

```
Dim average As Double
Dim m As Double = 5
Dim n As Double = 10
average = m + n / 2
```

What's wrong with the above?

What's wrong with this?



```
Private Sub Button1_Click(ByVal sender As System.Object,
    Dim phoneNumber As Double
    phoneNumber = "234-5678"
    TextBox1.Text = "My phone number is " & phoneNumber
End Sub
```

Is this allowed?

- Dim x as double = “23”

- dblVar = txtBox.text

- dblVar = 2 & 3

STRING VARIABLE

- Declaration:

```
Dim firstName As String
```

Variable name

Data type

- **Assignment:**

```
firstName = "Fred"
```

String Literal

A **string literal** is a sequence of characters surrounded by quotation marks. *Examples:*

Does this work?

```
“She said: “I’m tired.””
```

Substring Method

Let *str* be a string.
`str.Substring(`*`m`*`, `*`n`*`)` is the substring of length *n*, beginning at position *m* in *str*.

“Visual Basic”.Substring(2, 3) is “sua”
“Visual Basic”.Substring(0, 1) is “V”

Scope

- The **scope** of a variable is the portion of the program that can refer to it.

- Variables declared inside an event procedure are said to have **local scope** and are only available in the event procedure in which they are declared.

Scope

- Variables declared outside an event procedure are said to have **class-level scope** and are available to every event procedure.
- Usually declared after

  `Public Class` ***`formName`***

  (Declarations section of Code Editor.)

Example

When a = 3, b = 4

(a + b) < 2 * a **TRUE?**

Logical Operators

- Used with Boolean expressions
- *Not* – makes a False expression True and vice versa
- *And* – will yield a True if and only if both expressions are True
- *Or* – will yield a True if at least one of both expressions are True

If Block

The program will take a course of action based on whether a condition is true.

```
If condition Then
  action1
Else
  action2
End If
```

Will be executed if condition is true

Will be executed if condition is false

Sub Procedures

- Perform one or more related tasks
- General syntax

```
Sub ProcedureName()
  statements
End Sub
```

Calling a Sub Procedure

- The statement that invokes a Sub procedure is also referred to as a **Call statement**.
- A Call statement looks like this:

  ProcedureName()

EXAMPLE

```
Public Sub btnOne_Click (...) Handles _
                             btnOne.Click
  Dim n As Double = 19
  Triple(n)
  txtBox.Text = CStr(n)
End Sub

Sub Triple(ByVal num As Double)
  num = 3 * num
End Sub
```

What is output?

Example

```
Public Sub btnOne_Click (...) Handles _
                            btnOne.Click
  Dim num As Double = 4
  Triple(num)
  txtBox.Text = CStr(num)
End Sub

Sub Triple(ByRef num As Double)
  num = 3 * num
End Sub
```

What is output?

Named Constants

```
Const CONSTANT_NAME As DataType _
 = value
Ex)
Const PI As Double = 3.14
Dim num As Double = 4
```

Structured Programming

- Control structures in structured programming:
  - ***Sequences***: Statements are executed one after another.
  - ***Decisions***: One of two blocks of program code is executed based on a test for some condition.
  - ***Loops (iteration)***: One or more statements are executed repeatedly as long as a specified condition is true.