Chapter 5

How to code control structures

Objectives

Applied

1. Given a Boolean expression and the values for the variables in the expression, evaluate the expression.
2. Use if statements and switch statements as needed within your applications.
3. Use while, do-while, and for loops as needed within your applications.

Knowledge

1. Compare the if-else and switch statements.
2. Describe the differences between while, do-while, and for loops.
3. Explain how the break and continue statements work when used within a loop.

Objectives (cont.)

4. Describe the use of breakpoints, the Locals window, and stepping through code as they apply to debugging errors within loops.
5. Describe block scope in the context of the control structures.

Relational operators

Operator	Name
`==`	Equality
`!=`	Inequality
`>`	Greater than
`<`	Less than
`>=`	Greater than or equal
`<=`	Less than or equal

Examples that use relational operators

```
firstName == "Frank"      // equal to a string literal
txtYears.Text == ""       // equal to an empty string
message == null           // equal to a null value
discountPercent == 2.3    // equal to a numeric literal
isValid == false          // equal to the false value
code == productCode       // equal to another variable

lastName != "Jones"       // not equal to a string literal

years > 0                 // greater than a numeric literal
i < months                // less than a variable

subtotal >= 500           // greater than or equal to a literal
value
quantity <= reorderPoint  // less than or equal to a variable
```

Logical operators

Operator	Name	Description
&&	Conditional-And	Returns a true value if both expressions are true. Only evaluates the second expression if necessary.
||	Conditional-Or	Returns a true value if either expression is true. Only evaluates the second expression if necessary.
&	And	Returns a true value if both expressions are true. Always evaluates both expressions.
|	Or	Returns a true value if either expression is true. Always evaluates both expressions.
!	Not	Reverses the value of the expression.

Examples that use logical operators

```
subtotal >= 250 && subtotal < 500
timeInService <= 4 || timeInService >= 12

isValid == true & counter++ < years
isValid == true | counter++ < years

date > startDate && date < expirationDate || isValid == true
((thisYTD > lastYTD) || empType=="Part time") &&
    startYear < currentYear

!(counter++ >= years)
```

The syntax of the if-else statement

```
if (booleanExpression) { statements }
[else if (booleanExpression) { statements }] ...
[else { statements }]
```

If statements without else if or else clauses

With a single statement

```
if (subtotal >= 100)
   discountPercent = .2m;
```

With a block of statements

```
if (subtotal >= 100)
{
   discountPercent = .2m;
   status = "Bulk rate";
}
```

An if statement with an else clause

```
if (subtotal >= 100)
   discountPercent = .2m;
else
   discountPercent = .1m;
```

An if statement with else if and else clauses

```
if (subtotal >= 100 && subtotal < 200)
   discountPercent = .2m;
else if (subtotal >= 200 && subtotal < 300)
   discountPercent = .3m;
else if (subtotal >= 300)
   discountPercent = .4m;
else
   discountPercent = .1m;
```

Nested if statements

```
if (customerType == "R")
{
   if (subtotal >= 100)            // begin nested if
      discountPercent = .2m;
   else
      discountPercent = .1m;       // end nested if
}
else    // customerType isn't "R"
   discountPercent = .4m;
```

The syntax of the switch statement

```
switch (switchExpression)
{
   case constantExpression:
      statements
      break;
   [case constantExpression:
      statements
      break;] ...
   [default:
      statements
      break;]
}
```

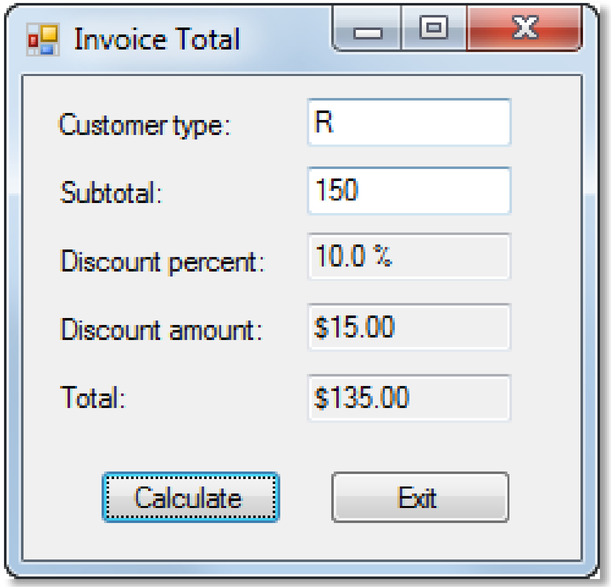
A switch statement with a default label

```
switch (customerType)
{
    case "R":
        discountPercent = .1m;
        break;
    case "C":
        discountPercent = .2m;
        break;
    default:
        discountPercent = .0m;
        break;
}
```

