por Nicolás Arrioja Landa Cosio





**>>** DESARROLLO / MICROSOFT

**>>** DESARROLL 

**>>** 400 PÁGINAS 

**>>** 400 PÁGINA

**>>** ISBN 978-987-1347-74-2 **>>** ISBN 978-98

**>>** DESARROLLO / INTERNET 

**>>** DESARROLL

**>>** 400 PÁGINAS 

**>>** 400 PÁGINA

**>>** ISBN 978-987-1347-70-4 **>>** ISBN 978-98



**>>** DESARROLLO / INTERNET

**>>** DESARROLL 

**>>** 368 PÁGINAS 

**>>** 368 PÁGINA

**>>** ISBN 978-987-663-039-9 **>>** ISBN 978-98

**>>** DESARROLLO 

**>>** DESARROLL

**>>** 336 PÁGINAS 

**>>** 336 PÁGINA

**>>** ISBN 978-987-1347-97-1 **>>** ISBN 978-98

**C#**

**GUÍA TOTAL**

**DEL PROGRAMADOR** por Nicolás Arrioja Landa Cosio



**TÍTULO**: C#

**AUTOR**: Nicolás Arrioja Landa Cosio

**COLECCIÓN**: Manuales USERS

**FORMATO**: 17 x 24 cm

**PÁGINAS**: 400

Copyright © MMX. Es una publicación de Fox Andina en coedición con Gradi S.A. He cho el depósito que marca la ley 11723. Todos los derechos reservados. No se permi te la reproducción parcial o total, el almacenamiento, el alquiler, la transmisión o la transformación de este libro, en cualquier forma o por cualquier medio, sea electró nico o mecánico, mediante fotocopias, digitalización u otros métodos, sin el permiso previo y escrito del editor. Su infracción está penada por las leyes 11723 y 25446. La editorial no asume responsabilidad alguna por cualquier consecuencia derivada de la fabricación, funcionamiento y/o utilización de los servicios y productos que se des criben y/o analizan. Todas las marcas mencionadas en este libro son propiedad ex clusiva de sus respectivos dueños. Impreso en Argentina. Libro de edición argentina. Primera impresión realizada en Sevagraf, Costa Rica 5226, Grand Bourg, Malvinas Argentinas, Pcia. de Buenos Aires en VIII, MMX.

ISBN 978-987-26013-5-5

Landa Cosio, Nicolás Arrioja

C#. - 1a ed. - Buenos Aires : Fox Andina; Gradi; Banfield - Lomas de Zamora, 2010. 400 p. ; 24x17 cm. - (Manual Users; 195)

ISBN 978-987-26013-5-5

1. Informática. I. Título

CDD 005.3

PRELIMINARES

Nicolás Arrioja Landa Cosio

Catedrático de **IUNIVERTECH** y de la Universidad Madero, ele gido el mejor profesor de los ciclos 2006-2007 y 2007-2008, se de dica a la investigación, consulta y capacitación en áreas relacionadas con la realidad virtual, la visualización científica y los videojuegos. En 1997, desarrolló el primer videojuego de realidad virtual in mersiva en Latinoamérica, conocido como **VRaptor**. También desarrolló el primer lenguaje de programación para realidad vir tual en Latinoamérica **CND-VR**, que es usado en el medio académico para enseñar de forma sencilla los conceptos de pro 

gramación de gráficas 3D. Ha sido catedrático en algunas de las universidades más importantes de México durante más de diez años. En esas instituciones enseñó des de las bases de programación en lenguaje C hasta Inteligencia Artificial. Ha otor gado más de 25 conferencias relacionadas con el desarrollo de la realidad virtual y los videojuegos en diversas universidades de México. Tiene una patente referida a interfaces cinemáticas para video juegos. Diseñó el plan y el programa de estudios para la Maestría en Realidad Virtual y Video Juegos en IUNlVERTECH. Es au tor del los libros DirectX, Inteligencia Artificial y Robótica Avanzada de esta mis ma editorial. Actualmente, realiza una investigación sobre Inteligencia Artificial no-Algorítmica y tiene disposición a realizar investigación y dar conferencia en es tas áreas en cualquier parte del mundo.

Dedicatoria

A María Eugenia Pérez Duarte por todo su apoyo y gran amistad.

Agradecimientos

A todas aquellas personas que hace 25 años me introdujeron a la programación de computadoras y me enseñaron las bases de esta disciplina. De **Sistem Club** al Ing. Alfonso Fernández de la Fuente y a Jacky, Annie y David Fox, Herbert Schildt y Kris Jamsa. Un agradecimiento especial a mis alumnos y al equipo de la editorial.

4

www.redusers.com 

Prólogo

**PRÓLOGO**

La facilidad con que hoy en día se tiene acceso a una computadora ha llevado a las per sonas a emplearlas en gran cantidad de actividades, usando para ello software que rea liza tareas específicas. Cuando se requiere que la computadora realice otro tipo de ac ciones y los programas con los que se cuenta no lo hacen, es necesario decirle en su pro pio idioma cómo deberá actuar, y para esto se recurre a los lenguajes de programación. Los lenguajes de programación intentan parecerse a nuestra propia forma de expre sarnos, para ello cuentan con características propias de sintaxis y semántica, así co mo símbolos que permiten decirle a la computadora qué procesos y qué datos de be manejar, cómo almacenarlos, qué acciones realizar con ellos y cómo presentarlos al usuario. Por esta razón es importante contar con una metodología que permita aprender hábitos correctos y efectivos para resolver problemas, los cuales, al ser sub divididos para su mejor comprensión, facilitarán su resolución.

Una vez que se ha comprendido el problema, se construye una secuencia de los pa sos a realizar, mejor conocida como algoritmo, el cual permitirá probar si la lógica es correcta y si los resultados que se obtendrán serán los adecuados. El potencial de los actuales lenguajes de programación es enorme, ya que permiten automatizar muy diversos tipos de actividades, disminuyendo de ese modo el tiempo que se invierte al hacerlo de forma manual y, si se aprende a usarlos correctamente, se podrá hacer más sencilla la vida de los usuarios que emplean dichos programas.

La experiencia adquirida en el desarrollo de sistemas por parte del Dr. Nicolás Arrioja lo ha convertido en una autoridad en cuanto a programación y buenas prácticas, com partiéndolas y enseñando a alumnos de diferentes instituciones de nivel superior, con resultados excelentes. Es por eso que esta nueva versión del libro podrá guiar paso a pa so al lector en el uso del poderoso lenguaje de programación C# del Visual Studio 2010, de modo que al finalizar su aprendizaje, esté listo para hacer sus propios programas.

