C sharp NET/Capítulo 4

Estructuras de control

Hay dos maneras de cambiar el rumbo de ejecución de un programa, estos pueden ser saltos incondicionales y saltos condicionales. En este capítulo se describen algunas de estas sentencias. Algunas son muy similares a las existentes en otros lenguajes, como las sentencias if, for, while, etc. y otras, como foreach, throw o continue, son algo más específicas.

Saltos incondicionales

Las instrucciones de un programa se ejecutan sentencia por sentencia empezando desde el método o función principal llamado Main() hasta terminar con el programa. El programa sin embargo, tomará otros rumbos incondicionalmente en dos oportunidades: 1. Cuando encuentre la llamada a otros métodos (Ejemplo 4.1) y 2. Con el uso de las palabras claves como goto, break, continue, return y throw las cuales se discutirán más adelante.

Ejemplo 4.1 - Salto incondicional a otra función

```
using System;
namespace Ejemplos{
  class Ejemplo4_1{
    static void Main(){
      Console.WriteLine ("Esta parte se ejecuta primero");
      LlamadaOtraFuncion();
      Console.WriteLine ("Esta parte se ejecuta al final");
    }

    static void LlamadaOtraFuncion(){
      Console.WriteLine ("Ha salido del método Main()");
    }
  }
}
```

En el ejemplo anterior el programa ejecuta sentencia por sentencia el método principal Main() hasta que encuentra la llamada a otro método. Después de que el método llamado haya terminado el método Main continuará con la ejecución de las sentencias restantes.

La sentencia *goto*

En los inicios de los lenguajes de programación la sentencia `goto` fue la más popular para ir de un lugar a otro dentro del programa. Sin embargo esto creaba una tremenda confusión al momento de diseñar la aplicación. Si el programador quería hacer un esquema de como funcionaba dicha aplicación, se veía con un laberinto tipo espagueti de líneas y símbolos conectados entre si. Es por esto que esta sentencia es un poco "problemática" y fuera de "moda" entre los lenguajes de programación modernos. C# sin embargo soporta esta sentencia. Os recomendamos no utilizarla a menos que sea necesario o si os sentís cómodos haciéndolo, pero cuando os cree un laberinto difícil de depurar, no digáis que no os advertimos de no utilizarla. Hay muchas otras mejores maneras de cumplir con el mismo propósito (la sentencia while por ejemplo es una de ellas), las cuales son más elegantes y más sencillas de depurar.

La sentencia `goto` funciona de la siguiente manera:

Primero se crea una etiqueta al inicio de cierto bloque de código y después en otro lugar podemos saltar hacia esa etiqueta usando la palabra clave `goto`. El siguiente ejemplo ilustra la sentencia goto:

```
using System;
namespace Ejemplos
{
   class Ejemplo4_2
   {
      static void Main()
      {
       int contador=0;
       REPETIR:
       Console.WriteLine ("Esta línea se repetirá 100 veces, esta es la linea numero: {0}", contador);
       if (contador++ < 100)
          goto REPETIR;
              Console.WriteLine ("Despues de que el contador sea igual o mayor que 100 se imprimirá esta línea");
      }
   }
}
```

Esta sentencia es un ejemplo de salto incondicional ya que por si solo saltará a la etiqueta seleccionada incondicionalmente.

Saltos condicionales

Los saltos condicionales sirven para ejecutar cierto código solamente si se cumple con alguna condición. Entre otros tenemos:

Instrucción if

Esta sentencia sirve para ejecutar unas instrucciones en caso de que se cumpla determinada condición. La forma completa de la instrucción if es

```
if( condición ) {
  instrucciones;
  ...
}
else {
  instrucciones;
  ...
}
```

donde la cláusula else es opcional. Si la condición es verdadera, se ejecutarán las instrucciones dentro del bloque if, mientras que si es falsa, se ejecutará el bloque else. El valor que controla la sentencia if debe ser de tipo bool. El siguiente ejemplo

```
   //programa que determina si un valor es positivo o negativo
   using System;

   class InstruccionIf{
```

```
        public static void Main()
        {
            double d;

            Console.WriteLine("Introduce un numero");
            d = Double.Parse( Console.ReadLine() );

            if( d>0 )
            {
                Console.WriteLine("El numero {0} es positivo", d);
            }
            else
            {
                Console.WriteLine("El numero {0} es negativo", d);
            }
        }
    }
```

te pide que introduzcas un número y dependiendo de si se cumple que dicho número es mayor que cero (condición), se ejecuta un bloque u otro.

