GLOBAL EDITION

Starting Out With Visual C#

FOURTH EDITION

Tony Gaddis

Pearson

Digital Resources for Students

Your new textbook provides 12-month access to digital resources that may include VideoNotes (step-by-step video tutorials on programming concepts), source code, web chapters, quizzes, and more. Refer to the preface in the textbook for a detailed list of resources.

Follow the instructions below to register for the Companion Website for Tony Gaddis' *Starting Out with Visual C#*, Fourth Edition, Global Edition.

1. Go to www.pearsonglobaleditions.com/Gaddis
2. Enter the title of your textbook or browse by author name
3. Click Companion Website
4. Click Register and follow the on-screen instructions to create a login name and password

Use a coin to scratch off the coating and reveal your access code.
Do not use a sharp knife or other sharp object as it may damage the code.

Use the login name and password you created during registration to start using the digital resources that accompany your textbook.

IMPORTANT:

This prepaid subscription does not include access to MyProgrammingLab, which is available at **www.myprogramminglab.com** for purchase.

This access code can only be used once. This subscription is valid for 12 months upon activation and is not transferrable. If the access code has already been revealed it may no longer be valid.

For technical support go to https://support.pearson.com/getsupport

Step 5: Switch your view back to the *Designer* and double-click the `exitButton` control. In the code editor you see an empty event handler named `exitButton_Click`. Complete the `exitButton_Click` event handler by typing the code shown in lines 78–79 in Program 4-3.

Step 6: Save the project and run the application. First, enter 45000 for the salary and 1 for the years at current job. Click the *Check Qualifications* button, and the application should display the message "Minimum years at current job not met."

Click the *Clear* button. Enter 35000 for the salary and 5 for the years at current job. Click the *Check Qualifications* button, and the application should display the message "Minimum salary requirement not met."

Click the *Clear* button. Enter 45000 for the salary and 5 for the years at current job. Click the *Check Qualifications* button, and the application should display the message "You qualify for the loan."

Continue to test the application as you wish. When you are finished, click the *Exit* button, and the form should close.

Program 4-3 Completed Form1 code for the *Loan Qualifier* application

```
 1 using System;
 2 using System.Collections.Generic;
 3 using System.ComponentModel;
 4 using System.Data;
 5 using System.Drawing;
 6 using System.Linq;
 7 using System.Text;
 8 using System.Threading.Tasks;
 9 using System.Windows.Forms;
10
11 namespace Loan_Qualifier
12 {
13    public partial class Form1 : Form
14    {
15       public Form1()
16       {
17           InitializeComponent();
18       }
19
20       private void checkButton_Click(object sender, EventArgs e)
21       {
22           try
23           {
24               // Names constants
25               const decimal MINIMUM_SALARY = 40000m;
26               const int MINIMUM_YEARS_ON_JOB = 2;
27
28               // Local variables
29               decimal salary;
30               int yearsOnJob;
31
32               // Get the salary and years on the job.
33               salary = decimal.Parse(salaryTextBox.Text);
34               yearsOnJob = int.Parse(yearsTextBox.Text);
35
```

```
36                // Determine whether the user qualifies.
37                if (salary >= MINIMUM_SALARY)
38                {
39                    if (yearsOnJob >= MINIMUM_YEARS_ON_JOB)
40                    {
41                        // The user qualifies.
42                        decisionLabel.Text = "You qualify for the loan.";
43                    }
44                    else
45                    {
46                        // The user does not qualify.
47                        decisionLabel.Text = "Minimum years at current " +
48                            "job not met.";
49                    }
50                }
51                else
52                {
53                    // The user does not qualify.
54                    decisionLabel.Text = "Minimum salary requirement " +
55                        "not met.";
56                }
57            }
58            catch (Exception ex)
59            {
60                // Display an error message.
61                MessageBox.Show(ex.Message);
62            }
63        }
64
65        private void clearButton_Click(object sender, EventArgs e)
66        {
67            // Clear the TextBoxes and the decisionLabel.
68            salaryTextBox.Text = "";
69            yearsTextBox.Text = "";
70            decisionLabel.Text = "";
71
72            // Reset the focus.
73            salaryTextBox.Focus();
74        }
75
76        private void exitButton_Click(object sender, EventArgs e)
77        {
78            // Close the form.
79            this.Close();
80        }
81    }
82 }
```

