Comments

Comments are short notes that are placed in different parts of a program, explaining how those parts of the program work. Comments are not intended (غير مخصصة) for the compiler. They are intended for any person who is reading the code and trying to understand what it does.

In C#, as beginners there are two types of comments:

1- **Line comments** appears on one line in a program. You begin a line comment with two forward slashes (//).

```
private void showRightAnswer_Click(object sender, EventArgs e)
{
  // Make the answer label appear to the user.
   answerLabel.visible= true;
}
```

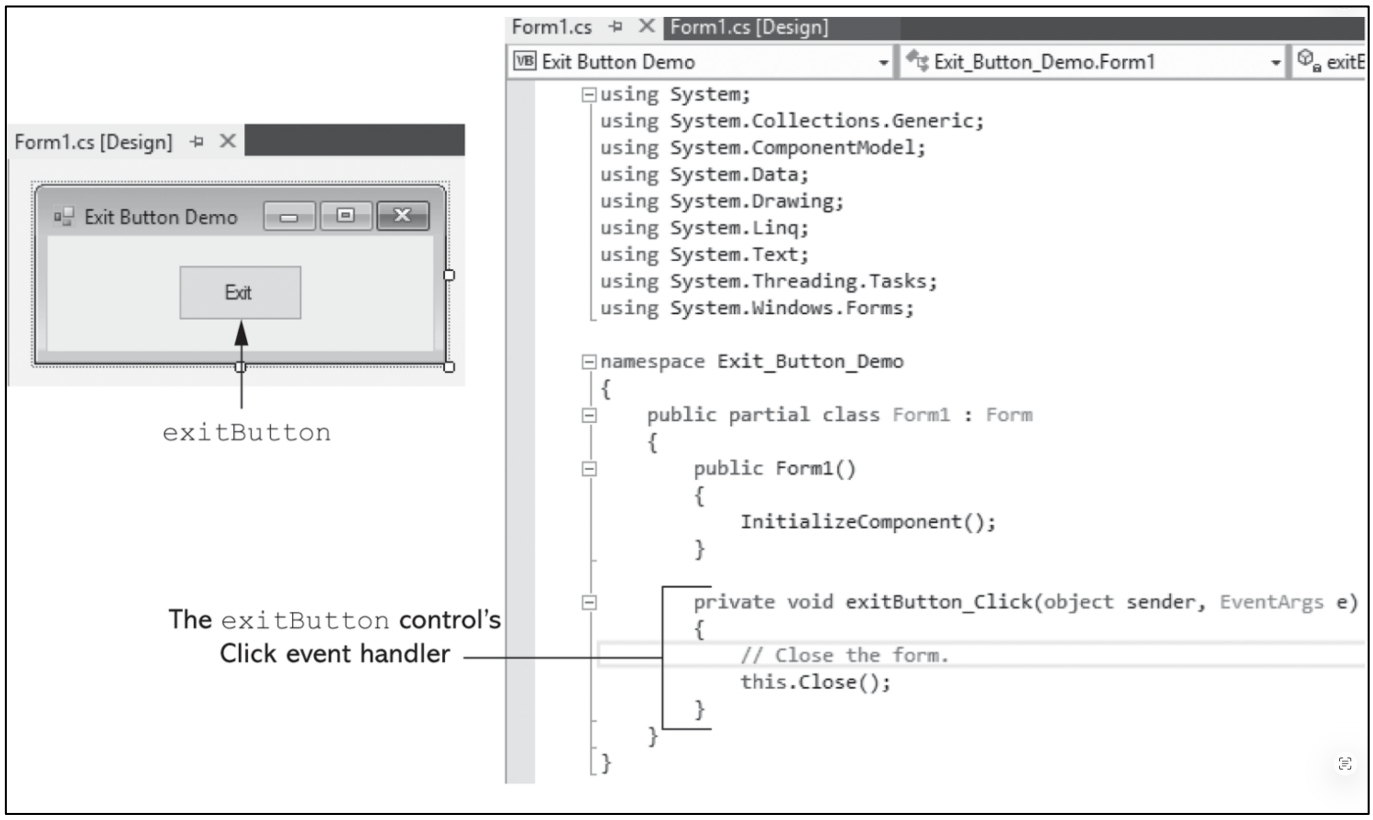
2- **Block comment** can occupy multiple lines in a program. A block comment starts with /* (a forward slash followed by an asterisk) and ends with */ (an asterisk followed by a forward slash).

```
private void showRightAnswer_Click(object sender, EventArgs e)
{
 /* Make the right answer label show to the user.
    When the user press the button it will show the result immediately */
  answerLabel.visible= true;
}
```

Writing the Code to Close an Application's Form

To close an application's form in code, you use the statement **this.Close();**. Figure 3-1 shows the form and code from a project named Exit Button Demo.

