

Chapter 4

How to work with numeric and string data

Objectives

Applied

1. Given an arithmetic expression and the values for the variables in the expression, evaluate the expression.
2. Use numeric and string data as needed within your applications. That means you should be able to do any of the following:
 - declare and initialize variables and constants
 - code arithmetic expressions and assignment statements
 - use the static methods of the Math class
 - cast and convert data from one type to another
 - use the correct scope for your variables
 - declare and use enumerations
 - work with nullable types and the null-coalescing operator

Objectives (cont.)

Knowledge

1. Distinguish between a variable and a constant and give the naming conventions for each.
2. Describe any of these data types: int, double, decimal, bool, and string.
3. Describe any of these terms: literal value, null value, empty string, concatenate, append, escape sequence, string literal, verbatim string literal, and nullable data type.
4. Describe the order of precedence for arithmetic expressions.
5. Distinguish between implicit casting and explicit casting.
6. Distinguish between a value type and a reference type.

Objectives (cont.)

7. Describe the differences in the ways that casting, the ToString method of a data structure, the Parse and TryParse methods of a data structure, and the methods of the Convert class can be used to convert data from one form to another.
8. Describe the differences between class scope and method scope.

The built-in value types

C# keyword	Bytes	.NET type	Description
byte	1	Byte	A positive integer value from 0 to 255
sbyte	1	SByte	A signed integer value from -128 to 127
short	2	Int16	An integer from -32,768 to +32,767
ushort	2	UInt16	An unsigned integer from 0 to 65,535
int	4	Int32	An integer from -2,147,483,648 to +2,147,483,647
uint	4	UInt32	An unsigned integer from 0 to 4,294,967,295

The built-in value types (cont.)

C# keyword	Bytes	.NET type	Description
long	8	Int64	An integer from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
ulong	8	UInt64	An unsigned integer from 0 to +18,446,744,073,709,551,615
float	4	Single	A non-integer number with approximately 7 significant digits
double	8	Double	A non-integer number with approximately 14 significant digits

The built-in value types (cont.)

C# keyword	Bytes	.NET type	Description
decimal	16	Decimal	A non-integer number with up to 28 significant digits (integer and fraction) that can represent values up to 7.9228×10^{28}
char	2	Char	A single Unicode character
bool	1	Boolean	A true or false value

How to declare and initialize a variable in two statements

Syntax

```
type variableName;  
variableName = value;
```

Example

```
int counter;  
counter = 1;
```

Description

- A *variable* stores a value that can change as a program executes.

Naming convention

- Start the names of variables with a lowercase letter, and capitalize the first letter of each word after the first word. This is known as *camel notation*.

How to declare and initialize a variable in one statement

Syntax

```
type variableName = value;
```

Examples

```
int counter = 1;  
long numberOfBytes = 20000;  
float interestRate = 8.125f;  
double price = 14.95;  
decimal total = 24218.1928m;  
double starCount = 3.65e+9;  
char letter = 'A';  
bool valid = false;  
int x = 0, y = 0;
```

How to declare and initialize a constant

Syntax

```
const type ConstantName = value;
```

Examples

```
const int DaysInNovember = 30;  
const decimal SalesTax = .075m;
```

Description

- A *constant* stores a value that can't be changed.

Naming convention

- Capitalize the first letter of each word of a constant name. This is known as *Pascal notation*.

Arithmetic operators

Operator	Name	Description
+	Addition	Adds two operands.
-	Subtraction	Subtracts the right operand from the left operand.
*	Multiplication	Multiplies the right operand and the left operand.
/	Division	Divides the right operand into the left operand. If both operands are integers, the result is an integer.
%	Modulus	Returns the value that is left over after dividing the right operand into the left operand.

Arithmetic operators (cont.)

Operator	Name	Description
+	Positive sign	Returns the value of the operand.
-	Negative sign	Changes a positive value to negative, and vice versa.
++	Increment	Adds 1 to the operand ($x = x + 1$).
--	Decrement	Subtracts 1 from the operand ($x = x - 1$).