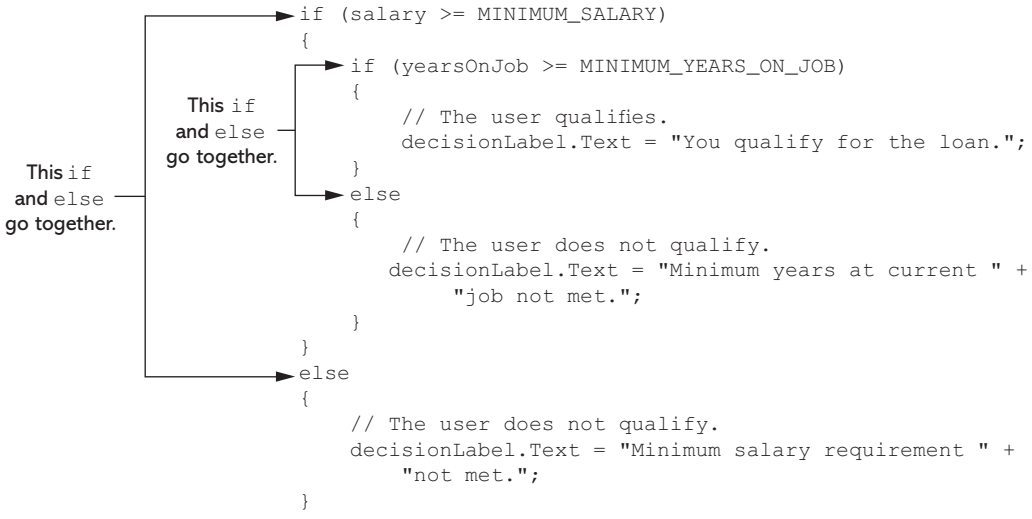
Indentation and Alignment in Nested Decision Structures

For debugging purposes, it's important to use proper alignment and indentation in a nested `if` statement. This makes it easier to see which actions are performed by each part of the structure. For example, the following code is functionally equivalent to lines 37–56 in Program 4-3. Although this code is logically correct, it would be very difficult to debug because it is not properly indented.

```
if (salary >= MINIMUM_SALARY)
{
if (yearsOnJob >= MINIMUM_YEARS_ON_JOB)
{
// The user qualifies.
decisionLabel.Text = "You qualify for the loan.";
}
else
{
// The user does not qualify.
decisionLabel.Text = "Minimum years at current " +
    "job not met.";
}
}
else
{
// The user does not qualify.
decisionLabel.Text = "Minimum salary requirement " +
    "not met.";
}
```

Fortunately, Visual Studio automatically indents and aligns the statements in a decision structure. Proper indentation and alignment make it easier to see which `if` and `else` clauses belong together, as shown in Figure 4-14.

Figure 4-14 Alignment of `if` and `else` clauses

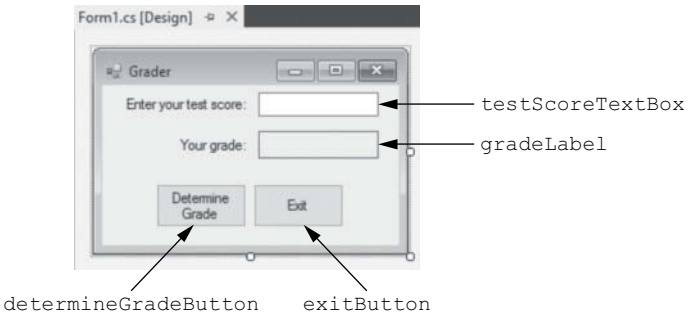


Testing a Series of Conditions

In Tutorial 4-3, you saw how a program can use nested decision structures to test more than one Boolean expression. It is not uncommon for a program to have a series of Boolean expressions to test and then perform an action, depending on which expression is true. One way to accomplish this it to have a decision structure with numerous other decision structures nested inside it. For example, look at the *Grader* application in the *Chap04* folder of this book's Student Sample Programs.