La sentencia d = Double.Parse(Console.ReadLine()); tal vez requiera algo de explicación adicional. En realidad, con Console.ReadLine() estamos leyendo lo que el usuario introduce por pantalla, que es una cadena de caractéres, y con Double.Parse lo que hacemos es interpretar esa cadena de caractéres y convertirla en un tipo numérico double, de forma que tendrá el valor del número que introduzcamos por la consola.

Las intrucciones if se pueden anidar, y existe también una extensión de la sentencia if, la sentencia if-else-if. Su formato es el siguiente:

```
        if( condicion1 )
        {
            instrucciones;
        }
        else if( condicion2 )
        {
            instrucciones;
         }
         ...
         else
         {
            instrucciones;
         }
```

Las instrucciones condicionales se evalúan de arriba a abajo. Tan pronto como se encuentra una condición true, se ejecuta la instrucción asociada con ella, y el resto de la escalera se omite. Si ninguna de las condiciones es true, se ejecutará la última instrucción else. La última instrucción else actúa como condición predeterminada, es decir, si no funciona ninguna de las otras pruebas condicionales, se realiza esta última instrucción. Si no existe esta instrucción else final y el resto de de las condiciones son falsas, entonces no se realizará ninguna acción. El siguiente ejemplo

```
    using System;
```

```
    class IfElseIf{
        public static void Main()
        {
            string opcion;

            Console.WriteLine("Elija una opción (si/no)");
            opcion = Console.ReadLine();

            if( opcion=="si" )
            {
                Console.WriteLine( "Muy bien, ha elegido si" );
            }
            else if( opcion=="no" )
            {
                Console.WriteLine( "Ha elegido no" );
            }
            else{
                Console.WriteLine("No entiendo lo que ha escrito");
            }
        }
    }
```

le pide al usuario que elija una opción si/no y la procesa usando una estructura if-else-if. Si la opción no es ni "si" ni "no", entonces se ejecuta la sentencia else por defecto, que imprime por pantalla el mensaje "No entiendo lo que ha escrito"

Nota: Hay que tener mucho cuidado que el simbolo = no es igual a ==, el primero sirve para asignar un valor a una variable y el segundo sirve para comparar si dos términos son iguales.

Instrucción switch

La instrucción switch es muy parecida a la estructura if-else-if, sólo que permite seleccionar entre varias alternativas de una manera más cómoda. Funciona de la siguiente manera: el valor de una expresión se prueba sucesivamente con una lista de constantes. Cuando se encuentra una coincidencia, se ejecuta la secuencia de instrucciones asociada con esa coincidencia. La forma general de la instrucción switch es la siguiente:

```
        switch( expresión ){
            case constante1:
                instrucciones;
                break;
            case constante2:
                instrucciones;
                break;
            ...
            default:
                instrucciones;
                break;
        }
```

La sentencia default se ejecutará sólo si ninguna constante de las que siguen a case coincide con expresión. Es algo similar al else final de la instrucción if-else-if.

Sin más, vamos a por un ejemplo

```
    using System;

    class InstruccionSwitch{
        public static void Main()
        {
            string s;

            Console.WriteLine( "Elige hacer algo con los números 2 y 3");
            Console.WriteLine( "    + para sumarlos" );
            Console.WriteLine( "    - para restarlos" );
            Console.WriteLine( "    * para multiplicarlos" );
            Console.WriteLine( "    / para dividirlos (division entera)" );

            s = Console.ReadLine();

            switch(s){
                case "+":
                    Console.WriteLine("El resultado es {0}", 2+3);
                    break;
                case "-":
                    Console.WriteLine("El resultado es {0}", 2-3);
                    break;
                case "*":
                    Console.WriteLine("El resultado es {0}", 2*3);
                    break;
                case "/":
                    Console.WriteLine("El resultado es {0}", 2/3);
                    break;
                default:
                    Console.WriteLine("No te entiendo");
                    break;
            }
        }
    }
```

El cual solicita al usuario que inserte uno de los símbolos +-*/ , y con un switch compara los resultados para hacer diferentes acciones dependiendo del valor de s, que es la cadena de caracteres que almacena la elección del usuario. El resultado debería ser algo parecido a esto:

```
Elige hacer algo con los números 2 y 3
    + para sumarlos
    - para restarlos
    * para multiplicarlos
    / para dividirlos (division entera)
*
```

```
El resultado es 6
```

Como habrá notado, al final de todo case siempre hay una sentencia break. Esto no es obligatorio, puede haber en su lugar otra sentencia de salto como un **goto** inclusive en el caso default.