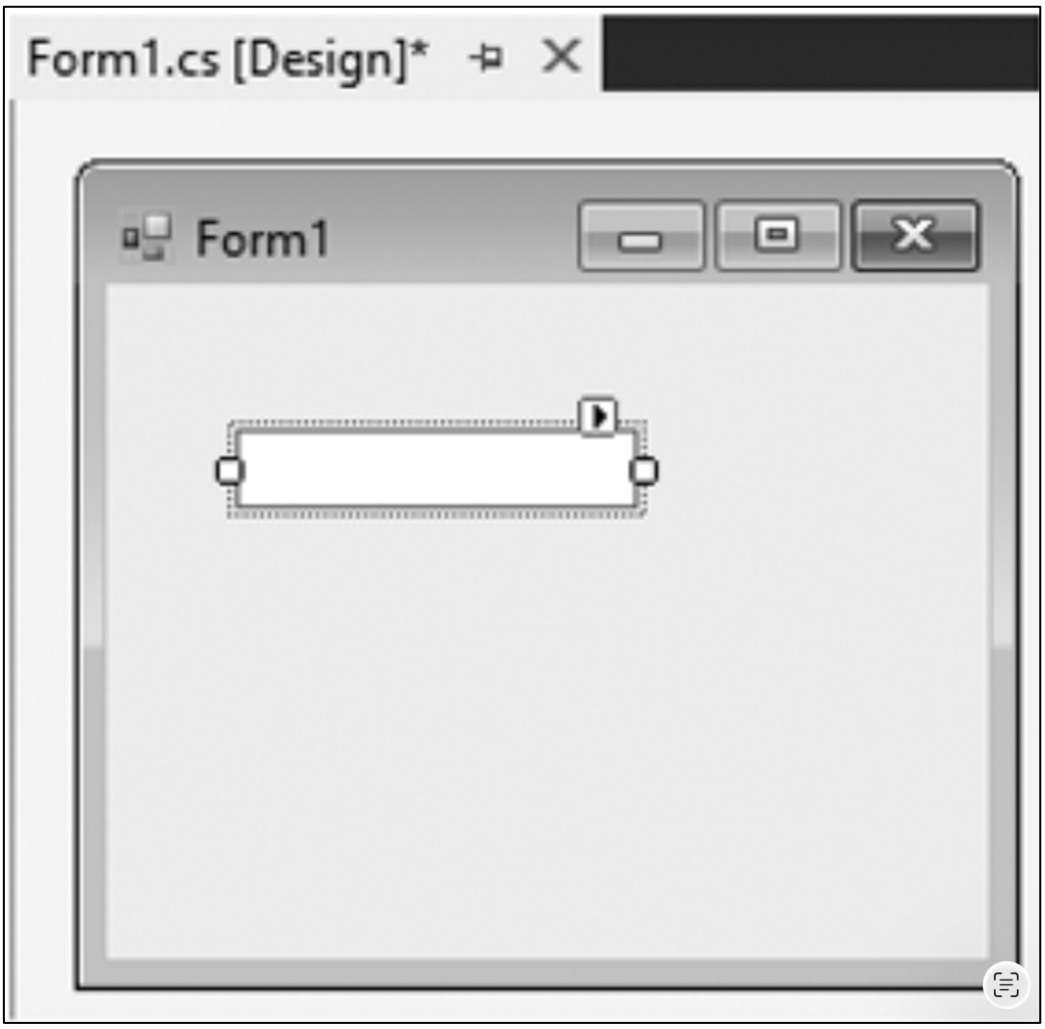
Figure 3-1 A form with an Exit button



TextBox Controls

The TextBox control is a rectangular area that can accept keyboard input from the user. When the application is running, the user can type text into a TextBox control. The program can then retrieve the text that the user entered and use that text in any necessary operations. The Figure 3-2 shows a sample of TextBox control. When you create TextBox controls, they are automatically given default names such as **textBox1**, **textBox2**, and so forth.

Figure 3-2 A TextBox control



When the user types into a TextBox control, the text is stored in the control's Text property. When you retrieve the contents of the Text property, you always get a string. Let's look at an example. Figure 3-3 shows the form, with most of the control names specified, and Figure 3-4 shows the form's code.