Aprender a programar es una tarea que requiere práctica y paciencia, pero tam bién es una actividad fascinante, y si además se cuenta con un excelente libro y la constancia necesaria, los resultados serán excelentes. ¡Felicidades por este nue vo reto que está por comenzar!

María Eugenia Pérez Duarte

*Maestra en Administración de Tecnologías de Información*

*Coordinadora de Ingenierías en Sistemas*

*Computacionales y Desarrollo de Software*

*Universidad Madero de Méjico*

**www.redusers.com 5

PRELIMINARES

**EL LIBRO DE UN VISTAZO**

El desarrollo de aplicaciones con C# se ha visto modificado en los últimos tiempos, y este libro apunta a afianzar los conocimientos indispensables que nos permitirán generar aplicaciones profesionales de nivel. Cada capítulo está dedicado a una técnica específica e ilustrado mediante ejemplos prácticos listos para implementar.

**Capítulo 1**

**C# Y .NET**

Este capitulo nos introduce a la historia del desarrollo de los programas para Windows y cómo el Framework de .NET ayuda a resolver muchos problemas. Aprenderemos cómo crear y ejecutar nuestro primer programa.

**Capítulo 2**

**LOS ELEMENTOS BÁSICOS DE UN**

**PROGRAMA**

Aprender cómo resolver problemas en la computadora es aún más importante que conocer la sintaxis de un lenguaje. Aquí aprenderemos los elementos básicos de un programa, pero también la metodología para diseñar buenos programas desde el inicio.

**Capítulo 3**

**EL PROGRAMA TOMA DECISIONES** Los programas de computadora necesitan lógicas complejas, por lo que es importante conocer las estructuras necesarias que nos permitirán decirle a la computadora cómo tomar una decisión.

**Capítulo 4**

**CREACIÓN DE CICLOS**

Los ciclos nos permiten repetir algo un número de veces. Los diferentes tipos de ciclos, sus partes y aplicaciones serán enseñados en este capítulo.

**Capítulo 5**

**FUNCIONES Y MÉTODOS**

Las funciones y los métodos nos brindan flexibilidad y la capacidad de poder volver a utilizar código fácilmente, y también nos ayudan a ordenar nuestros desarrollos, y facilitan su mantenimiento.

**Capítulo 6**

**LOS ARREGLOS**

Los arreglos, muy útiles en todos los casos, nos permiten administrar la información de una manera fácil y práctica, como si trabajásemos con una planilla de cálculo. En este capítulo estudiaremos arreglos de una y dos dimensiones, y por supuesto no dejaremos de lado los arreglos de tipo jagged.

**Capítulo 7**

**LAS COLECCIONES**

Las colecciones son estructuras de datos que nos permiten trabajar con grupos de información. Veremos aquellas más importantes, como el ArrayList, Stack, Queue y Hashtable.

**Capítulo 8**

**LAS CADENAS**

Las cadenas permiten guardar y manipular texto o frases. C# nos provee de éstas para su manejo. Aprenderemos cómo hacer uso de las cadenas y trabajar con ellas.

6 www.redusers.com 

**Capítulo 9**

**ESTRUCTURAS Y ENUMERACIONES**

El libro de un vistazo

**Capítulo 11**

**FLUJOS Y ARCHIVOS**

En este capítulo aprenderemos cómo crear nuestros propios tipos por medio de las estructuras y enumeraciones. Las estructuras también nos permiten agrupar la información de una manera óptima, lo cual nos ayuda a manejarla más fácilmente.

**Capítulo 10**

**CÓMO CREAR NUESTRAS PROPIAS CLASES** En este capítulo empezaremos a conocer los conceptos de la programación orientada a objetos, lo más utilizado hoy en día. Podremos crear nuestras propias clases y nuestros propios objetos, y también trabajaremos con métodos definidos en las clases.

Es muy frecuente que necesitemos guardar información desde un programa al disco. Por medio de los flujos y los archivos lo podemos hacer. Guardaremos y leeremos cadenas, y veremos que la manipulación de archivos también es posible.

**Capítulo 12**

**DEPURACIÓN**

La depuración nos permite corregir los problemas que tiene un programa. Conoceremos los tipos de errores y cómo usar el depurador para eliminarlos. También aprenderemos cómo administrar las excepciones de la mejor manera .

!

INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual encontrará una serie de recuadros que le brindarán información com plementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados. Cada recuadro está identificado con uno de los siguientes iconos:

CURIOSIDADES E IDEAS



ATENCIÓN SITIOS WEB

DATOS ÚTILES

Y NOVEDADES

www.redusers.com

7

PRELIMINARES

**CONTENIDO**

Sobre el autor 4 Prólogo 5 El libro de un vistazo 6 Introducción 12

**~~Capítulo 1~~**

**C# Y .NET**

Breve historia de la programación para Windows 14 Descubrir .NET 15 Cómo crear una aplicación .NET 17 Conseguir un compilador de C# 18 El ambiente de desarrollo 18 Crear nuestra primera aplicación 19 Resumen 29 Actividades 30



**Capítulo 2**

**LOS ELEMENTOS BÁSICOS**

**DE UN PROGRAMA**

Los lenguajes de programación 32 Los programas de computadora 32 Las variables 42 Operaciones aritméticas 49 Cómo pedirle datos al usuario 56 Cómo resolver problemas

en la computadora 60 Resolución de problemas

en la computadora 64 Resumen 69 Actividades 70

Inicio 





Pedir ancho



Pedir alto



Área = ancho • alto

Perímetro = 

(ancho + alto) • 2



Mostrar área



Mostrar

perímetro





Fin

8 www.redusers.com 

Contenido

**Capítulo 3**

**EL PROGRAMA TOMA DECISIONES** La toma de decisiones 72 Expresiones relacionales 72 El uso de if 76 El uso de else 84 Cómo usar if anidados 89 Escalera de if-else 92 Expresiones lógicas 96 El uso de switch 105 Resumen 109 Actividades 110

**~~Capítulo 5~~**

****Código 

Condición NO

SÍ

Inicio 



Pedir dividendo

Pedir divisor



Divisor!=0SÍ

Resultado =

**FUNCIONES Y MÉTODOS**

Las funciones 148 Funciones que ejecutan código 150 Funciones que regresan un valor 156 Funciones que reciben valores 159 Funciones que reciben parámetros

y regresan un valor 162 Optimizar con funciones 171

NO

Fin

**~~Capítulo 4~~**

**CREACIÓN DE CICLOS**

dividendo/divisor 

Mostrar resultado

Paso por copia y paso por referencia 178 Resumen 185 Actividades 186

Inicio 





Pedimos a



El ciclo for 112 El valor de inicio 117 El límite de conteo del ciclo 119 Control del incremento 121 Ejemplos con el ciclo for 128 El ciclo do while 134 El ciclo while 141 Resumen 145 Actividades 146

