LAS ESTRUCTURAS

Las estructuras son tipos definidos por el programador y son un conjunto de datos agrupados. Supongamos que programamos una agenda telefónica. Desde luego, la agenda necesita guardar mucha información, por lo que ya hemos pensado en usar arreglos. Pero también necesitamos pensar en los datos que debe contener como: nombre, edad, teléfono. Esto nos lleva a tener tres arreglos. Si necesitamos más datos, entonces más arreglos serán necesarios. Al final es posible que sea un poco complicado administrar tantos arreglos y esto reduce la flexibilidad de nuestro software. Lo mejor sería poder agrupar esa información, de forma tal que solamente tengamos que administrar un solo arreglo. La forma de poder agrupar la información es por medio del uso de las estructuras.

Cuando creamos una estructura, definimos un nuevo tipo y adentro de este tipo podremos colocar datos. Estos datos se conocen como campos. Cada campo está representado por una variable, que puede ser de cualquier tipo. Luego que hemos terminado de definir la estructura podemos crear variables de esta estructura y guardar la información que necesitemos en ellas.

Cómo definir una estructura

Definir una estructura es sencillo. Sin embargo, el primer paso no se lleva a cabo en la computadora. Lo primero que tenemos que hacer es encontrar cuáles son los campos y el tipo que debe guardar la estructura. En el programa la definiremos utilizando el código que comentamos en el siguiente bloque:

EJEMPLO:

```
acceso struct nombre
{
    acceso tipo campo1;
    - - -
    acceso tipo campoN;
}
```

El acceso indica si la estructura puede verse por afuera del ámbito donde ha sido definida o no. Si el acceso es de tipo público, se puede acceder a la estructura por afuera del ámbito. Para esto pondremos como acceso public. Si no deseamos que el exterior pueda acceder a la estructura, entonces la colocamos como privada. Para esto colocamos el acceso como private. Para indicar que definimos una estructura usamos struct seguido del nombre de la estructura. Éste puede ser cualquier nombre válido en C#. No hay que confundir el nombre de la estructura con la o las variables que usaremos de ella. Con este nombre identificaremos el nuevo tipo.

Luego tenemos que crear un bloque de código. Dentro de este bloque definiremos los campos que necesitamos. Éstos se definen igual que las variables, pero es necesario colocar el acceso.

Por ejemplo, la estructura de nuestra agenda puede ser definida de la siguiente forma:

EJEMPLO:

```
public struct Agenda
{
    public String Nombre;
    public int Edad;
    public String Telefono;
}
```

Así de sencilla es la definición. Ya tenemos una estructura llamada **Agenda** que contiene los campos: **Nombre**, **Edad** y **Teléfono**.

Cómo crear una variable del nuevo tipo

Ya que tenemos definida la estructura, es necesario poder definir sus variables para guardar información. Como hemos realizado la declaración de un nuevo tipo, entonces nos será posible crear variables de ésta, y la forma de hacer esto es similar a la de cualquier otro tipo.

Para definir la variable sólo deberemos poner el nombre del tipo seguido del nombre que le daremos a nuestra variable. No debemos olvidar colocar ; (punto y coma) al final de la sentencia. Supongamos que deseamos crear una variable cuyo nombre será **amigo** y es del tipo agenda.

EJEMPLO:

```
Agenda amigo;
```

Ahora, adentro del nuevo tipo de variable declarada llamada **amigo**, se encuentran los campos **Nombre**, **Edad** y **Teléfono**. Es posible declarar varias variables de **Agenda**.

EJEMPLO:

```
Agenda amigo1, amigo2, amigo3;
```

En el caso de necesitar hacer uso de éste, podemos crear un arreglo de la estructura. Este arreglo puede ser del tamaño que necesitemos para nuestra aplicación, y es entonces de esta forma que para la estructura agenda podemos tener un arreglo donde guardamos la información de los amigos, y otro arreglo donde se guardará la información de los clientes.

EJEMPLO:

```
Agenda []amigos=new Agenda[15];
Agenda []clientes=new Agenda[5];
```

Creación de variables de estructura

Podemos definir una variable de la estructura de la forma tradicional, pero también podemos hacerlo por medio **new**. Con ésta última es posible colocar un constructor adentro de la estructura y facilitarnos la introducción de información. Si se define de forma tradicional deberemos inicializar cada campo de la estructura manualmente.