Figure 3-3 The TextBox Demo application

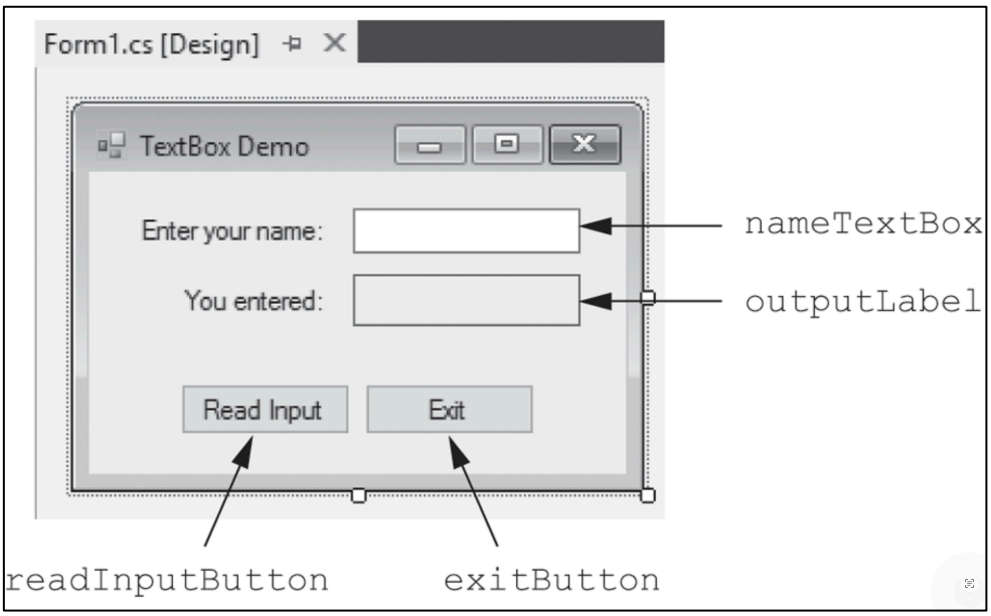
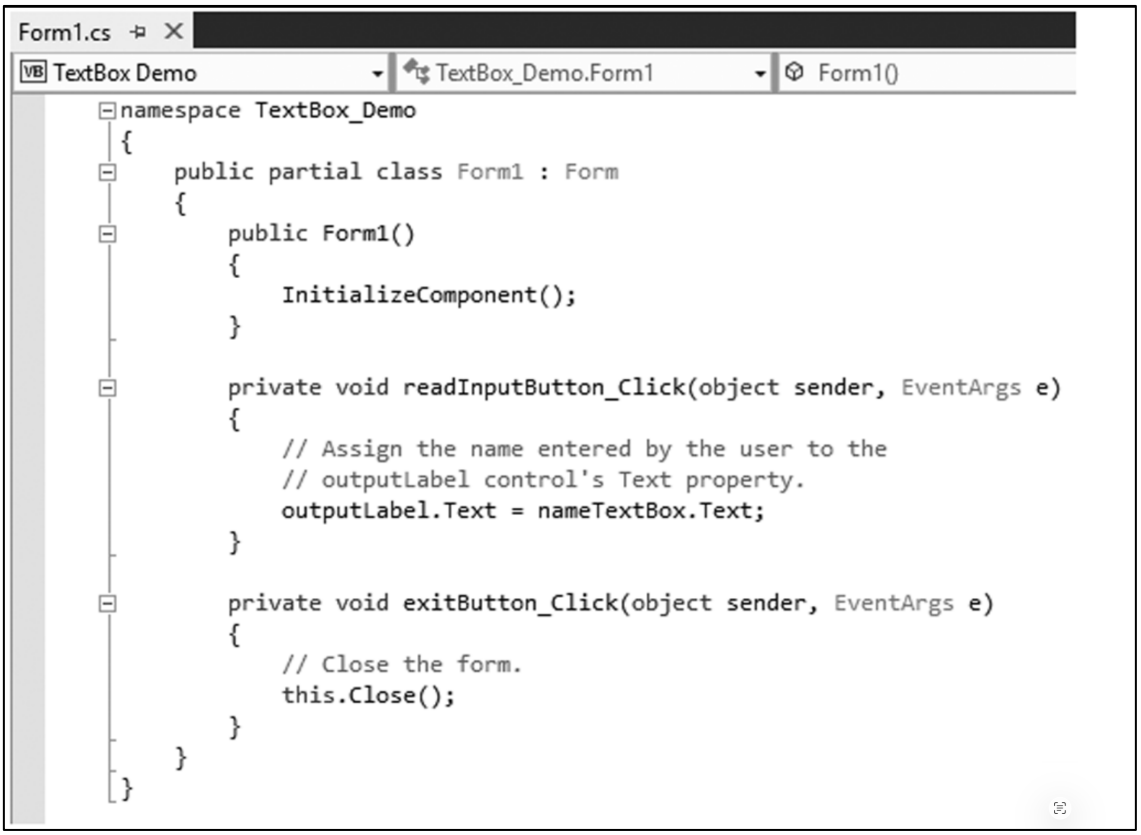