A switch statement that falls through the first case label

```
switch (customerType)
{
    case "R":
    case "C":
        discountPercent = .2m;
        break;
    case "T":
        discountPercent = .4m;
        break;
}
```

The enhanced Invoice Total form



Customer type:	R
Subtotal:	150
Discount percent:	10.0 %
Discount amount:	$15.00
Total:	$135.00

The event handler for the Click event of the Calculate button

```
private void btnCalculate_Click(object sender, EventArgs e)
{
    string customerType = txtCustomerType.Text;
    decimal subtotal = Convert.ToDecimal(txtSubtotal.Text);
    decimal discountPercent = .0m;

    if (customerType == "R")
    {
        if (subtotal < 100)
            discountPercent = .0m;
        else if (subtotal >= 100 && subtotal < 250)
            discountPercent = .1m;
        else if (subtotal >= 250)
            discountPercent =.25m;
    }
    else if (customerType == "C")
    {
        if (subtotal < 250)
            discountPercent = .2m;
        else
            discountPercent = .3m;
    }
```

The event handler for the Click event of the Calculate button (cont.)

```
    else
    {
        discountPercent = .4m;
    }

    decimal discountAmount = subtotal * discountPercent;
    decimal invoiceTotal = subtotal - discountAmount;

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

    txtCustomerType.Focus();
}
```

The syntax of the while statement

```
while (booleanExpression)
{
   statements
}
```

A while loop that adds the numbers 1 through 4

```
int i = 1, sum = 0;
while (i < 5)
{
    sum += i;
    i++;
}
```

A while loop that calculates a future value

```
int i = 1;
while (i <= months)
{
    futureValue = (futureValue + monthlyPayment) *
        (1 + monthlyInterestRate);
    i++;
}
```

The syntax of the do-while statement

```
do
{
   statements
}
while (booleanExpression);
```

A do-while loop that calculates a future value

```
int i = 1;
do
{
    futureValue = (futureValue + monthlyPayment) *
        (1 + monthlyInterestRate);
    i++;
}
while (i <= months);
```

The syntax of the for statement

```
for (initializationExpression; booleanExpression;
     incrementExpression)
{
   statements
}
```

A for loop that stores the numbers 0 through 4 in a string

With a single statement

```
string numbers = null;
for (int i = 0; i < 5; i++)
    numbers += i + " ";
```

With a block of statements

```
string numbers = null;
for (int i = 0; i < 5; i++)
{
    numbers += i;
    numbers += " ";
}
```

A for loop that adds the numbers 8, 6, 4, and 2

```
int sum = 0;
for (int j = 8; j > 0; j-=2)
{
   sum += j;
}
```

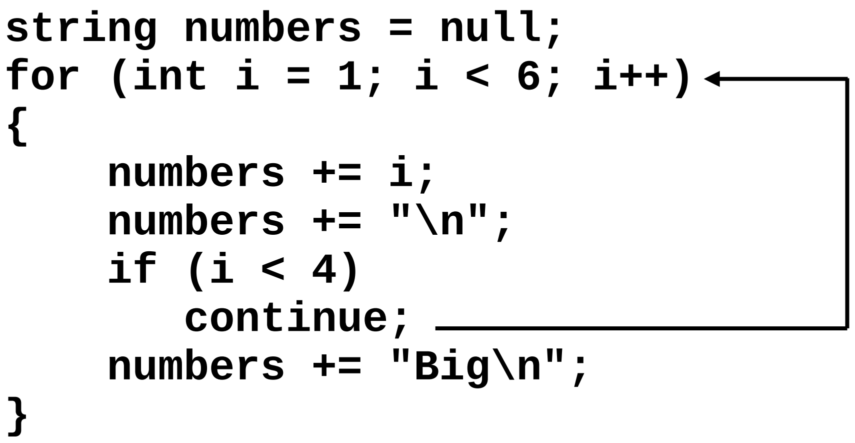
A for loop that calculates a future value

```
for (int i = 1; i <= months; i++)
{
    futureValue = (futureValue + monthlyPayment) *
        (1 + monthlyInterestRate);
}
```