Arithmetic expressions that use integers

```
int x = 14;
int y = 8;
int result1 = x + y;           // result1 = 22
int result2 = x - y;           // result2 = 6
int result3 = x * y;           // result3 = 112
int result4 = x / y;           // result4 = 1
int result5 = x % y;           // result5 = 6
int result6 = -y + x;          // result6 = 8
int result7 = --y;             // result7 = 7, y = 7
int result8 = ++x;             // result8 = 15, x = 15
```

Arithmetic expressions that use decimals

```
decimal a = 8.5m;  
decimal b = 3.4m;  
decimal result11 = a + b;           // result11 = 11.9  
decimal result12 = a - b;           // result12 = 5.1  
decimal result13 = a / b;           // result13 = 2.5  
decimal result14 = a * b;           // result14 = 28.90  
decimal result15 = a % b;           // result15 = 1.7  
decimal result16 = -a;               // result16 = -8.5  
decimal result17 = --a;              // result17 = 7.5, a = 7.5  
decimal result18 = ++b;              // result18 = 4.4, b = 4.4
```

Assignment operators

Operator	Name
=	Assignment
+=	Addition
-=	Subtraction
*=	Multiplication
/=	Division
%=	Modulus

The syntax for a simple assignment statement

```
variableName = expression;
```

Typical assignment statements

```
counter = 7;  
newCounter = counter;  
discountAmount = subtotal * .2m;  
total = subtotal - discountAmount;
```


Statements that use the same variable on both sides of the equals sign

```
total = total + 100m;  
total = total - 100m;  
price = price * .8m;
```

Statements that use the shortcut assignment operators

```
total += 100m;  
total -= 100m;  
price *= .8m;
```

The order of precedence for arithmetic operations

1. Increment and decrement
2. Positive and negative
3. Multiplication, division, and modulus
4. Addition and subtraction

A calculation that uses the default order of precedence

```
decimal discountPercent = .2m;           // 20% discount
decimal price = 100m;                     // $100 price
price = price * 1 - discountPercent;      // price = $99.8
```

A calculation that uses parentheses to specify the order of precedence

```
decimal discountPercent = .2m;           // 20% discount
decimal price = 100m;                     // $100 price
price = price * (1 - discountPercent);    // price = $80
```

The use of prefixed and postfix increment and decrement operators

```
int a = 5;
int b = 5;
int y = ++a;           // a = 6, y = 6
int z = b++;           // b = 6, z = 5
```

How implicit casting works

Casting from less precise to more precise data types

byte→short→int→long→decimal

int→double

short→float→double

char→int

Examples

```
double grade = 93;           // convert int to double

int letter = 'A';           // convert char to int

double a = 95.0;
int b = 86, c = 91;
double average = (a+b+c)/3;  // convert b and c to doubles
                             // (average = 90.666666...)
```

How to code an explicit cast

The syntax for coding an explicit cast

`(type) expression`

Examples

```
int grade = (int)93.75;           // convert double to int
                                   // (grade = 93)
```

```
char letter = (char)65;           // convert int to char
                                   // (letter = 'A')
```

```
double a = 95.0;
int b = 86, c = 91;
int average = ((int)a+b+c)/3;      // convert a to int value
                                   // (average = 90)
```

```
decimal result = (decimal)b/(decimal)c; // result has decimal
                                           // places
```

Five static methods of the Math class

The syntax of the Round method

`Math.Round(decimalNumber[, precision[, mode]])`

The syntax of the Pow method

`Math.Pow(number, power)`

The syntax of the Sqrt method

`Math.Sqrt(number)`

The syntax of the Min and Max methods

`Math.{Min|Max}(number1, number2)`

Statements that use static methods of the Math class

```
int shipWeight = Math.Round(shipWeightDouble);  
    // round to a whole number
```

```
double orderTotal = Math.Round(orderTotal, 2);  
    // round to 2 decimal places
```

```
double area = Math.Pow(radius, 2) * Math.PI;  
    // area of circle
```

```
double sqrtX = Math.Sqrt(x);
```

```
double maxSales = Math.Max(lastYearSales, thisYearSales);
```

```
int minQty = Math.Min(lastYearQty, thisYearQty);
```