Pedimos b R=a+b Mostramos r Fin

www.redusers.com 9

PRELIMINARES

**Capítulo 6**

**LOS ARREGLOS**

Los arreglos 188 Declaración de los arreglos

de una dimensión 189 Asignación y uso de valores 191 Arreglos de dos dimensiones 196 Arreglos de tipo jagged 205 Los arreglos como parámetros

a funciones 212 Resumen 215 Actividades 216

**~~Capítulo 7~~**

**LAS COLECCIONES**

Las colecciones más importantes 218 El ArrayList 218 El Stack 232 El Queue 241 El Hashtable 249 Resumen 253 Actividades 254

Inicio 





Inicialización



**Capítulo 8**

**LAS CADENAS**

El uso de las cadenas 256 Cómo declarar la cadena 256 El método ToString() 256 Cómo convertir y formatear

fechas a cadenas 257 Para darles formato a valores

numéricos 259 Cómo concatenar cadenas 260 Uso de StringBuilder 261 Comparación de cadenas 263 Para encontrar una subcadena 264 Para obtener una subcadena 265 Para determinar si una cadena

finaliza en una subcadena 266 Cómo copiar una parte de la cadena 267 Cómo insertar una cadena 268 Para encontrar la posición de una

subcadena 269 Justificación del texto 270 Para eliminar caracteres de la cadena 271 Cómo reemplazar una subcadena 271 Cómo dividir la cadena 272 Intercambiar mayúsculas y minúsculas 273 Cómo podar la cadena 275 Resumen 277 Actividades 278

Condición 

SÍ



NO Código 

'H' 'o' 'l' 'a' 'H' 'o' 'l' 'a'

'' 'm' 'u' 'n' 

Incremento Fin

'm' 'u' 'n' 'd' 'o' 'r' 'e' 'd' 'n''o' 'o''d'

10 www.redusers.com 

Contenido

**Capítulo 9**

**ESTRUCTURAS Y ENUMERACIONES** Las estructuras 280 Cómo definir una estructura 280 Cómo crear una variable del nuevo tipo 282 Cómo acceder a los campos

de la estructura 283 Cómo mostrar los campos

de la estructura 283 Creación de un constructor

para la estructura 286 Estructuras enlazadas 299 Las enumeraciones 309 Resumen 315 Actividades 316



Agenda Nombre: String



Dirección: String

Teléfono: String

Edad: Int



**~~Capítulo 10~~**

**CÓMO CREAR NUESTRAS PROPIAS CLASES** La programación orientada a objetos 318 Las clases 318 Cómo declarar la clase y los datos 320 Cómo crear la instancia

de nuestra clase 324 Cómo asignarles valores

a datos publicos 324 Cómo invocar métodos de un objeto 325 Cómo imprimir un dato público 325 Protección de datos

y creación de propiedades 329 Cómo acceder a las propiedades 333 Métodos públicos y privados 333

Convertir un objeto a cadena 334 El constructor en las clases 336 Sobrecarga del constructor 339 Resumen 347 Actividades 348

Cubo 

+ lado: int

+ área: int

+ volumen: int

+ CalculaArea() : void

+ CalculaVolumen() : void

**~~Capítulo 11~~**

**FLUJOS Y ARCHIVOS**

Los flujos 350 Los stream en la memoria 351 El uso de archivos 363 Resumen 371 Actividades 372

P. Actual

Tamaño

HOLA A TODO Capacidad

**~~Capítulo 12~~**

**DEPURACIÓN**

Cómo empezar a depurar un programa 374 Corregir los errores de compilación 374 Cómo corregir los errores

en tiempo de ejecución 379 Cómo manejar los errores 383 Resumen 391 Actividades 392

www.redusers.com 11

PRELIMINARES

**INTRODUCCIÓN**

La programación de computadoras es una actividad fascinante, el poder hacer una aplicación a la medida y de acuerdo a nuestras necesidades es una actividad interesante y llena de recompensas. Mucha gente llega a pensar que solamente con aprender la sintaxis de un lenguaje de programación es suficiente para po der hacer programas, pero se equivocan. Aun más importante que la sintaxis del lenguaje es la técnica de resolución del problema y el análisis adecuado. El tener buenos hábitos de análisis y programación desde el inicio es fundamental para convertirnos en buenos programadores.

Una de los lenguajes que está adquiriendo gran popularidad es C#, un lenguaje sen cillo, amigable y poderoso. Con C# se pueden crear todo tipo de aplicaciones, des de programas de consola para Windows y para páginas web en unión con ASP, has ta video juegos para Xbox 360 con XNA.

El presente libro introduce a cualquier persona, aun sin conocimientos de progra mación, al lenguaje C#. Esta obra está actualizada para Visual Studio 2010 y el Fra mework de .NET 4.0. En un inicio veremos la forma de realizar el análisis de un problema e iremos construyendo el conocimiento hasta llegar a las bases de la pro gramación orientada a objetos y, por supuesto, la depuración.

El convertirse en un buen programador requiere de mucha práctica y experi mentación, pero también de una guía adecuada que vaya construyendo paso a paso las habilidades necesarias. Esperamos que este libro sea de utilidad a todos aquellos que, aun sin experiencia en programación, tengan el deseo de aprender como hacer sus propios programas.

Nicolás Arrioja Landa Cosio

12 www.redusers.com 

C#

C#.NET

Este libro está dirigido a todas aquellas personas que desean aprender

el lenguaje de programación C# y tienen conocimientos básicos de programación o no tienen ninguna experiencia previa programando computadoras. El enfoque del libro es guiar paso a paso

en el aprendizaje del lenguaje y aunque C# es un lenguaje orientado a objetos, en los primeros capítulos tomaremos una filosofía estructurada con el fin de hacer todavía más fácil el aprendizaje para todos nosotros.