Figure 3-4 The form's code (excluding the using directives)

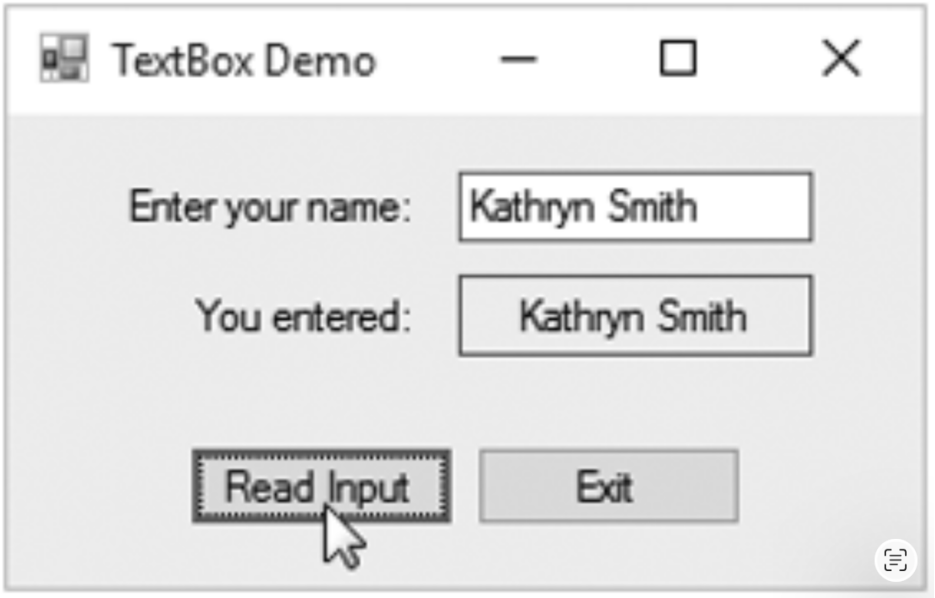


Notice in Figure 3-4 that the readInputButton control's Click event handler performs the following assignment statement:

```
outputLabel.Text = nameTextBox.Text;
```

This statement assigns the value of the nameTextBox control's Text property to the outputLabel control's Text property. If you run the application, Figure 3-5 shows an example of how the form appears after you have entered Kathryn Smith and clicked the readInputButton control.

Figure 3-5 The user's name displayed in the label



Variables

A variable is a storage location in memory that is represented by a name.

For example, a program that manages a company's customer mailing list might use a variable named lastName to hold a customer's last name, a variable named firstName to hold the customer's first name, a variable named address to hold the customer's mailing address, and so forth.

In C#, you must declare a variable in a program before you can use it to store data. You do this with a variable declaration, which specifies two things about the variable:

The variable's data type, which is the type of data the variable will hold The variable's name A variable declaration statement is written in this general format:

```
DataType VariableName;
```

Data Type

A variable's data type indicates the type of data that the variable will hold.

The C# language provides many data types for storing fundamental types of data, such as strings, integers, and real numbers. These data types are known as primitive data types.

Variable Name

A variable name identifies a variable in the program code. When naming a variable, you should always choose a meaningful name that indicates what the variable is used for.

Note: The same rules for identifiers that apply to control names also apply to variable names.

String Concatenation

In C# you use the + operator to concatenate strings. The + operator produces a string that is the combination of the two strings used as its operands.

```
string message;
message = "Hello " + "world";
MessageBox.Show(message);
```

When the message box is displayed, it shows the string Hello world.

Let's look at an application that further demonstrates string concatenation. Figure 3-6 shows the form, with most of the control names specified, and Figure 3-7 shows the form's code.