Results from static methods of the Math class

Statement	Result
<code>Math.Round(23.75, 1)</code>	23.8
<code>Math.Round(23.85, 1)</code>	23.8
<code>Math.Round(23.744, 2)</code>	23.74
<code>Math.Round(23.745, 2)</code>	23.74
<code>Math.Round(23.745, 2, MidpointRounding.AwayFromZero)</code>	23.75
<code>Math.Pow(5, 2)</code>	25
<code>Math.Sqrt(20.25)</code>	4.5
<code>Math.Max(23.75, 20.25)</code>	23.75
<code>Math.Min(23.75, 20.25)</code>	20.25

How to declare and initialize a string

```
string message1 = "Invalid data entry.";
string message2 = "";
string message3 = null;
```

How to join strings

```
string firstName = "Bob";           // firstName is "Bob"
string lastName = "Smith";          // lastName is "Smith"
string name = firstName + " " + lastName;
                                     // name is "Bob Smith"
```

How to join a string and a number

```
double price = 14.95;
string priceString = "Price: $" + price;
// priceString is "Price: $14.95"
```

How to append one string to another string

```
string firstName = "Bob";           // firstName is "Bob"  
string lastName = "Smith";         // lastName is "Smith"  
string name = firstName + " ";     // name is "Bob "  
name = name + lastName;            // name is "Bob Smith"
```

How to append one string to another with the += operator

```
string firstName = "Bob";           // firstName is "Bob"  
string lastName = "Smith";         // lastName is "Smith"  
string name = firstName + " ";     // name is "Bob "  
name += lastName;                  // name is "Bob Smith"
```

Common escape sequences

Key	Description
<code>\n</code>	New line
<code>\t</code>	Tab
<code>\r</code>	Return
<code>\\</code>	Backslash
<code>\"</code>	Quotation

Examples that use escape sequences

Code	Result
<pre>string code = "JSPS"; decimal price = 49.50m; string result = "Code: " + code + "\n" + "Price: \$" + price + "\n";</pre>	<pre>Code: JSPS Price: \$49.50</pre>
<pre>string names = Joe "Joe\tSmith\rKate\tLewis\r";</pre>	<pre>Smith Kate Lewis</pre>
<pre>string path = "c:\\c#.net\\files";</pre>	<pre>c:\c#.net\files</pre>
<pre>string message = "Type \"x\" to exit";</pre>	<pre>Type "x" to exit</pre>

Examples that use verbatim string literals

Code	Result
<code>string names = @"Joe Smith Kate Lewis";</code>	<code>Joe Smith Kate Lewis</code>
<code>string path = @"c:\c#.net\files";</code>	<code>c:\c#.net\files</code>
<code>string message = @"Type ""x"" to exit";</code>	<code>Type "x" to exit</code>

Common .NET structures that define value types

Structure	C# keyword	What the value type holds
Byte	byte	An 8-bit unsigned integer
Int16	short	A 16-bit signed integer
Int32	int	A 32-bit signed integer
Int64	long	A 64-bit signed integer
Single	float	A single-precision floating-point number
Double	double	A double-precision floating-point number
Decimal	decimal	A 96-bit decimal value
Boolean	bool	A true or false value
Char	char	A single character

Common .NET classes that define reference types

Class	C# keyword	What the reference type holds
String	<code>string</code>	A reference to a String object
Object	<code>object</code>	A reference to any type of object

Common methods for data conversion

Method	Description
ToString ([format])	A method that converts the value to its equivalent string representation using the specified format. If the format is omitted, the value isn't formatted.
Parse (string)	A static method that converts the specified string to an equivalent data value. If the string can't be converted, an exception occurs.
TryParse (string, result)	A static method that converts the specified string to an equivalent data value and stores it in the result variable. Returns a true value if the string is converted. Otherwise, returns a false value.