**SERVICIO DE ATENCIÓN AL LECTOR: usershop@redusers.com**

Capítulo 1

**Breve historia de la programación para Windows 14** Comprendiendo .NET 15 Cómo se crea una aplicación .NET 16 Cómo conseguir un compilador de C# 17 El ambiente de desarrollo 18 Cómo crear nuestra primera

aplicación 19 **Resumen 29 Actividades 30**

1. C#.NET

BREVE HISTORIA DE LA PROGRAMACIÓN PARA WINDOWS

Hace algunos años la única forma como se podía programar para Windows era hacer uso de un compilador de C o C++ y de un **API** de Windows. El API es una gran colección de funciones que se relacionan, las que nos permiten comu nicarnos con el sistema operativo. Por medio del API de **Win32** se programaban las ventanas, botones y demás elementos.

El problema de este tipo de programación es que el API de Win32 es realmente com plejo y enorme. Con miles de funciones en su interior, por lo que pocos programa dores podían conocerlo en su totalidad. Pero la complejidad no solamente estaba en la cantidad de funciones, también en la sintaxis y la forma como se programa.

Para facilitar la programación de aplicaciones para Windows surgen diferentes op ciones; la finalidad de estos intentos era poder hacer las aplicaciones sin tener que pasar por la complejidad de Win32. Uno de estos intentos fue conocido como **OWL**; sin embargo, obtuvo más éxito **MFC**, creado por Microsoft.

MFC es un conjunto de clases que envuelve a Win32 y facilita su programación. Con MFC los procesos más comunes se agrupan en funciones de tal forma que con una simple llamada a una función de MFC se puede hacer una determinada tarea, para la que antes necesitábamos por lo menos 10 llamadas en Win32 y muchos pa rámetros. Sin embargo Win32 está debajo de MFC; la programación MFC simpli fica mucho las cosas, pero muchos programadores que venían del paradigma de pro gramación estructurada no se sentían a gusto con él.

Otra de las opciones que surgieron es **Visual Basic**, este lenguaje logró gran popu laridad, especialmente en Latinoamérica. Visual Basic también trabaja por arriba de Win32, pero basa su sintaxis en el antiguo lenguaje **Basic**. Es muy sencillo de apren der y una de las características que le dio gran popularidad fue la facilidad con la que se podían crear interfaces de usuario y conectividad a bases de datos. Pero has ta antes de la versión .NET, este lenguaje tenía ciertos limitantes ya que no se po día llevar a cabo programación orientada a objetos con él.

Otro lenguaje que surge, pero con su propio **Framework**, es **JAVA**; su principal ventaja es ser multiplataforma. Una de sus características es el uso de un **runtime**, la aplicación en lugar de correr directamente en el microprocesador, se ejecuta en un programa llamado runtime y este se encarga de ejecutar el código en el mi croprocesador correspondiente. Si se tiene el runtime para Windows, sin pro blema se ejecuta el programa de JAVA.

Cuando nosotros deseábamos tener un programa que se pudiera ejecutar, era necesa rio **compilarlo**. Cada uno de los lenguajes tenía su propio **compilador**, por ello no era sencillo poder compartir código de C++ con código de Visual Basic ya que el traducir entre lenguajes era difícil. Para poder compartir código entre los lenguajes surge un mo

14 www.redusers.com 

Breve historia de la programación para Windows

delo conocido como **COM**, éste nos permite crear componentes binarios, esto quiere decir que es posible programar un componente en Visual Basic y un programador de C++ puede tomarlo y hacer uso de él. Esto se debe a que el componente ya es código compilado y no código fuente en el lenguaje de origen; la programación de COM tam bién tenía sus complejidades y surge **ATL** para ayudar en su desarrollo.

Con todo esto, llega el momento en el cual es necesario ordenar, facilitar y organi zar el desarrollo de las aplicaciones para Windows, con esta filosofía surge .NET.

Descubrir .NET

El **Framework** de **.NET** es una solución a toda la problemática en torno al desa rrollo de aplicaciones, brinda grandes beneficios no solamente al desarrollador, si no también al proceso de desarrollo. En primer lugar .NET permite trabajar con código ya existente, podemos hacer uso de los componentes COM, e incluso, si lo necesitáramos usar el API de Windows. Cuando el programa .NET está listo es mu cho más fácil de instalar en la computadora de los clientes, que las aplicaciones tra dicionales ya que se tiene una integración fuerte entre los lenguajes.

Un programador de C# puede entender fácilmente el código de un programador de Visual Basic .NET y ambos pueden programar en el lenguaje con el que se sienten más cómodos. Esto se debe a que todos los lenguajes que hacen uso de .NET com parten las librerías de .NET, por lo que no importa en qué lenguaje programemos, las reconocemos en cualquiera. A continuación conoceremos los diferentes compo nentes de .NET: **CLR**, **assembly** y **CIL**.

CLR

El primer componente de .NET que conoceremos es el **Common Language Runtime**, también denominado **CLR**. Este es un programa de ejecución común a todos los lenguajes. Este programa se encarga de leer el código generado por el com pilador y empieza su ejecución. Sin importar si el programa fue creado con C#, con Visual Basic .NET o algún otro lenguaje de .NET el CLR lo lee y ejecuta.

Assembly

Cuando tenemos un programa escrito en un lenguaje de .NET y lo compilamos se ge nera el **assembly**. El assembly contiene el programa compilado en lo que conocemos como CIL y también información sobre todos los tipos que se utilizan en el programa.

CIL

Los programas de .NET no se compilan directamente en código ensamblador del compilador, en su lugar son compilados a un lenguaje intermedio conocido como CIL. Este lenguaje es leído y ejecutado por el runtime. El uso del CIL y el runtime es lo que le da a .NET su gran flexibilidad y su capacidad de ser multiplataforma.

www.redusers.com 15

1. C#.NET

El Framework de .NET tiene lo que se conoce como las **especificaciones comunes de lenguaje** o **CLS** por sus siglas en inglés, estas especificaciones son las guías que cual quier lenguaje que desee usar .NET debe de cumplir para poder trabajar con el runti me. Una ventaja de esto es que si nuestro código cumple con las CLS podemos tener interoperabilidad con otros lenguajes, por ejemplo, es posible crear una librería en C# y un programador de Visual Basic .NET puede utilizarla sin ningún problema.

Uno de los puntos más importantes de estás guías es el **CTS** o **sistema de tipos comunes**. En los lenguajes de programación, cuando deseamos guardar informa ción, ésta se coloca en una variable, las variables van a tener un tipo dependiendo del la información a guardar, por ejemplo, el tipo puede ser para guardar un nú mero entero, otro para guardar un número con decimales y otro para guardar una frase o palabra. El problema con esto es que cada lenguaje guarda la información de manera diferente, algunos lenguajes guardan los enteros con **16 bits** de memo ria y otros con **32 bits**; incluso algunos lenguajes como C y C++ no tienen un ti po para guardar las **cadenas** o frases.