Siempre se deberá tener un break o un goto en cada caso a menos que la sentencia esté vacía. En esta situación se ejecutará el siguiente caso que viene en la lista. Si no se toma en cuenta ésto se obtiene un error en tiempo de compilación. Otros lenguajes, como C/C++ o Java no tienen esta restricción. La razón de adoptarla en C# es doble: por un lado, elimina muchos errores comunes y en segundo lugar permite al compilador reorganizar las sentencias de los case, y así permitir su optimización.

Ejemplo:

```
using System;

class InstruccionSwitch{
    public static void Main()
    {
        int voto;
        Console.WriteLine( "Qué tipo de musica te gusta más");
        Console.WriteLine( "1 - Rock" );
        Console.WriteLine( "2 - Clásica  (clasica cuenta como instrumental)" );
        Console.WriteLine( "3 - Instrumental" );
        Console.WriteLine( "4 - Alternativa (alternativo cuenta como Rock)" );

        voto = Int32.Parse(Console.ReadLine());

        switch(voto){
            case 1:
                Console.WriteLine("Has votado por Rock o Alternativo");
                break;
            case 2: //Debido a que no tiene ni un goto ni break y está vacía va al siguiente caso
            case 3:
                Console.WriteLine("Has votado por Clásica o Instrumental");
                break;
            case 4:
                goto case 1;
            default:
                Console.WriteLine("No te entiendo");
                break;
        }
    }
}
```

Como bien se puede notar en el ejemplo, en el **case 4**, se utiliza la instrucción **goto**, indicando que se vaya al **case 1**, ya que según la lógica del ejemplo, es igual elegir 1 o 4. Nótese que no es necesario usar **break;** en el **case 4** ya que se utilizó goto.

Bucle for

El bucle for de C# es idéntico al encontrado en los lenguajes C/C++ y Java. El formato general es

```
        for( inicialización; condición; iteración )
        {
            instrucciones;
        }
```

Las sentencias de inicialización se ejecutan una vez al principio y sirven principalmente para asignar valores a las variables que servirán de contador. Las sentencias de condición, por su parte, se ejecutan cada vez que el bucle vuelve al principio y sirven para controlar el bucle: éste seguirá realizándose siempre y cuando estas condiciones sean true. Las sentencias de iteración se ejecutan también cada vez que se realiza una nuevo ciclo en el bucle, y sirven para cambiar el estado de las variables que gobiernan las sentencias de condición. Pero todo esto se entiende mejor con un ejemplo

```
    using System;

    class BucleFor{
        public static void Main()
        {
            int i;  //el contador

            for( i = 0; i < 10; i++)
            {
                Console.WriteLine( i );
            }
        }
    }
```

Este ejemplo imprime por pantalla los 10 primero enteros positivos. Es un caso muy simple del bucle for. Por cierto, el operador ++ lo que hace es que añade una unidad a la variable a la que acompaña, de forma que, por ejemplo, 9++ es 10. De esta forma, la variable i se incrementa a cada vuelta.

En el ejemplo anterior, las sentencias de inicialización y de iteración eran únicas, pero esto no tiene por qué ser así, de hecho se pueden utilizar varias sentencias separadas por comas. Por ejemplo, se pueden usar dos variables para controlar el bucle

</pre>

```
   using System;

   class BucleFor2{
       public static void Main()
       {
            int i;
            int j;

            for( i=0, j=10; i<j; i++, j--)
            {
                Console.WriteLine("( {0} , {1} )", i, j);
            }
```

```
        }
    }
```

</pre>

Por su parte, la expresión condicional del bucle for puede ser cualquier expresión que genere un valor booleano. En este caso se ha usado "i<j", pero también hubiera sido válida "i==5", "true" (el bucle se realizará indefinidamente) o "false" (el bucle no se realizará).

