VB: Control Flow

Finding the path to a solution

Overview

- Branching
- Looping
- Procedures
- Exceptions

If Statements

```
If age <= 2 Then
    ServeMilk()
End If
```

```
If age <= 2 Then
    ServeMilk()
Else
    ServeSoda()
End If
```

```
If age <= 2 Then
    ServeMilk()
ElseIf age < 21 Then
    ServeSoda()
Else
    ServeDrink()
End If
```

```
If status = "PartTime" Then
    If age < 18 Then
        ' ...
    End If
End If
```

```
If status = "PartTime" AndAlso _
    age < 18 Then
' ...
End If
```

Case Statements

- Similar to If/ElseIf but cleaner when there are many options
- Restricted to integers, characters, strings, and enumerations

```
Select Case code
    Case 1
        ' ...
    Case Is < 10
        ' ...
    Case 11, 13, 15, 17, 19
        ' ...
    Case Else
        ' ...
End Select
```

For and While Loops

```
Dim max As Integer = 10

For i = 1 To max
    Console.WriteLine(i)
Next

For i = max To 1 Step -1
    Console.WriteLine(i)
Next
```

```
Dim index As Integer = 0

While index <= 10
    index += 1
    Console.WriteLine(index)
End While
```

Jumping

- The Exit keyword allows you to exit the loop at any point
- The Continue keyword allows you to skip to the next iteration

```
For Each age As Integer In ages
    If age = 2 Then
        Exit For
    End If
    ' ...
    If age > 21 Then
        Continue For
    End If
    ' ...
Next
```

For Each Loops

- Iterate an array or collection

```
Dim ages() As Integer = {2, 21, 40, 72, 100}

For Each age As Integer In ages
    Console.WriteLine(age)
Next
```

Procedures

- **Procedures enable you to:**
  - Divide large sections of code into a series of smaller ones
    - Easier to read and understand
  - Define commonly needed functionality in a single place
    - You can call the procedure from multiple places
  - Performing these tasks is commonly called *modularizing* your code
- **There are two types of procedures:**
  - Subroutines: do not return a value
  - Functions: return a value
- **May have parameters (arguments) passed to them**

Subroutines

- **In the declaration:**
  - Use the Sub keyword

```
Sub DisplayReverseString(value As String)
    For index As Integer = value.Length - 1 To 0 Step -1
        Console.Write(value(index))
    Next
    Console.WriteLine()
End Sub
```

Functions

- **In the declaration:**
  - Use the Function keyword
  - Indicate the return type
- **In the body:**
  - Use the Return keyword to indicate the result of the function

```
Function Factorial(ByVal n As Integer) _
    As Integer

    If n <= 1 Then
        Return 1
    Else
        Return Factorial(n - 1) * n
    End If
End Function
```

Overloading

- Define multiple functions or subroutines with the same name
- Compiler finds the best match by the number and type of the parameters

```
Sub Post(ByVal customerName As String, ByVal amount As Single)
    ' Insert code to access customer record by customer name.
End Sub

Sub Post(ByVal customerID As Integer, ByVal amount As Single)
    ' Insert code to access customer record by account number.
End Sub
```

```
Post("ALFKI", 100.0)
Post(42, 100.0)
```

Handling Exceptions

- **Handle exceptions using a Try/Catch block**
  - Runtime will search for the closest matching catch statement

```
Try
    CheckAges()
Catch ex As DivideByZeroException
    Console.WriteLine(ex.Message)
    Console.WriteLine(ex.StackTrace)
End Try
```

Chaining Catch Blocks

- Place most specific type in the first catch clause
- Catching a System.Exception catches everything
  - … except for a few "special" exceptions

```
Try
    CheckAges()
Catch ex1 As DivideByZeroException
    ' ...
Catch ex2 As ArithmeticException
    ' ...
Catch ex3 As Exception
    ' ...
End Try
```

Built-in Exceptions

- **Dozens of exceptions already defined in the FCL**
  - All derive from System.Exception

Type	Description
System.DivideByZeroException	Attempt to divide an integral value by zero occurs.
System.IndexOutOfRangeException	Attempt to index an array via an index that is outside the bounds of the array.
System.InvalidCastException	Thrown when an explicit conversion from a base type or interface to a derived type fails at run time.
System.NullReferenceException	Thrown when a null reference is used in a way that causes the referenced object to be required.
System.StackOverflowException	Thrown when the execution stack is exhausted by having too many pending method calls.
System.TypeInitializationException	Thrown when a static constructor throws an exception, and no catch clauses exists to catch it.

Throwing

- **Use throw to raise an exception**
  - Exceptions provide type safe and structured error handling in .NET
- **Runtime unwinds the stack until it finds a handler**
  - Exception may terminate an application

```
Sub BuyDrinks(age As Integer)
    If age < 21 Then
        Throw New ArgumentException("Must be 21 or older")
    End If
    ' ...
End Sub
```

Summary

- Flow control statements
  - Branching
  - Looping
- Procedures
  - Subroutines
  - Functions
- Exceptions
  - Exception handling
  - Throwing exceptions