Figure 3-6 The String Variable Demo application

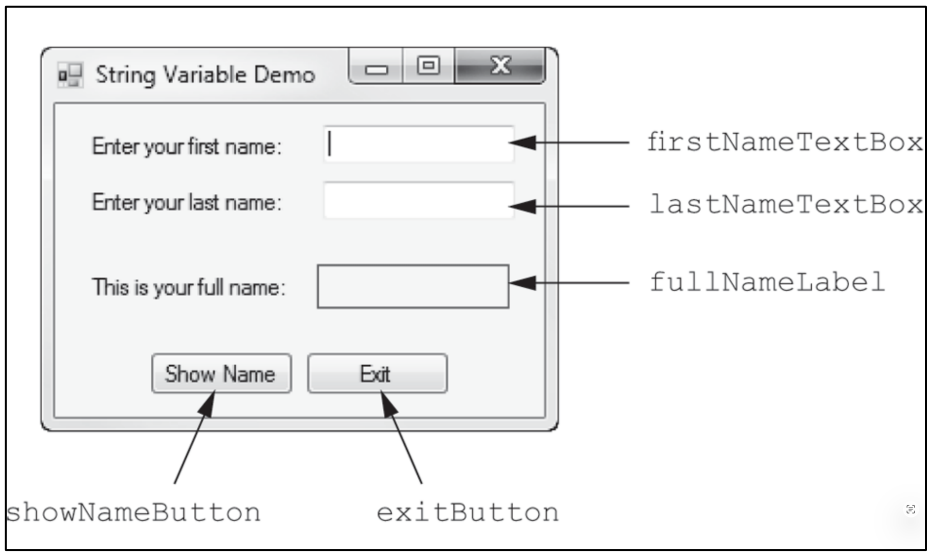
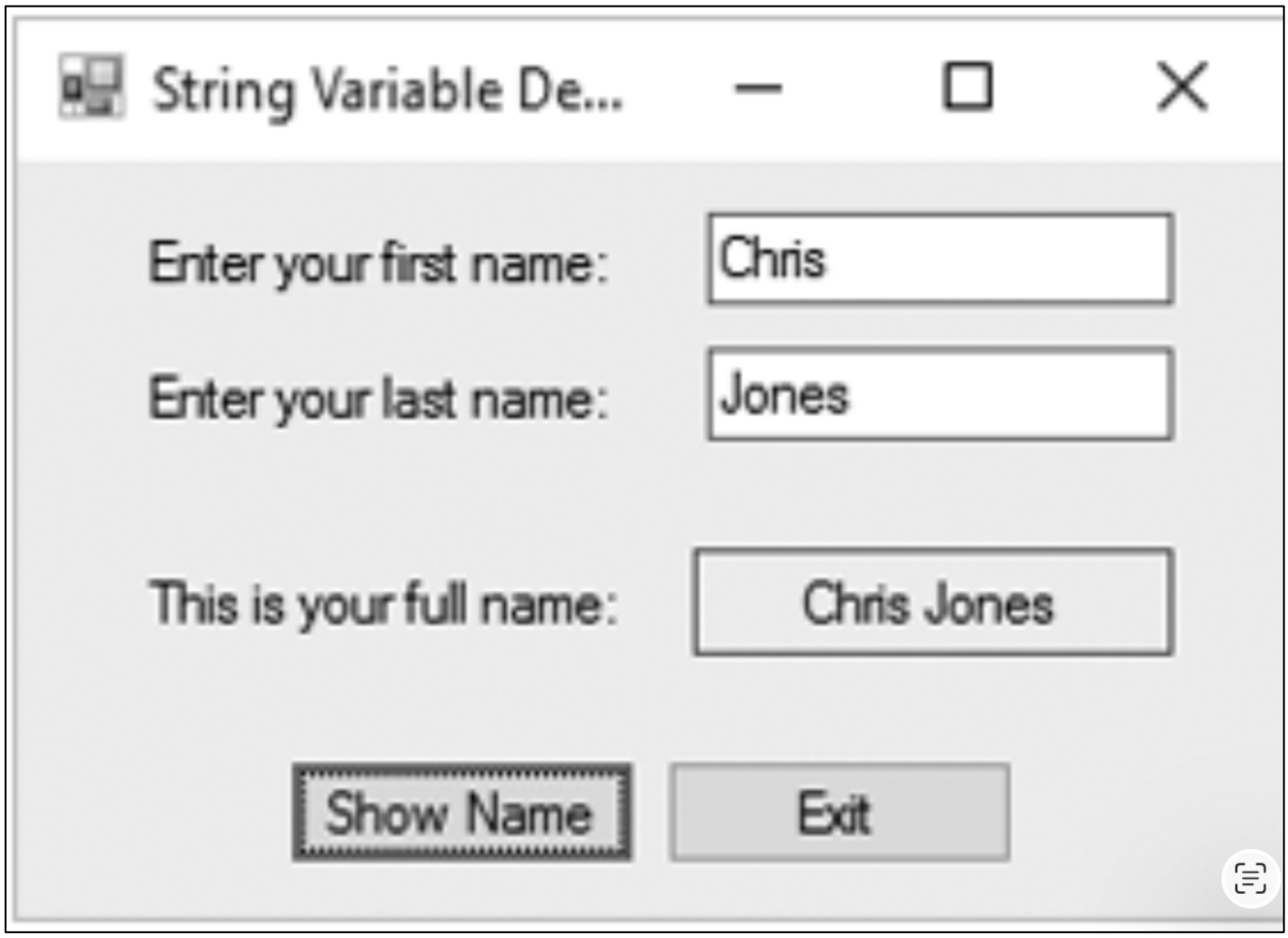