Figure 4-15 shows the application's form, with the names of several controls. When you run the application, you enter a numeric test score into the `testScoreTexBox` control and click the `determineGradeButton` control; a grade is then displayed in the `gradeLabel` control.

Figure 4-15 The *Grader* application's form



The following 10-point grading scale is used to determine the grade:

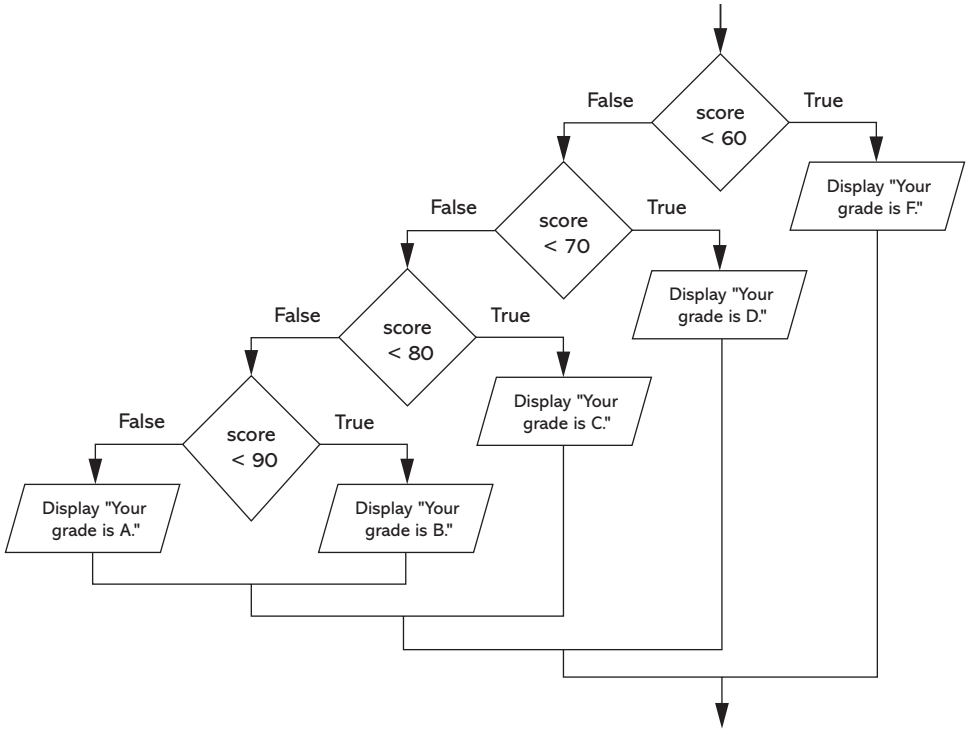
Test Score	Grade
90 and above	A
80–89	B
70–79	C
60–69	D
Below 60	F

The logic of determining the grade can be expressed like this:

If the test score is less than 60, then the grade is "F."
Otherwise, if the test score is less than 70, then the grade is "D."
Otherwise, if the test score is less than 80, then the grade is "C."
Otherwise, if the test score is less than 90, then the grade is "B."
Otherwise, the grade is "A."

This logic requires several nested decision structures, as shown in the flowchart in Figure 4-16.