Para solucionar esto el Framework de .NET define por medio del CTS cómo van a funcionar los tipos en su entorno. Cualquier lenguaje que trabaje con .NET de be de usar los tipos tal y como se señalan en el CTS. Ahora que ya conocemos los conceptos básicos, podemos ver cómo es que todo esto se une.

Cómo se crea una aplicación .NET

Podemos crear una aplicación .NET utilizando un lenguaje de programación, para este efecto será C#; con el lenguaje de programación creamos el código fuente del programa (instrucciones que le dicen al programa qué hacer). Cuando hemos finalizado con nuestro código fuente, entonces utilizamos el com pilador. El compilador toma el código fuente y crea un assembly para nosotros, es te assembly tendrá el equivalente de nuestro código, pero escrito en CIL; esto nos lleva a otra de las ventajas de .NET: nuestro código puede ser optimizado por el compilador para la plataforma hacia la cual vamos a usar el programa, es decir que el mismo programa puede ser optimizado para un dispositivo móvil, una PC nor mal o un servidor, sin que nosotros tengamos que hacer cambios en él.

.NET ES MULTIPLATAFORMA

El Framework de .NET se puede ejecutar en muchas plataformas, no solo en Windows. Esto significa que podemos programar en una plataforma en particular y si otra plataforma tiene el runtime, nuestro programa se ejecutará sin ningún problema. Un programa .NET desarrollado en Windows puede ejecutarse en Linux, siempre y cuando se tenga el runtime correspondiente.

16 www.redusers.com 

Breve historia de la programación para Windows

Cuando nosotros deseamos invocar al programa, entonces el runtime entra en ac ción, lee el assembly y crea para nosotros todo el entorno necesario. El runtime empieza a leer las instrucciones CIL del assembly y conforme las va leyendo las compila para el microprocesador de la computadora en la que corre el programa; esto se conoce como **JIT** o **compilación justo a tiempo**. De esta manera confor me se avanza en la ejecución del programa se va compilando; todo esto ocurre de manera transparente para el usuario.

El Framework de .NET provee, para los programas que se están ejecutando, los servicios de **administración de memoria** y **recolector de basura**. En lenguajes no administrados como C y C++ el programador es responsable de la adminis tración de memoria, en programas grandes esto puede ser una labor complicada, que puede llevar a errores durante la ejecución del programa. Afortunadamente lenguajes administrados como C# tienen un modelo en el cual nosotros como programadores ya no necesitamos ser responsables por el uso de la memoria. El recolector de basura se encarga de eliminar todos los objetos que ya no son ne cesarios, cuando un objeto deja de ser útil el recolector lo toma y lo elimina. De esta forma se liberan memoria y recursos.

El recolector de basura trabaja de forma automática para nosotros y ayuda a elimi nar toda la administración de recursos y memoria que era necesaria en Win32. En algunos casos especiales como los archivos, las conexiones a bases de datos o de red se tratan de recursos no administrados, para estos casos debemos de indicar explíci tamente cuando es necesario que sean destruidos.

Cómo conseguir un compilador de C#

Existen varias opciones de compiladores para C#, en este libro utilizaremos la ver sión de C# que viene con **Visual Studio 2010**, pero también es posible utilizar ver siones anteriores. Este compilador, al momento de publicación de este libro, utili za la versión 4.0 del Framework. La mayoría de los ejemplos deben poder compi larse y ejecutarse sin problema, aún utilizando versiones anteriores del Framework, esto es válido incluso para aquellos programadores que decidan trabajar con el lla mado Mono, como alternativa libre y gratuita.

EL COMPILADOR JIT

Al **compilador JIT** también se le conoce como **Jitter**. Forma parte del runtime y es muy eficiente, si el programa necesita volver a ejecutar un código que ya se ha compilado, el Jitter en lugar de volver a compilar, ejecuta lo ya compilado, mejorando de esta forma el desempeño y los tiempos de respuesta de cara al usuario.

www.redusers.com 17

1. C#.NET

Una mejor opción, en caso de que no podamos conseguir la versión profesional de Visual Studio 2010, es la versión **Express**. Esta versión es gratuita y se puede des cargar directamente de Internet, lo único que se necesita es llevar a cabo un peque ño registro por medio de cualquier cuenta de **Hotmail** o **passport**.

Para descargar el compilador de **C# Express** de forma completamente gratuita, po demos acceder al sitio web que se encuentra en la dirección **www.microsoft. com/express**, dentro del portal de Microsoft. Para realizar esta descarga sólo será necesario completar un formulario de registro.

Una vez que hemos descargado el compilador, podemos proceder a llevar a cabo la instalación; esta tarea es muy similar a la instalación de cualquier otro programa de Windows, pero no debemos olvidar registrarlo antes de 30 días.

El ambiente de desarrollo

Una vez finalizada la instalación podemos iniciar el programa seleccionándolo desde el menú **Inicio** de Windows, veremos una ventana como la que aparece a continua ción, que muestra la interfaz de uso de la aplicación.

*Figura 1. Aquí vemos la ventana principal de C# Express*

*2010. Esta interfaz aparece cada vez que iniciamos el programa.*

Básicamente la interfaz está dividida en dos partes, en el centro tenemos la página de inicio, aquí encontramos enlaces a nuestros proyectos más recientes, pero también

18 www.redusers.com 

Breve historia de la programación para Windows

podemos recibir comunicados y noticias sobre el desarrollo de software usando C#. En esta misma parte de la interfaz pero como páginas diferentes, tendremos las zo nas de edición. Estas zonas de edición nos permitirán editar el código del progra ma, editar la interfaz de usuario, iconos y demás recursos.

Editar el código del programa es una actividad muy sencilla que no debe preocu parnos, si sabemos utilizar cualquier tipo de editor de textos como **Microsoft Word** o el **Bloc de notas de Windows**, entonces podemos editar el código del programa. Los demás editores también son bastante amigables para el usuario, pero no los ne cesitaremos para realizar los ejemplos mostrados en este libro.

Del lado derecho tenemos una ventana que se conoce como **Explorador de solucio nes**, en esta misma área aparecerán otras ventanas que nos dan información sobre el proyecto que estamos desarrollando. Veremos los detalles mostrados por algunas de estas ventanas un poco más adelante en este libro.

En la parte inferior encontramos otra ventana, en esta zona generalmente aparecen los apartados que utilizará el compilador para comunicarse con nosotros. Por ejem plo, en esta zona veremos la ventana que nos indica los errores que tiene nuestro programa, en la parte superior aparecen los menús y las barras de herramientas.