Cómo acceder a los campos de la estructura

Ya tenemos variables de la estructura y sabemos que adentro de éstas se encuentran los campos que guardarán la información. Debemos tener acceso a los campos para poder guardar, leer o modificar los datos. El acceso al campo se lleva a cabo de la siguiente manera:

EJEMPLO:

```
VariableEstructura.Campo
```

Esto quiere decir que primero debemos indicar cuál es la variable con la que deseamos trabajar. Luego necesitamos utilizar el operador punto y seguido de él colocamos el nombre del campo a acceder. Veamos un ejemplo de esto:

EJEMPLO:

```
amigo.Edad  = 25;
```

En este caso decimos que al campo Edad de la variable amigo le colocamos en su interior el valor 25. Podemos utilizar los campos en cualquier tipo de expresión válida, tal y como una variable:

EJEMPLO:

```
if(amigo.Edad>18)
...
diasVividos=amigo.Edad * 365;
...
Console.WriteLine("El nombre es   {0}",amigo.Nombre);
```

Cómo mostrar los campos de la estructura

Algo que necesitaremos constantemente es presentar los campos de la estructura. La manera más evidente de hacerlo es con el método **WriteLine()**. Simplemente mostramos el contenido del campo como si fuera otra variable del programa.

EJEMPLO:

```
Console.WriteLine("La edad es {0}",amigo.Edad);
```

CAMPOS OLVIDADOS

Es posible que tengamos una estructura y que sólo utilicemos determinados campos, sin embargo, los demás campos continúan existiendo. Algunos de éstos pueden inicializarse automáticamente a **0** o **null** si usamos **new** para crear la variable. Pero si se hace de forma tradicional y no se han inicializado, entonces ese campo no tiene instancia y originará problemas si se intenta usar.

Esta forma es sencilla y útil en los casos en los que solamente necesitamos mostrar un campo de la estructura. Sin embargo, de necesitar mostrar todos los campos, deberemos colocar **WriteLine()** para cada campo, o uno solo con una cadena de formato larga. En cualquier caso resulta incómodo.

La mejor forma de mostrar todos los datos contenidos en los campos sería mediante la conversión de la estructura a una cadena. Sabemos que existe un método llamado **ToString()** que hemos usado con las variables numéricas. Sin embargo, C# no se lo puede dar directamente a nuestra estructura porque no puede saber cuáles son los campos que contiene ni cómo los deseamos mostrar. Por esto, cae en nuestra responsabilidad programar el método **ToString()** de nuestra estructura.

EJEMPLO:

```
public struct Agenda
{
      public String Nombre;
      public int Edad;
      public String Telefono;


      public override String ToString()
      {
            StringBuilder sb = new StringBuilder();
           sb.AppendFormat("Nombre:   {0},    Edad:    {1},
                  Telefono:   {2}",      Nombre,   Edad,
                  Telefono);
                   return   (sb.ToString());
      }
}
```

Podemos observar que adentro del bloque de código de la estructura hemos colocado nuestra versión del método **ToString()**. El acceso debe ser público para que se pueda invocar desde el exterior de la estructura. Este método debe regresar un objeto de tipo **String** y no necesita ningún parámetro.

En el interior del método simplemente colocamos el código necesario para darle formato a la cadena y en nuestro caso hacemos uso de **StringBuilder**, aunque es válido usar cualquier otro método. Al finalizar ya tenemos la cadena con la información de nuestros campos y la regresamos por medio de **return.**

Ahora que ya tenemos implementado **ToString()** mostrar la información contenida será tan sencillo como:

EJEMPLO:

```
Console.WriteLine(amigo.ToString());
```

Veamos un ejemplo completo de estos conceptos:

```
public struct Agenda
{
      public String Nombre;
      public int Edad;
      public String Telefono;

      public override String ToString()
      {
            StringBuilder sb = new StringBuilder();
          sb.AppendFormat("Nombre:  {0},  Edad:  {1},
                Telefono:  {2}",  Nombre,  Edad,
                Telefono);
                  return  (sb.ToString());
      }
}
static void  Main(string[] args)
{
      Agenda []amigos=new Agenda[5];
      amigos[1].Edad=25;
      amigos[1].Nombre="Juan";
      amigos[1].Telefono="(555)123-4567";

      Console.WriteLine(amigos[1].ToString());
}
```

Como parte del ejemplo creamos un arreglo de la estructura **Agenda** que se llama **amigos** y tiene 5 elementos, luego, para el elemento en el índice 1 colocamos los **datos**. Para finalizar simplemente imprimimos el elemento 1 del arreglo **amigos**, pero usamos el método **ToString()** programado por nosotros.

Creación de un constructor para la estructura

En el ejemplo anterior hemos visto la manera de cómo cada uno de los campos que componen la estructura ha tenido que inicializarse en forma individual. Esto es correcto y no presenta ningún problema, sin embargo, hay una forma de poder inicializar los campos más fácilmente, sin tantas complicaciones.