Some of the static methods of the Convert class

Method	Description
<code>ToDecimal(value)</code>	Converts the value to the decimal data type.
<code>ToDouble(value)</code>	Converts the value to the double data type.
<code>ToInt32(value)</code>	Converts the value to the int data type.
<code>ToChar(value)</code>	Converts the value to the char data type.
<code>ToBool(value)</code>	Converts the value to the bool data type.
<code>ToString(value)</code>	Converts the value to a string object.

Statements that use ToString, Parse, and TryParse

```
decimal sales = 2574.98m;  
string salesString = sales.ToString();           // decimal to string  
sales = Decimal.Parse(salesString);              // string to decimal  
Decimal.TryParse(salesString, out sales);         // string to decimal
```

An implicit call of the ToString method

```
double price = 49.50;  
string priceString = "Price: $" + price;         // automatic ToString  
call
```

A TryParse method that handles invalid data

```
string salesString = "$2574.98";  
decimal sales = 0m;  
Decimal.TryParse(salesString, out sales);         // sales is 0
```

Conversion statements that use the Convert class

```
decimal subtotal = Convert.ToDecimal(txtSubtotal.Text);  
    // string to decimal  
int years = Convert.ToInt32(txtYears.Text);  
    // string to int  
txtSubtotal.Text = Convert.ToString(subtotal);  
    // decimal to string  
int subtotalInt = Convert.ToInt32(subtotal);  
    // decimal to int
```

Standard numeric formatting codes

Code	Format
C or c	Currency
P or p	Percent
N or n	Number
F or f	Float
D or d	Digits
E or e	Exponential
G or g	General

How to use the ToString method to format a number

Statement	Example
<code>string monthlyAmount = amount.ToString("c");</code>	\$1,547.20
<code>string interestRate = interest.ToString("p1");</code>	2.3%
<code>string quantityString = quantity.ToString("n0");</code>	15,000
<code>string paymentString = payment.ToString("f3");</code>	432.818

How to use the Format method of the String class to format a number

Statement	Result
<code>string monthlyAmount = String.Format("{0:c}", 1547.2m);</code>	\$1,547.20
<code>string interestRate = String.Format("{0:p1}", .023m);</code>	2.3%
<code>string quantityString = String.Format("{0:n0}", 15000);</code>	15,000
<code>string paymentString = String.Format("{0:f3}", 432.8175);</code>	432.818

The syntax of the format specification used by the Format method

`{index:formatCode}`

Code that declares and uses variables with class scope

```
public frmInvoiceTotal()  
{  
    InitializeComponent();  
}
```

```
decimal numberOfInvoices = 0m;  
decimal totalOfInvoices = 0m;
```

```
private void btnCalculate_Click(object sender, EventArgs e)  
{  
    decimal subtotal = Convert.ToDecimal(txtEnterSubtotal.Text);  
    decimal discountPercent = .25m;  
    decimal discountAmount = subtotal * discountPercent;  
    decimal invoiceTotal = subtotal - discountAmount;  
  
    numberOfInvoices++;  
    totalOfInvoices += invoiceTotal;  
  
    // the rest of the code for the method  
}
```

```
private void btnClearTotals_Click(object sender, EventArgs e)  
{  
    numberOfInvoices = 0m;  
    totalOfInvoices = 0m;  
}
```

How to work with scope

- The *scope* of a variable determines what code has access to it. If you try to refer to a variable outside of its scope, it will cause a build error.
- The scope of a variable is determined by where you declare it. If you declare a variable within a method, it has *method scope*. If you declare a variable within a class but not within a method, it has *class scope*.
- A variable with method scope can only be referred to by statements within that method. A variable with class scope can be referred to by all of the methods in the class.
- The *lifetime* of a variable is the period of time that it's available for use. A variable with method scope is only available while the method is executing. A variable with class scope is available while the class is instantiated.