Figure 3-7 The form's code (excluding the using directives)

```
namespace String_Variable_Demo
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void showNameButton_Click(object sender, EventArgs e)
        {
            // Declare a string variable to hold the full name.
(1) ->      string fullName;

            // Combine the names, with a space between them. Assign the
            // result to the fullName variable.
(2) ->      fullName = firstNameTextBox.Text + " " + lastNameTextBox.Text;

            // Display the fullName variable in the fullNameLabel control.
(3) ->      fullNameLabel.Text = fullName;
        }

        private void exitButton_Click(object sender, EventArgs e)
        {
            // Close the form.
            this.Close();
        }
    }
}
```

If you run the application, Figure 3-8 shows an example of how the form appears after you have entered Chris for the first name and Jones for the last name and clicked the showNameButton control.

Figure 3-8 The user's full name displayed in the label



Local Variables

Variables that are declared inside a method are known as local variables. A local variable belongs to the method in which it is declared, and only statements inside that method can access the variable.

An error will occur if a statement in one method tries to access a local variable that belongs to another method. For example, the sample code shown in Figure 3-9:

Figure 3-9 One method trying to access a variable that is local to another method

```
    private void firstButton_Click(object sender, EventArgs e)
    {
        // Declare a string variable.
1 ──►   string myName;

        // Assign the nameTextBox control's Text property
        // to the myName variable.
2 ──►   myName = nameTextBox.Text;
    }

    private void secondButton_Click(object sender, EventArgs e)
    {
        // Assign the myName variable to the outputLabel
        // control's Text property.
3 ──►   outputLabel.Text = myName;  ◄──── ERROR!
    }
```

The Birth Date String Application

create an application that lets the user enter the following information about his or her birthdate:

The day of the week (Monday, Tuesday, etc.)

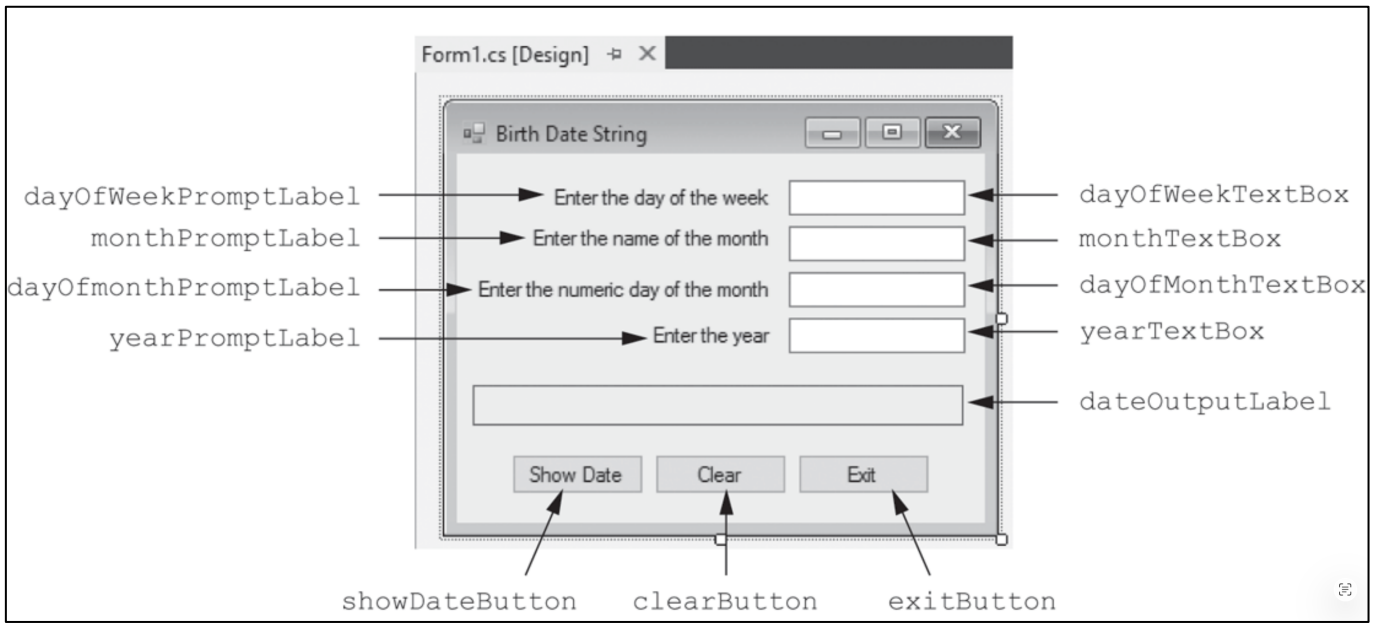
The name of the month (January, February, etc.)

