The C# Keywords

At its foundation, a computer language is defined by its keywords because they determine the features built into the language. C# defines two general types of keywords: *reserved* and *contextual.* The reserved keywords cannot be used as names for variables, classes, or methods.They can be used only as keywords. This is why they are called *reserved.* The terms *reserved* *words* or *reserved identifiers* are also sometimes used. There are currently 77 reserved keywordsdefined by version 3.0 of the C# language. They are shown in Table 2-1.

C# 3.0 defines 13 contextual keywords that have a special meaning in certain contexts. In those contexts, they act as keywords. Outside those contexts, they can be used as names for other program elements, such as variable names. Thus, they are not technically reserved. As a general rule, however, you should consider the contextual keywords reserved and avoid using them for any other purpose. Using a contextual keyword as a name for some other program element can be confusing and is considered bad practice by many programmers. The contextual keywords are shown in Table 2-2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| abstract | as | base | bool | break |
| byte | case | catch | char | checked |
| class | const | continue | decimal | default |
| delegate | do | double | else | enum |
| event | explicit | extern | false | finally |
| fixed | float | for | foreach | goto |
| if | implicit | in | int | inter face |
| internal | is | lock | long | namespace |
| new | null | object | operator | out |
| override | params | private | protected | public |
| readonly | ref | return | sbyte | sealed |
| shor t | sizeof | stackalloc | static | string |
| struct | switch | this | throw | true |
| tr y | typeof | uint | ulong | unchecked |
| unsafe | ushor t | using | vir tual | volatile |
| void | while |  |  |  |
|  |  |  |  |  |

TABLE 2-1: The C# Reserved Keywords

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| from | get | group |  | into |  | join |  |
| let | Order by | partial |  | select |  | set |  |
| value | where | yield |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

TABLE 2-2: The C# Contextual Keywords

Identifiers

In C#, an identifier is a name assigned to a method, a variable, or any other user-defined item. Identifiers can be one or more characters long. Variable names may start with any letter of the alphabet or an underscore. Next may be a letter, a digit, or an underscore. The underscore can be used to enhance the readability of a variable name, as in **line\_count**. However, identifers containing two consecutive underscores, such as **max\_ \_value**, are reserved for use by the compiler. Uppercase and lowercase are different; that is, to C#, **myvar** and **MyVar** are separate names. Here are some examples of acceptable identifiers:

|  |  |  |  |
| --- | --- | --- | --- |
| Test | x | y2 | MaxLoad |
| up | \_top | my\_var | sample23 |
|  |  |  |  |

Remember, you can’t start an identifier with a digit. Thus, **12x** is invalid, for example. Good programming practice dictates that you choose identifiers that reflect the meaning or usage of the items being named.

Although you cannot use any of the reserved C# keywords as identifiers, C# does allow you to precede a keyword with an @, allowing it to be a legal identifier. For example, **@for** is a valid identifier. In this case, the identifier is actually **for** and the @ is ignored. Here is a program that illustrates the use of an **@** identifier:

* Demonstrate an @ identifier. using System;
* class IdTest {

static void Main() {

int @if; // use if as an identifier

for(@if = 0; @if < 10; @if++)

Console.WriteLine("@if is " + @if);

}

}

The output shown here proves the **@if** is properly interpreted as an identifier:

@if is 0

@if is 1

@if is 2

@if is 3

@if is 4

|  |
| --- |
| PART I |



Part I: T h e C # L a n g u a g e

@if is 5

@if is 6

@if is 7

@if is 8

@if is 9

Frankly, using @-qualified keywords for identifiers is not recommended, except for

special purposes. Also, the @ can precede any identifier, but this is considered bad practice.