Cómo crear nuestra primera aplicación

Para familiarizarnos más con C# Express, lo mejor es crear un primer **proyecto** y trabajar sobre él. Generaremos un pequeño programa que nos mande un mensaje en la consola, luego adicionaremos otros elementos para que podamos explorar las diferentes ventanas de la interfaz de usuario.

Para crear un proyecto tenemos que seleccionar el menú **Archivo** y luego de esto de bemos hacer clic en la opción llamada **Nuevo Proyecto**, de esta forma veremos un cuadro de diálogo que muestra algunas opciones relacionadas.

En la parte central del cuadro de diálogo aparecen listados los diferentes tipos de proyectos que podemos crear, para este ejemplo debemos seleccionar el que indica aplicación de consola. En la parte inferior escribiremos el nombre de nuestro pro yecto, que en este caso será **MiProyecto**. Cada nuevo proyecto que crearemos tendrá su propio nombre; después, simplemente oprimimos el botón **OK**.

ENCONTRAR CÓDIGO FUENTE RÁPIDAMENTE

Si por algún motivo llegamos a cerrar el editor del código fuente o no aparece en nuestra interfaz de usuario, la forma más sencilla de encontrarlo es por medio del Explorador de soluciones. Simplemente debemos dirigirnos al documento que representa nuestro código fuente y hacer doble clic en él, la ventana con el editor de código fuente aparecerá.

www.redusers.com 19

1. C#.NET

*Figura 2. Éste es el cuadro de dialogo que usamos para crear un nuevo proyecto, gracias a sus opciones podemos generar diferentes tipos de proyectos.*

En unos pocos segundos C# Express crea el proyecto para nosotros. Visual Studio y la versión Express hacen uso de **soluciones** y proyectos, una solución puede tener varios proyectos, por ejemplo, Office tiene diferentes productos como Word, Excel y PowerPoint. Office es una solución y cada producto es un proyecto. El proyecto puede ser un programa independiente o una librería, y puede tener uno o varios do cumentos, estos documentos pueden ser el código fuente y recursos adicionales.

Podemos observar que nuestra interfaz de usuario ha cambiado un poco, las venta nas ya nos muestran información y también podemos ver que en la zona de edición se muestra el esqueleto para nuestro programa.



*Figura 3. Esta figura nos muestra el panel derecho denominado* ***Solution Explorer****.*

20 www.redusers.com 

Breve historia de la programación para Windows

El **Explorador de soluciones** nos muestra la información de la solución de forma lógica, si observamos es como un pequeño árbol. En la raíz encontramos a la so lución, cada proyecto que tengamos en esa solución será una rama, cada proyecto, a su vez, tendrá también sus divisiones. En nuestro caso vemos tres elementos, dos de esos elementos son carpetas, en una se guardan las propiedades del proyecto y en la otra las referencias (durante este libro no haremos uso de estas carpetas). El tercer elemento es un documento llamado **Program.cs**, éste representa al documento donde guardamos el código fuente de nuestra aplicación. Vemos que la extensión de los programas de C# es .CS.

En el área de edición podemos observar que tenemos un esqueleto para que, a partir de ahí, podamos crear nuestro propio programa. Para entender lo que te nemos ahí es necesario conocer un concepto: **namespace**. El namespace es una agrupación lógica, por ejemplo, todo el código que podemos tener relacionado con matemáticas puede quedar agrupado dentro del namespace de **Math**. Otro uso que tiene el namespace es el de resolver conflictos con los nombres, por ejem plo, supongamos que tenemos un proyecto muy grande y varios programadores trabajando en él. Es posible que ambos programadores crearan un método que tuviera el mismo nombre, esto nos genera un conflicto ya que el programa no podría saber cual versión utilizar. La forma de resolver esto es que cada progra mador tenga su propio namespace y hacer referencia al namespace correspon diente según la versión que deseáramos utilizar.

El Framework de .NET nos provee de varios namespaces donde tenemos miles de clases y métodos ya creados para nuestro uso. Cuando deseamos utilizar los recur sos que se encuentran en un namespace programado por nosotros o por otros pro gramadores, debemos hacer uso de un comando de C# conocido como **using**.

Como podemos ver en la parte superior del código, tenemos varios **using** haciendo referencia a los namespace que necesita nuestra aplicación; si necesitáramos adicio nar más namespaces, lo haríamos en esta sección.

Más abajo se está definiendo el namespace propio de nuestro proyecto, esto se ha ce de la siguiente manera:

**namespace MiProyecto**

**{**

**}**

El namespace que estamos creando se llama **MiProyecto**, como podemos ver el na mespace usa **{ }** como delimitadores, esto se conoce como un **bloque de código**, todo lo que se coloque entre **{ }** pertenecerá al namespace; ahí es donde será nece sario escribir el código correspondiente a nuestra aplicación.

www.redusers.com 21

1. C#.NET

Dentro del bloque de código encontramos la declaración de una **clase**, C# es un lenguaje orientado a objetos y por eso necesita que declaremos una clase para nues tra aplicación. La clase tiene su propio bloque de código y en nuestra aplicación se llamará **Program**. El concepto de clase se verá en el Capítulo 10 de este libro.

Todos los programas necesitan de un punto de inicio, un lugar que indique dónde empieza la ejecución del programa, en C#, al igual que en otros lenguajes, el pun to de inicio es la función **Main()**; esta función también tiene su propio bloque de código. Dentro de esta función generalmente colocaremos el código principal de nuestra aplicación, aunque es posible tener más funciones o métodos y clases. Las partes y características de las funciones se ven en el Capítulo 5 de este libro.

Ahora es el momento de crear nuestra primera aplicación. Vamos a modificar el có digo de la función tal y como se muestra a continuación. Cuando estemos adicio nando la sentencia dentro de **Main()**, debemos fijarnos que sucede inmediatamen te después de colocar el punto.

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**namespace MiProyecto**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**Console.WriteLine(“Hola Mundo!”);**

**}**

**}**

**}**

Si logramos observar, cuando se escribió la palabra **Console** y luego el punto, apare ció un recuadro listando diferentes elementos. Este cuadro se llama **autocompletar** y nos permite trabajar rápidamente y reducir la cantidad de errores de sintaxis. El cuadro de autocompletar nos muestra sugerencia de la palabra, comando, variable, etc. que se podría usar a continuación. Si lo deseamos, lo seleccionamos de esa lis ta, y él lo escribe por nosotros, conforme avancemos y nos familiaricemos con la programación de C# veremos que es de gran ayuda.