Some of the constants in the `FormBorderStyle` enumeration

Constant	Description
<code>FormBorderStyle.FixedDialog</code>	A fixed, thick border typically used for dialog boxes.
<code>FormBorderStyle.FixedSingle</code>	A single-line border that isn't resizable.
<code>FormBorderStyle.Sizable</code>	A resizable border

A statement that uses the `FormBorderStyle` enumeration

```
this.FormBorderStyle = FormBorderStyle.FixedSingle;
```

The syntax for declaring an enumeration

```
enum EnumerationName [: type]
{
    ConstantName1 [= value][,
    ConstantName2 [= value]]...
}
```

An enumeration that sets the constant values to 0, 1, and 2

```
enum Terms
{
    Net30Days,
    Net60Days,
    Net90Days
}
```

An enumeration that sets the constant values to 30, 60, and 90

```
enum TermValues : short
{
    Net30Days = 30,
    Net60Days = 60,
    Net90Days = 90
}
```

Statements that use these enumerations

```
Terms t = Terms.Net30Days;  
int i = (int) Terms.Net30Days;           // i is 0  
int i = (int) TermValues.Net60Days;      // i is 60  
string s = Terms.Net30Days.ToString();    // s is "Net30Days"
```

How to declare a value type that can contain null values

```
int? quantity;  
quantity = null;  
quantity = 0;  
quantity = 20;
```

```
decimal? salesTotal = null;
```

```
bool? isValid = null;
```

```
Terms? paymentTerm = null;
```

```
// string? message = null; // not necessary or allowed
```

Two properties for working with nullable types

Property	Description
HasValue	Returns a true value if the nullable type contains a value. Returns a false value if the nullable type is null.
Value	Returns the value of the nullable type.

How to check if a nullable type contains a value

```
bool hasValue = quantity.HasValue;
```

How to get the value of a nullable type

```
int qty = quantity.Value;
```

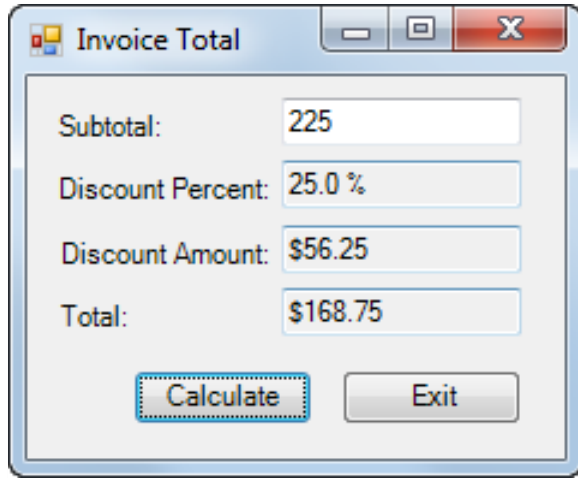
How to use the null-coalescing operator to assign a default value

```
int qty = quantity ?? -1;
```

How to use nullable types in arithmetic expressions

```
decimal? sales1 = 3267.58m;  
decimal? sales2 = null;  
decimal? salesTotal = sales1 + sales2; // result = null
```

The Invoice Total form



The screenshot shows a Windows-style dialog box titled "Invoice Total". It contains four text input fields with the following labels and values: "Subtotal:" with "225", "Discount Percent:" with "25.0 %", "Discount Amount:" with "\$56.25", and "Total:" with "\$168.75". At the bottom, there are two buttons: "Calculate" (which is highlighted with a blue dashed border) and "Exit".

Field	Value
Subtotal:	225
Discount Percent:	25.0 %
Discount Amount:	\$56.25
Total:	\$168.75

The controls that are referred to in the code

Object type	Name	Description
TextBox	txtSubtotal	Accepts a subtotal amount
TextBox	txtDiscountPercent	Displays the discount percent
TextBox	txtDiscountAmount	Displays the discount amount
TextBox	txtTotal	Displays the invoice total
Button	btnCalculate	Calculates the discount amount and invoice total when clicked
Button	btnExit	Closes the form when clicked