The numeric day of the month

The year

Figure 3-10 shows the application's form, along with the names of all the controls.

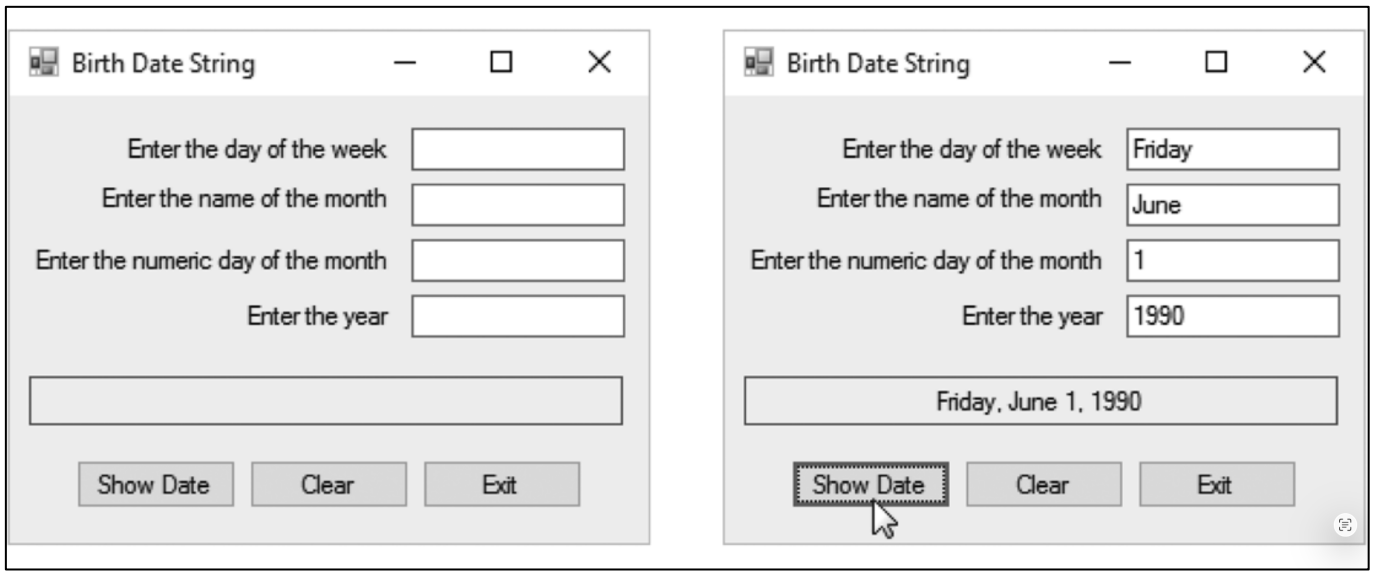
Figure 3-10 Birth date application



Control Name	Control Type	Property Settings
dayOfWeekPromptLabel	Label	**Text:** *Enter the day of the week*
monthPromptLabel	Label	**Text:** *Enter the name of the month*
dayOfMonthPromptLabel	Label	**Text:** *Enter the numeric day of the month*
yearPromptLabel	Label	**Text:** *Enter the year*
dayOfWeekTextBox	TextBox	No properties changed
monthTextBox	TextBox	No properties changed
dayOfMonthTextBox	TextBox	No properties changed
yearTextBox	TextBox	No properties changed
dateOutputLabel	Label	**AutoSize:** False **BorderStyle:** FixedSingle **Text:** (The contents of the Text property have been erased.) **TextAlign:** MiddleCenter
showDateButton	Button	**Text:** *Show Date*
clearButton	Button	**Text:** *Clear*
exitButton	Button	**Text:** *Exit*

