Chapter 7 - Methods

Outline

Note: Inconsistent with textbook subsection numbering

[...]  
7.13 Recursion  
[...]

© 2002 Prentice Hall. All rights reserved.
7.13 Recursion

• Recursive methods
  – Methods that call themselves
    • Directly
    • Indirectly
      – Call others methods which call it
  – Continually breaks problem down to simpler forms
  – Must converge in order to end recursion
  – Each method call remains open (unfinished)
    • Finishes each call and then finishes itself
7.13 Recursion

(a) Procession of recursive calls. (b) Values returned from each recursive call.

Cf. Fig. 7.16  Recursive evaluation of 5!.

Final value = 120

5! = 5 * 24 = 120 is returned
4! = 4 * 6 = 24 is returned
3! = 3 * 2 = 6 is returned
2! = 2 * 1 = 2 is returned
1 returned
// Fig. 6.15: FactorialTest.cs [prev. textbook edition]
// cf. Fig. 7.17 for a simplified version (with console output)
// Recursive Factorial method.

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

public class FactorialTest : System.Windows.Forms.Form
{
    private System.ComponentModel.Container components = null;

    private System.Windows.Forms.Label outputLabel;

    public FactorialTest()
    {
        InitializeComponent();

        for (long i = 0; i <= 10; i++)
            outputLabel.Text += i + "! = " + Factorial(i) + "\n";
    }
}

// Recursive Factorial method.
long Factorial(long n)
{
    if (n == 0) return 1;
    else return n * Factorial(n - 1);
}
```csharp
// Visual Studio .NET-generated code
public long Factorial(long number)
{
    if (number <= 1) // base case
        return 1;
    else
        return number * Factorial(number - 1);
}

[STAThread]
static void Main()
{
    Application.Run(new FactorialTest());
}

} // end of class FactorialTest
```

The Factorial method calls itself (recursion)

The recursion ends when the value is less than or equal to 1

Program Output

0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
6.15 Example Using Recursion: The Fibonacci Sequence

• Fibonacci Sequence
  – \( F(0) = 0 \)
  – \( F(1) = 1 \)
  – \( F(n) = F(n - 1) + F(n - 2) \)
  – Recursion is used to evaluate \( F(n) \)

• Complexity theory
  – How hard computers need to work to perform algorithms
Recall:
F(0) = 0
F(1) = 1
F(n) = F(n - 1) + F(n - 2)

Fig. 6.17  Set of recursive calls to method Fibonacci (abbreviated as F).
GUI for Recursive Fibonacci Method

Enter an Integer: 22

Fibonacci Value is 17711
public class FibonacciTest : System.Windows.Forms.Form
{
    private System.ComponentModel.Container components = null;
    private System.Windows.Forms.TextBox inputTextBox;
    private System.Windows.Forms.Label displayLabel;
    private System.Windows.Forms.Label promptLabel;
    public FibonacciTest()
    {
        InitializeComponent();
    }

    // Visual Studio .NET-generated code
// call Fibonacci and display results
protected void calculateButton_Click(
    object sender, System.EventArgs e )
{
    string numberString = ( inputTextBox.Text );
    int number = System.Convert.ToInt32( numberString );
    int fibonacciNumber = Fibonacci( number );
    displayLabel.Text = "Fibonacci Value is " + fibonacciNumber;
}

// calculates Fibonacci number
public int Fibonacci( int number )
{
    if ( number == 0 || number == 1 )
        return number;
    else
        return Fibonacci( number - 1 ) + Fibonacci( number - 2 );
}

[STAThread]
static void Main()
{
    Application.Run( new FibonacciTest() );
}

} // end of class FibonacciTest

The number uses the Fibonacci method to get its result

The recursion ends when the number is 0 or 1

Calls itself twice, to get the result of the the two previous numbers
FibonacciTest.cs

Program Output

Enter an Integer: 22

Fibonacci Value is 17711

Calculate Fibonacci
6.16 Recursion vs. Iteration

• Iteration
  – Uses repetition structures
    • *while, do/while, for, foreach*
  – Continues until counter fails repetition case

• Recursion
  – Uses selection structures
    • *if, if/else, switch*
  – Repetition through method calls
  – Continues until a base case (e.g., F(0) or F(1)) is reached
  – Creates a duplicate of the variables
    • Can consume memory and processor speed
THE END