The event handlers for the Invoice Total form

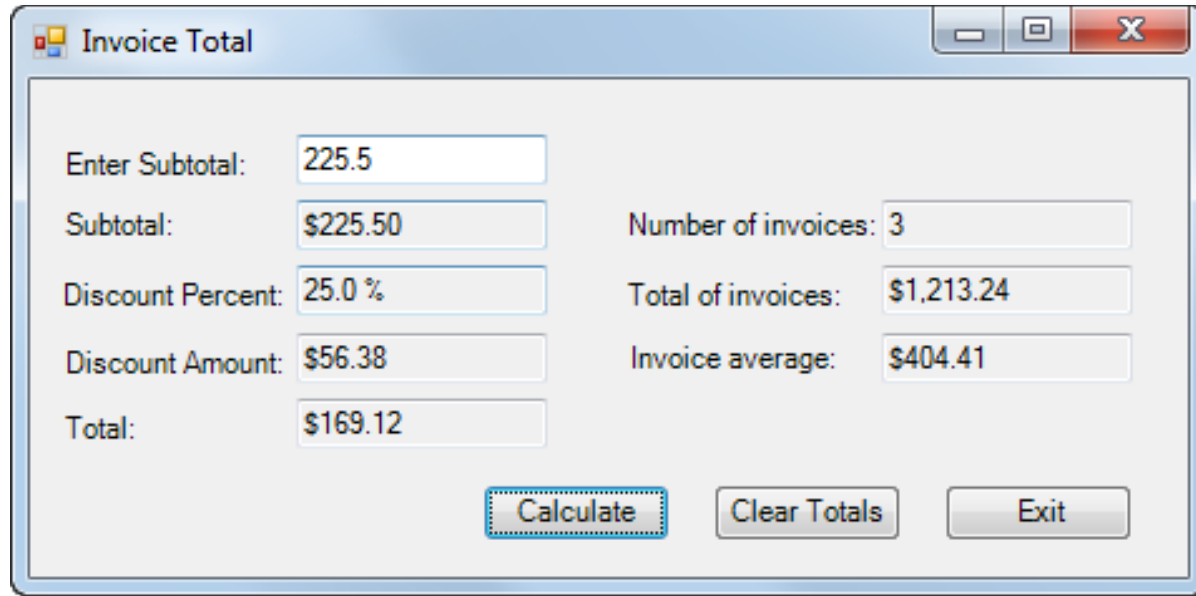
```
private void btnCalculate_Click(object sender, EventArgs e)
{
    decimal subtotal = Convert.ToDecimal(txtSubtotal.Text);
    decimal discountPercent = .25m;
    decimal discountAmount = subtotal * discountPercent;
    decimal invoiceTotal = subtotal - discountAmount;

    txtDiscountPercent.Text = discountPercent.ToString("p1");
    txtDiscountAmount.Text = discountAmount.ToString("c");
    txtTotal.Text = invoiceTotal.ToString("c");

    txtSubtotal.Focus();
}

private void btnExit_Click(object sender, EventArgs e)
{
    this.Close();
}
```

The enhanced Invoice Total form



The screenshot shows a Windows-style application window titled "Invoice Total". Inside the window, there are several input fields and calculated values arranged in two columns. The left column contains: "Enter Subtotal:" with a text box containing "225.5"; "Subtotal:" with a text box containing "\$225.50"; "Discount Percent:" with a text box containing "25.0 %"; "Discount Amount:" with a text box containing "\$56.38"; and "Total:" with a text box containing "\$169.12". The right column contains: "Number of invoices:" with a text box containing "3"; "Total of invoices:" with a text box containing "\$1,213.24"; and "Invoice average:" with a text box containing "\$404.41". At the bottom of the window, there are three buttons: "Calculate" (highlighted with a dashed border), "Clear Totals", and "Exit".

Field	Value
Enter Subtotal:	225.5
Subtotal:	\$225.50
Discount Percent:	25.0 %
Discount Amount:	\$56.38
Total:	\$169.12
Number of invoices:	3
Total of invoices:	\$1,213.24
Invoice average:	\$404.41