Bucle while

El bucle while es un bucle que se realiza mientras se cumpla determinada condición. Tiene la forma

```
            while( condición )
            {
                instrucciones;
            }
```

Donde la condición tiene que ser un valor booleano. Tiene una estructura muy sencilla, así que vamos a ver directamente un ejemplo.

```
    using System;

    class BucleWhile{
        public static void Main()
        {
            int i = 0;
            while( i<10)
            {
                Console.WriteLine( i );
                i = i+1;
            }
        }
    }
```

En el que se realiza lo mismo que en el ejemplo anterior, sólo que ahora con un bucle while.

Bucle do-while

Se trata de una ligera variante del bucle anterior, con la diferencia de que ahora primero se ejecutan las instrucciones y luego se evalúa la condición, de forma que tiene tiene una estructura:

```
            do{
                instrucciones;
            }
            while( condición );
```

El siguiente ejemplo

```
    using System;

    class BucleDoWhile{
        public static void Main()
        {
```

```
            string s = "";

            do
            {
                Console.WriteLine( "Introduce si para salir del bucle" );
                s = Console.ReadLine();
            }
            while( s != "si" );
        }
    }
```

muestra un programa que ejecuta un bucle hasta que el usuario introduce "si". Por cierto, != es lo contrario de ==, es decir, != devuelve true cuando los valores comparados son distintos.

Bucle foreach

El bucle foreach se utiliza para hacer iteraciones sobre elementos de una colección, como pueden ser los enteros dentro de un arreglo de enteros. La sintaxis sigue la siguiente estructura:

```
           foreach( tipo in coleccion )
           {
                instrucciones;
           }
```

Como hemos comentado, el uso más inmediato es iterar sobre un arreglo de números:

```
    using System;

    class BucleForeach{
        public static void Main()
        {
            int[,] arr =  {{1,2},{2,3}};

            foreach( int elem in arr )
            {
                Console.WriteLine( elem );
            }
        }
    }
```

Este ejemplo sólo imprime los valores de una matriz, pero como se puede comprobar mejora mucho la claridad del código comparándolo con una implementación con bucles for como esta

```
    using System;

    class BucleForeach{
        public static void Main()
        {
            int i, j;   //seran los indexadores de la matriz

            int[,] arr =  {{1,2},{2,3}};
```

```
            for(i = 0; i<2; i++ )
            {
                for( j = 0; j<2; j++ )
                {
                    Console.WriteLine( arr[i,j] );
                }
            }
        }
    }
```

Además, es posible utilizar el bucle foreach con cualquier tipo que sea una colección, no solo con arreglos, como veremos más adelante.

Usando *continue* y *break*

continue y *break* son dos palabras clave que nos permiten saltar incondicionalmente al inicio de un bucle (continue) o fuera de un bucle (break) cuando se necesite. Por ejemplo:

```
using System;

class continueBreak
{
   public static void Main()
   {
      for(int i = 0; i<10; i++ )
      {
         if (i==5)
            continue;
         if (i==9)
            break;
         Console.Write("{0},",i);
      }
   }
}
```

Este pequeño programa entrará en un bucle *for* que hará que la variable *i* tome los valores del 1 al 10, pero al llegar al número 5 el bucle saltará incondicionalmente al inicio del bucle sin ejecutar las líneas que siguen más adelante por lo que no ejecutará la línea que imprime en la pantalla el número 5. Cosa similar sucede cuando llega al número 9: el bucle será detenido por el salto incondicional break que romperá el bucle cuando encuentre esta palabra. El resultado será el siguiente:

```
0,1,2,3,4,6,7,8,
```

El bucle saltó la línea que imprime 5 y terminó cuando llegó a 9 gracias a las palabras clave `continue` y `break`.

Fuentes y contribuyentes del artículo

C sharp NET/Capítulo 4 *Fuente*: http://es.wikibooks.org/w/index.php?oldid=174783 *Contribuyentes*: Alanbrito2, Davidcanar, Fseoane, LadyInGrey, MarcoAurelio, 37 ediciones anónimas

Licencia