En nuestro ejemplo hemos colocado una sentencia que escribirá el mensaje **Hola Mundo** en la consola. Es un programa muy sencillo, pero nos permitirá aprender có mo llevar a cabo la compilación de la aplicación y ver el programa ejecutándose.

22 www.redusers.com 

Para compilar la aplicación

Breve historia de la programación para Windows

Una vez que terminemos de escribir nuestro programa, podemos llevar a cabo la compilación, como aprendimos anteriormente esto va a generar el assembly que lue go usará el runtime cuando se ejecute.

Para poder compilar la aplicación debemos seleccionar el menú de **Depuración** o **De bug** y luego **Construir Solución** o **Build Solution**. El compilador empezará a trabajar y en la barra de estado veremos que nuestra solución se ha compilado exitosamente.



*Figura 4. Para compilar nuestra aplicación, primero debemos*

*construirla, seleccionando la opción adecuada desde el menú.*

Para ejecutar la aplicación

Una vez que la compilación ha sido exitosa, podemos ejecutar nuestro programa y ver cómo trabaja. Para esto tenemos dos opciones: **ejecutar con depuración** y **eje cutar sin depurador**. En la versión Express únicamente aparece la ejecución con depuración, pero podemos usar la ejecución sin depurador con las teclas **CTRL+F5** o adicionándola usando el menú de herramientas.

El **depurador** es un programa que nos ayuda a corregir errores en tiempo de eje cución y también errores de lógica. En el Capítulo 12 de este libro aprenderemos có mo utilizarlo. De preferencia debemos utilizar la ejecución sin depurador y hacer

INTERFAZ GRÁFICA

Éste es un libro de inicio a la programación con C#, por eso todos los programas se ejecutarán en la consola. Una vez comprendidos los conceptos principales que se enseñan, es posible aprender cómo se programan las formas e interfaz gráfica en C#. Este tipo de programación no es difícil, pero si requiere tener los conocimientos básicos de programación orientada a objetos.

www.redusers.com 23

1. C#.NET

uso de la ejecución con depuración únicamente cuando realmente la necesitemos. Ahora ejecutaremos nuestra aplicación, para esto oprimimos las teclas **CTRL+F5**.



*Figura 5. Con el menú* ***Debug/Start Without***

***Debbuging*** *nuestro programa se ejecutará.*

Cuando el programa se ejecuta, aparece una ventana, a ella la llamamos consola y se encarga de mostrar la ejecución del programa. De esta forma podremos leer el mensaje que habíamos colocado en nuestro ejemplo.



*Figura 6. Ahora podemos ver nuestra aplicación ejecutándose en la consola.*

Cómo detectar errores en un programa

En algunas ocasiones puede ocurrir que escribimos erróneamente el programa, cuan do esto sucede, la aplicación no podrá compilarse ni ejecutarse; si esto llegara a ocu rrir debemos cambiar lo que se encuentra mal.

24 www.redusers.com 

Breve historia de la programación para Windows

Escribamos a propósito un error para que veamos cómo se comporta el compilador.

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**namespace MiProyecto**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**Consola.WriteLine(“Hola Mundo!”);**

**}**

**}**

**}**

En el programa hemos cambiado **Console** por **Consola**, esto provocará un error; aho ra podemos tratar de compilar nuevamente y ver lo que sucede.

*Figura 7. En esta imagen vemos que el programa*

*tiene un error y por lo tanto no se puede compilar.*

**www.redusers.com 25

1. C#.NET

Como ya sabemos, la ventana que aparece en la parte inferior de nuestra interfaz de usuario es utilizada por el compilador para comunicarse, vemos que aparece una ventana que nos entrega un listado de errores; en el Capítulo 12, dedicado a la depu ración, aprenderemos a utilizarla, ahora simplemente debemos saber que siempre es necesario resolver el primer problema de la lista y que podemos ir directamente al error haciendo doble clic con el mouse sobre él.

La vista de clases

Ahora revisaremos la **vista de clases**. En esta vista podemos obtener información sobre la solución que estamos creando, pero a diferencia del Explorador de solu ciones, la información se encuentra ordenada en forma lógica.

Con esta vista podemos encontrar rápidamente los namespace de nuestra solución y dentro de ellos las clases que contienen; si lo deseamos, podemos ver los métodos que se encuentran en cada clase. Esta vista no solamente permite observar la información lógica, también nos da la posibilidad de navegar rápidamente en nuestro código.

En la versión Express no viene la opción previamente configurada, por lo que es ne cesario adicionar el comando al menú **Vista**.

Para mostrar la vista de clases debemos de ir al menú **Vista** y luego seleccionar **Vis ta de Clases**, una ventana aparecerá en nuestra interfaz de usuario.



*Figura 8. Para mostrar la vista de clases debemos de usar el menú* ***View****.*

La ventana de la vista de clases está divida en dos secciones, la sección superior nos muestra la relación lógica y jerárquica de los elementos mientras que en la parte in ferior vemos los métodos que componen a alguna clase en particular.

.NET EN LINUX

Si deseamos trabajar bajo **Linux**, es posible utilizar **Mono**. Solamente debemos recordar que no siempre la versión .NET que maneja Mono es la más reciente de Microsoft. El sitio de este proyecto lo encontramos en la dirección web **www.mono-project.com/Main\_Page**, desde ella podemos descargarlo en forma completamente gratuita.

26 www.redusers.com 

Breve historia de la programación para Windows



*Figura 9. Aquí podemos observar a la vista de clases en el lado*

*derecho. El árbol está abierto para poder ver los elementos que lo componen.*

En la raíz del árbol encontramos al proyecto, éste va a contener las referencias ne cesarias y el namesapace de **MiProyecto**. Si tuviéramos más namespaces estos apare cerían ahí, al abrir el namespace de **Miproyecto** encontramos las clases que están de claradas dentro de él. En este caso solamente tenemos la clase **Program**, si la selec cionamos veremos que en la parte inferior se muestran los elementos que estén de clarados en su interior. En nuestro ejemplo tenemos el método **Main()**.

Si en este momento damos doble clic en cualquiera de los métodos, nos dirigiremos automáticamente al código donde se define, como nuestro programa es muy pe queño, posiblemente no vemos la ventaja de esto, pero en programas con miles de líneas de código, el poder navegar rápidamente es una gran ventaja.