A loop with a break statement

```
   string message = null;
   int i = 1;
   while (i <= months)
   {
       futureValue = (futureValue + monthlyPayment) *
           (1 + monthlyInterestRate);
       if (futureValue > 100000)
       {
           message = "Future value is too large.";
           break;
       }
       i++;
}
```

A loop with a continue statement

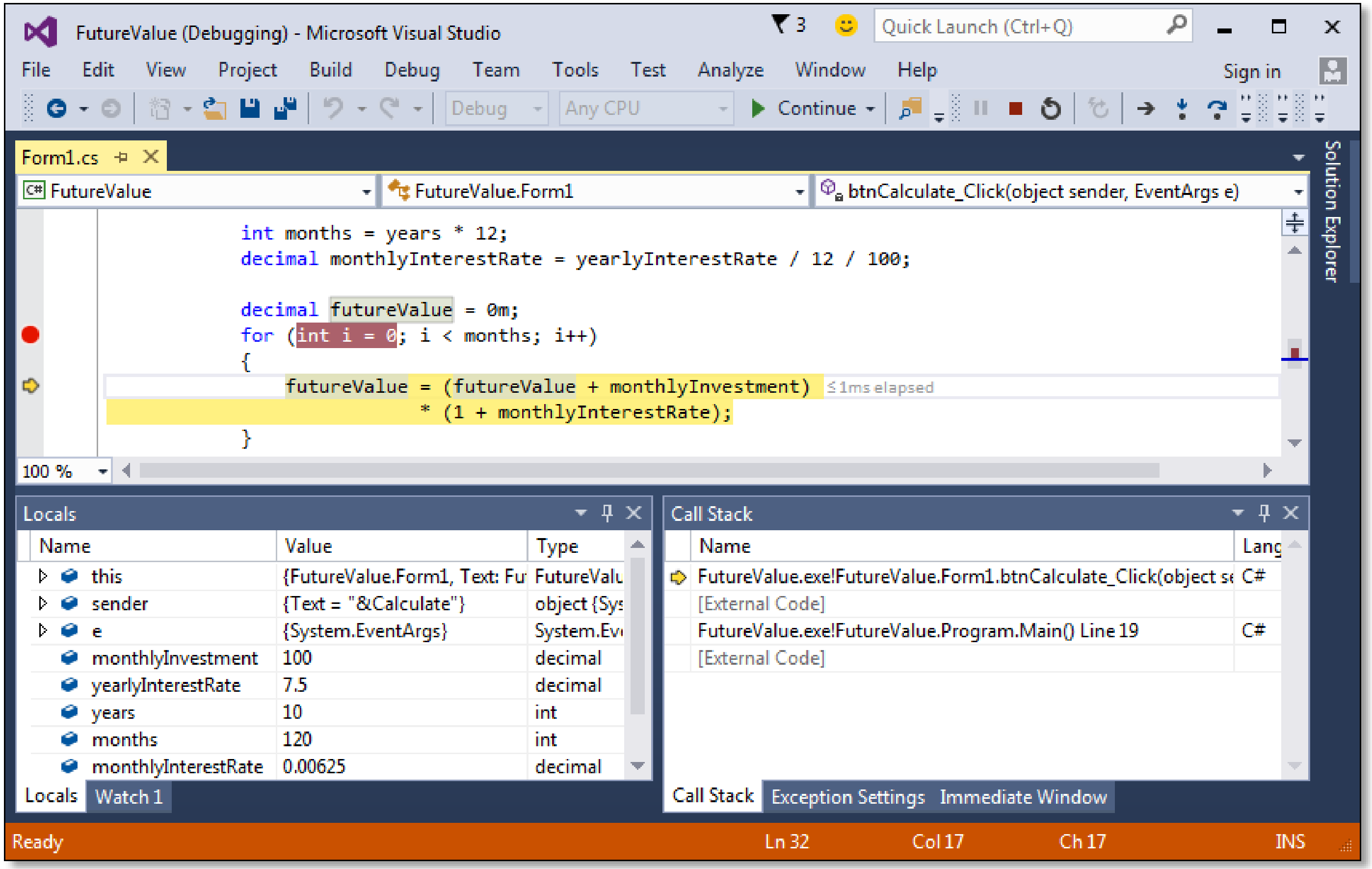


```
string numbers = null;
for (int i = 1; i < 6; i++)
{
    numbers += i;
    numbers += "\n";
    if (i < 4)
       continue;
    numbers += "Big\n";
}
```