Figure 4-16 Nested decision structure to determine a grade

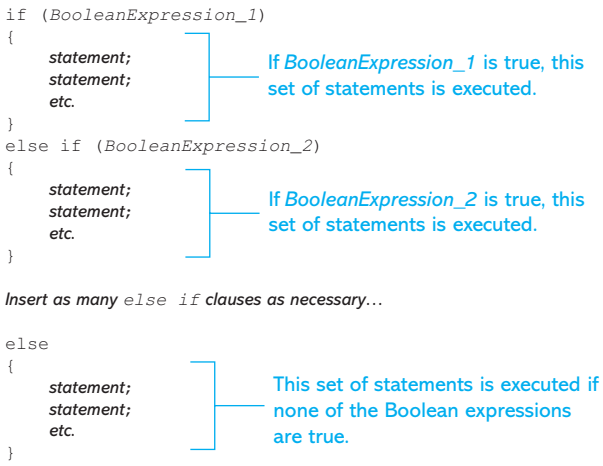


Open the code editor and look at the `determineGradeButton_Click` event handler, shown in the following code sample. The nested decision structure appears in lines 12–41.

```
 1 private void determineGradeButton_Click(object sender, EventArgs e)
 2 {
 3     try
 4     {
 5         // Variable to hold the test score.
 6         double testScore;
 7
 8         // Get the test score.
 9         testScore = double.Parse(testScoreTextBox.Text);
10
11         // Determine the grade.
12         if (testScore < 60)
13         {
14             gradeLabel.Text = "F";
15         }
16         else
17         {
18             if (testScore < 70)
19             {
20                 gradeLabel.Text = "D";
21             }
22             else
23             {
24                 if (testScore < 80)
25                 {
26                     gradeLabel.Text = "C";
27                 }
28                 else
29                 {
30                     if (testScore < 90)
31                     {
32                         gradeLabel.Text = "B";
33                     }
34                     else
35                     {
36                         gradeLabel.Text = "A";
37                     }
38                 }
39             }
40         }
41     }
42     catch (Exception ex)
43     {
44         // Display an error message.
45         MessageBox.Show(ex.Message);
46     }
47 }
```

The `if-else-if` Statement

Even though the *Grader* application previously shown is a simple example, the logic of the nested decision structure is fairly complex. C# provides a special version of the decision structure known as the **if-else-if statement**, which makes this type of logic simpler to write. You write the `if-else-if` statement using the following general format:



When the statement executes, *BooleanExpression_1* is tested. If *BooleanExpression_1* is true, the set of statements that immediately follows is executed, and the rest of the structure is skipped. If *BooleanExpression_1* is false, however, the program jumps to the very next `else if` clause and tests *BooleanExpression_2*. If it is true, the set of statements that immediately follows is executed, and the rest of the structure is then skipped. This process continues until a Boolean expression is found to be true, or no more `else if` clauses are left. If none of the Boolean expressions are true, the set of statements following the final `else` clause is executed.

For example, look at the *Grader2* application in the *Chap04* folder of this book's Student Sample Programs. This application works just like the *Grader* application that was previously discussed. The user enters a numeric test score, and the application displays a grade. Its form is identical to the form shown in Figure 4-15. The *Grader2* application, however, uses an `if-else-if` statement to determine the grade instead of nested `if-else` statements. The *Grader2* application's `determineGradeButton_Click` event handler is shown here:

```
1  private void determineGradeButton_Click(object sender, EventArgs e)
2  {
3      try
4      {
5          // Variable to hold the test score.
6          double testScore;
7
8          // Get the test score.
9          testScore = double.Parse(testScoreTextBox.Text);
10
11         // Determine the grade.
12         if (testScore < 60)
13         {
14             gradeLabel.Text = "F";
15         }
16         else if (testScore < 70)
17         {
18             gradeLabel.Text = "D";
19         }
20         else if (testScore < 80)
21         {
22             gradeLabel.Text = "C";
23         }
24         else if (testScore < 90)
25         {
26             gradeLabel.Text = "B";
```