Buttons: Calculate, Clear Totals, Exit

The code for the class variables and two event handlers

```
int numberOfInvoices = 0;
decimal totalOfInvoices = 0m;
decimal invoiceAverage = 0m;

private void btnCalculate_Click(object sender, EventArgs e)
{
    decimal subtotal = Convert.ToDecimal(txtEnterSubtotal.Text);
    decimal discountPercent = .25m;
    decimal discountAmount = Math.Round(subtotal * discountPercent, 2);
    decimal invoiceTotal = subtotal - discountAmount;

    txtSubtotal.Text = subtotal.ToString("c");
    txtDiscountPercent.Text = discountPercent.ToString("p1");
    txtDiscountAmount.Text = discountAmount.ToString("c");
    txtTotal.Text = invoiceTotal.ToString("c");

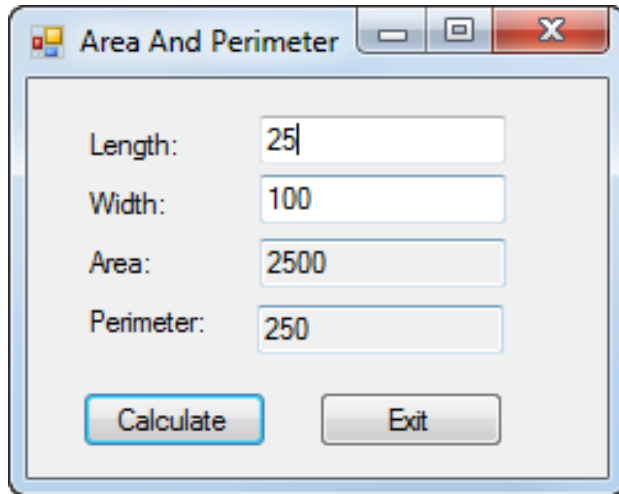
    numberOfInvoices++;
    totalOfInvoices += invoiceTotal;
    invoiceAverage = totalOfInvoices / numberOfInvoices;

    txtNumberOfInvoices.Text = numberOfInvoices.ToString();
    txtTotalOfInvoices.Text = totalOfInvoices.ToString("c");
    txtInvoiceAverage.Text = invoiceAverage.ToString("c");
}
```

The class variables and event handlers (cont.)

```
txtEnterSubtotal.Text = "";  
    txtEnterSubtotal.Focus();  
}  
  
private void btnClearTotals_Click(object sender, System.EventArgs e)  
{  
    numberOfInvoices = 0;  
    totalOfInvoices = 0m;  
    invoiceAverage = 0m;  
  
    txtNumberOfInvoices.Text = "";  
    txtTotalOfInvoices.Text = "";  
    txtInvoiceAverage.Text = "";  
  
    txtEnterSubtotal.Focus();  
}
```

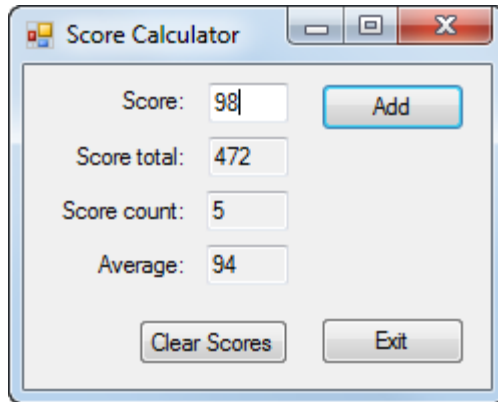
Extra 4-1 Calculate area and perimeter



A screenshot of a Windows application window titled "Area And Perimeter". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. Inside the window, there are four text input fields arranged vertically. The first field is labeled "Length:" and contains the value "25". The second field is labeled "Width:" and contains the value "100". The third field is labeled "Area:" and contains the value "2500". The fourth field is labeled "Perimeter:" and contains the value "250". At the bottom of the window, there are two buttons: a blue "Calculate" button and a gray "Exit" button.

Accept the length and width of a rectangle and calculate the area and perimeter.

Extra 4-2 Accumulate test score data



The screenshot shows a Windows-style application window titled "Score Calculator". It contains four text input fields: "Score:" with the value "98", "Score total:" with "472", "Score count:" with "5", and "Average:" with "94". To the right of the "Score:" field is a blue "Add" button. At the bottom of the window are two buttons: "Clear Scores" and "Exit".

Accept one or more scores and calculate the score total, score count, and average score.