The result of this loop

```
1
2
3
4
Big
5
Big
```

A for loop with a breakpoint and an execution point



```
            int months = years * 12;
            decimal monthlyInterestRate = yearlyInterestRate / 12 / 100;

            decimal futureValue = 0m;
            for (int i = 0; i < months; i++)
            {
                futureValue = (futureValue + monthlyInvestment)  ≤1ms elapsed
                    * (1 + monthlyInterestRate);
            }
```

Locals

Name	Value	Type
this	{FutureValue.Form1, Text: Fu	FutureValu
sender	{Text = "&Calculate"}	object {Sys
e	{System.EventArgs}	System.Ev
monthlyInvestment	100	decimal
yearlyInterestRate	7.5	decimal
years	10	int
months	120	int
monthlyInterestRate	0.00625	decimal

Call Stack

Name	Lang
FutureValue.exe!FutureValue.Form1.btnCalculate_Click(object se	C#
[External Code]	
FutureValue.exe!FutureValue.Program.Main() Line 19	C#
[External Code]	

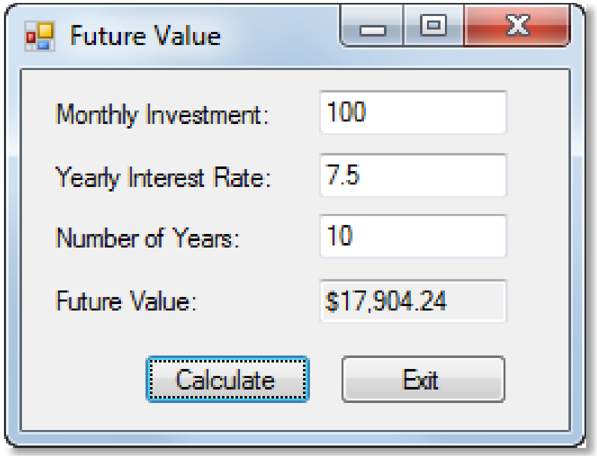
How to set and clear breakpoints

- To set a breakpoint, click or tap in the *margin indicator bar* to the left of a statement. Or, press the F9 key to set a breakpoint at the cursor insertion point. Then, a red dot will mark the breakpoint.
- To remove a breakpoint, use any of the techniques for setting a breakpoint. To remove all breakpoints at once, use the Delete All Breakpoints command in the Debug menu.

How to work in break mode

- In break mode, a yellow arrowhead marks the current *execution point*, which points to the next statement that will be executed.
- To *step through* your code one statement at a time, press the F11 key or click the Step Into button on the Debug toolbar.
- To continue normal processing until the next breakpoint is reached, press the F5 key.

The Future Value form



Monthly Investment: 100

Yearly Interest Rate: 7.5

Number of Years: 10

Future Value: $17,904.24

The property settings for the form

Default name	Property	Setting
`Form1`	`Text`	`Future Value`
	`AcceptButton`	`btnCalculate`
	`CancelButton`	`btnExit`
	`StartPosition`	`CenterScreen`

The property settings for the controls

Default name	Property	Setting
`label1`	`Text`	`Monthly Investment:`
`label2`	`Text`	`Yearly Interest Rate:`
`label3`	`Text`	`Number of Years:`
`label4`	`Text`	`Future Value:`
`textBox1`	`Name`	`txtMonthlyInvestment`
`textBox2`	`Name`	`txtInterestRate`
`textBox3`	`Name`	`txtYears`

`textBox4`	`Name`
	`ReadOnly`
	`TabStop`
		`txtFutureValue`
		`True`
		`False`

`button1`	`Name` `Text`	`btnCalculate` `&Calculate`
`button2`	`Name` `Text`	`btnExit` `E&xit`