Para hacer esto podemos hacer uso de un constructor. El constructor no es otra cosa que un método que nos permitirá llevar a cabo la inicialización de los campos. Sin embargo, este método tiene algunas características especiales. La primera característica es que siempre se llamará igual que la estructura a la que pertenece. La segunda es muy importante: el constructor se invoca automáticamente cuando llevamos a cabo la instanciación de la variable de la estructura. La última característica del constructor es que no tiene tipo. No sólo no regresa nada, no tiene tipo.

Adentro del constructor podemos colocar cualquier código válido de C#, pero es evidente que colocaremos código dedicado a la inicialización de los campos. Veamos un primer ejemplo de cómo podemos crear un constructor. El constructor siempre va adentro del bloque de código de la estructura.

EJEMPLO:

```
public Agenda(String pNombre, int  pEdad,  String
   pTelefono)
{
      //  Llevamos a cabo la asignación
      Nombre=pNombre; Edad=pEdad; Telefono=pTelefono;
}
```

En el código del constructor vemos que el acceso es público. Esto es necesario y siempre debemos dejarlo así. Si observamos luego se coloca directamente el nombre del constructor. El nombre es Agenda, ya que pertenece a la estructura Agenda. A continuación tenemos la lista de parámetros. Los valores pasados como parámetros serán asignados a los campos correspondientes. En la declaración de la variable lo tendremos que usar de la siguiente forma:

EJEMPLO:

```
Agenda amigo=new Agenda("Juan",25,"(555)123-4567");
```

Aquí vemos que declaramos la variable amigo que es de tipo Agenda. Al hacer la instanciación por medio de new vemos que pasamos los parámetros. Éstos serán los datos que quedarán guardados en los campos de la variable amigo. La cadena "Juan" quedaría adentro del campo Nombre, el valor de 25 adentro del campo Edad y la cadena "(555) 123-4567" en el campo Teléfono.

EJEMPLO:

```
public struct Agenda
{
      public String Nombre;
      public int Edad;
      public  String  Telefono;

      public  Agenda(String pNombre, int  pEdad, String
         pTelefono)
      {
            Nombre=pNombre; Edad=pEdad;
            Telefono=pTelefono;
      }
      public  override String  ToString()
      {
             StringBuilder sb  =  new StringBuilder();
            sb.AppendFormat("Nombre:   {0},    Edad:    {1},
                    Telefono:   {2}",      Nombre,   Edad,
                    Telefono);
                     return   (sb.ToString());
      }
}
static void   Main(string[] args)
{
      Agenda amigo=new Agenda("Juan",25,"(555)  123-4567");
```

```
            Console.WriteLine(amigo.ToString());
        }
```

Cómo usar el constructor para validar información

El constructor no solamente puede ser utilizado para colocar la información adentro de los campos, una de las grandes ventajas que éste nos da es la posibi- lidad de poder validar la información antes de que sea asignada a uno de sus cam- pos. Es posible que existan ciertas reglas sobre lo que se considera información válida para nuestros campos, y por supuesto, podemos implementar estas mis- mas en el constructor.

Supongamos que el teléfono sólo se considera válido si tiene más de 8 caracteres. En caso de que sea menor es inválido y no debe ser asignado para evitar proble- mas. Esta simple regla es fácil de programar en el constructor. En nuestro caso podría quedar de la siguiente forma:

EJEMPLO:

```
public Agenda(String pNombre, int  pEdad, String pTelefono)
   {
         Nombre=pNombre;
         Edad=pEdad;

         if(pTelefono.Length>8)
               Telefono=pTelefono;
         Else
               Telefono=”Teléfono  no válido”;
```

Como vemos, hacemos uso de **if** y verificamos la longitud de la cadena **pTelefono**. Si es mayor a 8 entonces le asignamos **pTelefono** al campo **Telefono**. En caso contrario, colocamos un mensaje que dice que el teléfono no es válido.

EJEMPLO:

```
public struct Agenda
{
   public String Nombre;
   public int Edad;
   public String Telefono;
   public Agenda(String pNombre, int pEdad,  String pTelefono)
   {
         Nombre=pNombre;
         Edad=pEdad;

         if(pTelefono.Length>8)
         Telefono=pTelefono;
               Else
                     Telefono=”Telefono  no valido”;

         public override String ToString()
         {
               StringBuilder sb = new StringBuilder();
         sb.AppendFormat(“Nombre: {0}, Edad: {1}, Telefono: {2}”,
         Nombre, Edad,Telefono);
               return   (sb.ToString());
```

```
        }
    }
    static void   Main(string[] args)
    {
        Agenda amigo=new Agenda("Juan",25,"(555)  123-4567");
        Agenda amigo1=new Agenda("Pedro",32,"(555)");
        Console.WriteLine(amigo.ToString());
        Console.WriteLine(amigo1.ToString());
}
```