The form will appear as shown in the image on the left in Figure 3-11

Figure 3-11 The Birth Date String application



Declaring Multiple Variables with One Statement

You can declare multiple variables of the same data type with one declaration statement. Here is an example:

```
string lastName, firstName, middleName;
```

Numeric Data Types and Variables

If you need to store a number in a variable and use that number in a mathematical operation, the variable must be of a numeric data type. Table 3.1 shows the primitive numeric data types.

Table 3.1 The primitive numeric data types that you will use most often

Data Type	Description
int	A variable of the int data type can hold whole numbers only. For example, an int variable can hold values such as 42, 0, and −99. An int variable cannot hold numbers with a fractional (كسري) part, such as 22.1 or −4.9.
double	A variable of the double data type can hold real numbers, such as 3.5, −87.95, or 3.0. A number that is stored in a double variable is rounded to 15 digits of precision.

Here are examples of declaring variables of each data type:

```
int speed;
double distance;
```

Declaring Local Variables with the var Keyword

C# provides an alternative way to declare local variables, using the var keyword and an initialization value. Here is an example:

```
var amount = 100;
```

Notice that this statement uses the word var instead of a data type. The var keyword tells the compiler to determine the variable's data type from the initialization value.

var age = 30

var user_name = “Ahmed”

var avarage = 89.5;

Inputting and Outputting Numeric Values

If the user has entered a number into a TextBox, the number will be stored as a string in the TextBox’s Text property. If you want to store that number in a numeric variable, you have to convert it to the appropriate numeric data type. When you want to display the value of a numeric variable in a Label control or a message box, you have to convert it to a string.

Getting a Number from a TextBox

Any data that the user enters into a TextBox control is stored in the control’s Text property as a **string**, even if it is a number. For example, if the user enters the number 72 into a TextBox control, the input is stored as the string **"72"** in the control’s Text property.

The **Parse** methods are used to convert a string to a specific data type.

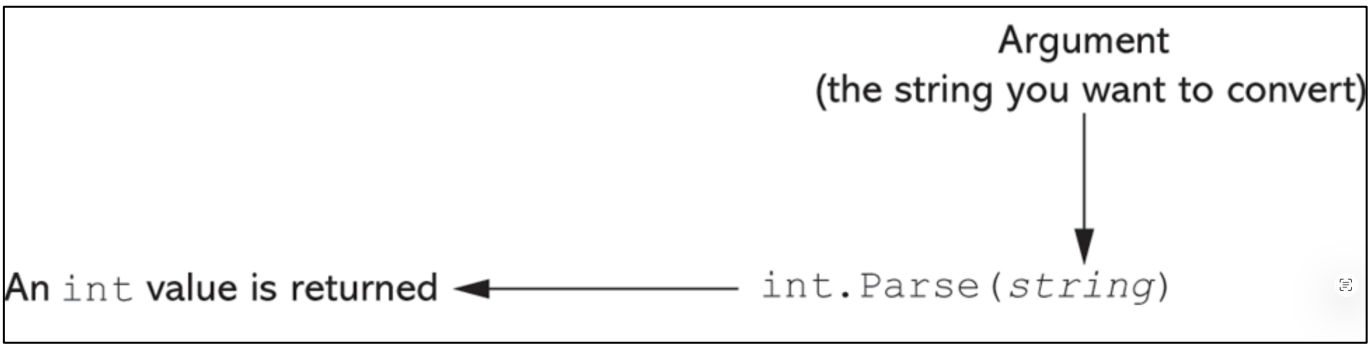
We use the **int.Parse** method to convert a string to an int.

We use the **double.Parse** method to convert a string to a double.

We use the **decimal.Parse** method to convert a string to a decimal.

Figure 3-12 illustrates this concept using the int.Parse method as an example.

Figure 3-12 The int.Parse method



The following code sample shows how to use the int.Parse method to convert a control's Text property to an int. Assume that hoursWorkedTextBox is the name of a TextBox control. Figure 3-13 illustrates this process.

```
int hoursWorked;

hoursWorked = int.Parse(hoursWorkedTextBox.Text);
```

Figure 3-13 Converting TextBox input to an int