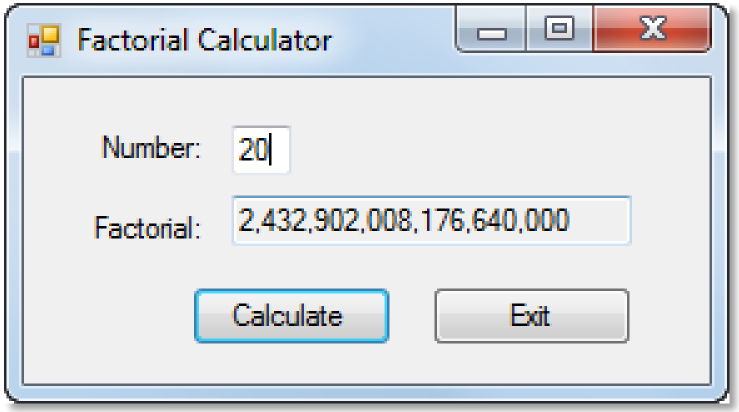
Additional property settings

- The TextAlign property of each of the labels is set to MiddleLeft.
- The TabIndex properties of the controls are set so the focus moves from top to bottom and left to right.

The code for the event handlers in the Future Value application

```
private void btnCalculate_Click(object sender, EventArgs e)
{
    decimal monthlyInvestment =
        Convert.ToDecimal(txtMonthlyInvestment.Text);
    decimal yearlyInterestRate = Convert.ToDecimal(txtInterestRate.Text);
    int years = Convert.ToInt32(txtYears.Text);

    int months = years * 12;
    decimal monthlyInterestRate = yearlyInterestRate / 12 / 100;

    decimal futureValue = 0m;
    for (int i = 0; i < months; i++)
    {
        futureValue = (futureValue + monthlyInvestment)
                    * (1 + monthlyInterestRate);
    }

    txtFutureValue.Text = futureValue.ToString("c");
    txtMonthlyInvestment.Focus();
}

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

Extra 5-1 Calculate the factorial of a number

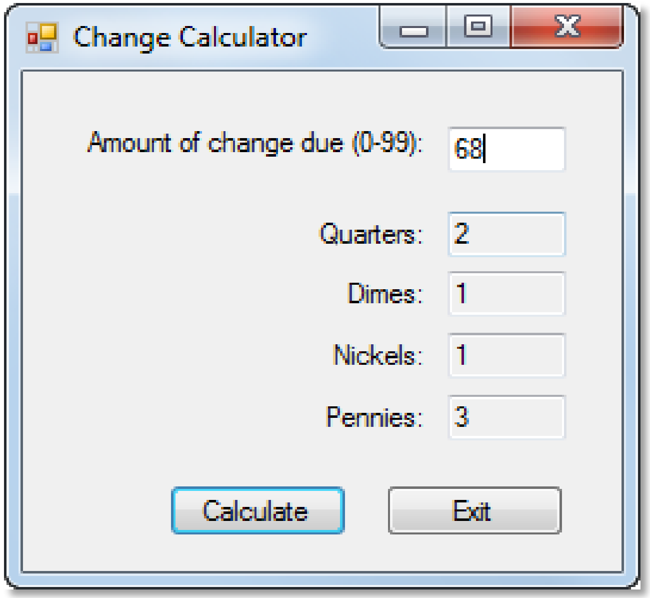


Number: 20

Factorial: 2,432,902,008,176,640,000

Accept an integer and then calculate the factorial of that integer.

Extra 5-2 Calculate change



Amount of change due (0-99): 68

Quarters: 2

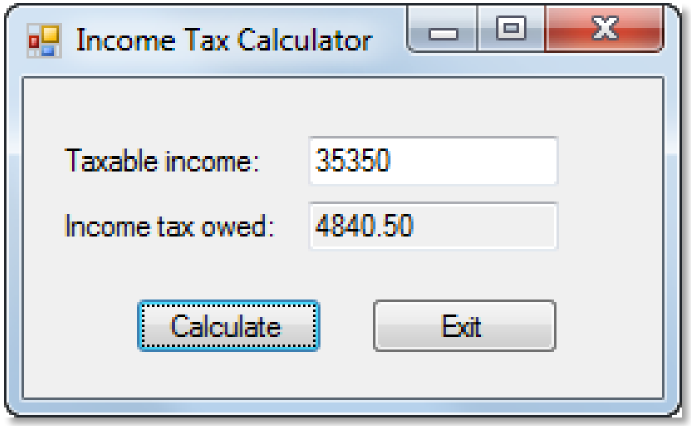
Dimes: 1

Nickels: 1

Pennies: 3

Calculate the number of quarters, dimes, nickels, and pennies needed to make change for an amount between 0 and 99 cents.

Extra 5-3 Calculate income tax



Taxable income: 35350

Income tax owed: 4840.50

Calculate the federal income tax owed for a taxable income amount.