Configurar los menús del compilador

Para poder adicionar las opciones que nos hacen falta en los menús debemos seguir una serie de pasos muy sencillos. Nos dirigimos primero al menú **Tools** o **Herra mientas** y seleccionamos la opción **Customize**.

MSDN

Uno de los sitios más importantes que tenemos que visitar cuando desarrollamos para .NET es el llamado **MSDN**. En este sitio encontraremos toda la documentación y variados ejemplos para todas las clases y métodos que componen a .NET. El sitio se encuentra en la dirección web **www.msdn.com**.

www.redusers.com 27

1. C#.NET

*Figura 10. Es necesario seleccionar la opción de* ***Customize…*** *en el menú* ***Tools****.*

Nos aparece un cuadro de diálogo, en el cual debemos seleccionar **Commands**, en esta sección podemos elegir el menú al cual queremos adicionar un comando, por ejemplo el menú **View**.



*Figura 11. En esta ventana debemos seleccionar el menú que deseamos modificar.*

28 www.redusers.com 

Breve historia de la programación para Windows

Aparecen listados los comandos actuales del menú, damos clic en la zona donde de seamos que se inserte el nuevo comando y oprimimos el botón **Add Command**. Con esto aparece un nuevo cuadro de dialogo que muestra todos los posibles comandos que podemos adicionar clasificados en categorías.

En las categorías seleccionamos **View** y en los comandos **Class View**, este comando es el que nos permite tener la vista de clases. El comando de **Iniciar Sin Depurar** se encuentra en la categoría **Debug**.

*Figura 12. En esta ventana seleccionamos el comando que deseamos insertar.*

Hasta aquí hemos analizado los conceptos básicos más importantes necesarios para comprender el funcionamiento de .NET, a partir del próximo capítulo comenzare mos con nuestro aprendizaje de programación y del lenguaje C#.

**…** RESUMEN

**Hace algunos años la programación para Windows resultaba complicada y requería de un alto grado de especialización. El Framework de .NET soluciona muchos de los problemas relacionados con la programación en Windows y la interoperabilidad, las aplicaciones de .NET se compilan a un assembly que contiene el programa escrito en CIL; éste es un lenguaje intermedio que lee el runtime cuando se ejecuta la aplicación. El CLR compila el CIL para el microprocesador según va siendo necesario, el uso de un runtime le da a .NET la flexibilidad de ser multiplataforma. Todos los lenguajes .NET deben cumplir con los lineamientos que encontramos en el CLS.**

****www.redusers.com 29

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1 ¿Cuál es la historia del desarrollo de las aplicaciones para Windows?

2 ¿Qué problemas ayuda a resolver .NET? 3 ¿Qué es un assembly?

4 ¿Cuál es la definición de CIL y CLR?

5 Describa qué hace el CLR con el assembly.

6 ¿Cuál es el significado de Jitter?

7 Mencione algunos compiladores de C# que podemos utilizar.

8 ¿Qué es el CTS?

9 ¿Porqué .NET puede ser multiplataforma? 10 ¿Qué otro lenguaje que use .NET existe? 11 ¿Cuál es la última versión de .NET? 12 ¿Qué otro lenguaje que use .NET existe?

EJERCICIOS PRÁCTICOS

1 Cree un proyecto nuevo que imprima su nombre en la ventana de la consola.

2 Coloque un error a propósito y vea cómo se comporta el compilador.

3 Explore el proyecto usando la vista de clases.

4 Experimente con el explorador de soluciones y vea la información que le provee.

5 Agregue más mensajes a la aplicación que ha creado.

30 www.redusers.com 

C#

Los elementos básicos de un programa

Llegó el momento de aprender cómo desarrollar programas de computadoras. Algunas personas piensan que programar software es complejo y difícil,

sin embargo, esto no es cierto,

o lo fue hace muchísimo tiempo atrás. Para ser programadores, sólo

necesitamos conocer un lenguaje de programación y las técnicas necesarias para desarrollar programas. A lo largo de estos capítulos aprenderemos ambas.

**SERVICIO DE ATENCIÓN AL LECTOR: usershop@redusers.com**

Capítulo 2

**Los lenguajes de programación 32** Los programas de computadora 32 Las variables 42 Operaciones aritméticas 49 Cómo pedirle datos al usuario 56 Cómo resolver problemas

en la computadora 60 Resolución de problemas

en la computadora 64 **Resumen 69 Actividades 70**

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

LOS LENGUAJES DE PROGRAMACIÓN

Cuando queremos comunicarnos con una persona, hacemos uso de un idioma en común para entendernos. Nosotros, con este libro, nos comunicamos por medio del lenguaje español, pero si llegara a visitarnos una persona de otro país donde no se hable español, entonces tendríamos un problema. La solución a este problema es muy sencilla: necesitamos aprender el idioma de esta persona para poder co municarnos con ella. Lo mismo sucede con la computadora si queremos progra marla, es necesario comunicarnos con ella, y la forma correcta de hacerlo es ha blando su idioma. Entonces podemos decir que el idioma de la computadora se conoce como **lenguaje de programación** y el lenguaje de programación que apren deremos es **C#**. Este lenguaje es sencillo y poderoso a su vez.

Una vez que sepamos el lenguaje de programación, entonces la computadora en tenderá qué es lo que queremos que haga.

Los lenguajes de programación existen desde hace muchos años y son de diferentes tipos. Algunos han evolucionado y han surgido nuevos lenguajes como C#. Otros lenguajes son especializados para determinadas labores y muchos otros se han ex tinguido o han quedado obsoletos.

Los programas de computadora

Todos o casi todos sabemos cómo cocinar una torta, y de hecho, éste es un buen ejemplo para saber cómo hacer un programa de computadora. Cuando queremos cocinar una torta, lo primero que necesitamos son los ingredientes. Si falta alguno, entonces no tendremos una rica torta. Ya una vez que hemos preparado o recolec tado nuestros ingredientes, seguimos la receta al pie de la letra. Ésta nos dice paso a paso qué es lo que tenemos que hacer.

Pero, ¿qué sucedería si no seguimos los pasos de la receta?, ¿qué sucede si ponemos los ingredientes primero en el horno y luego los revolvemos?, ¿el resultado es una torta? La respuesta a estas preguntas es evidente: el resultado no es una torta. De esto dedu cimos que para lograr algo, necesitamos una serie de pasos, pero también es necesario seguirlos en el orden adecuado. A esta serie de pasos la llamamos receta.

CÓMO EVITAR ERRORES

Un error que se puede tener al iniciarnos con la programación de computadoras con C# es olvidar colocar **;** al finalizar la sentencia. Si nuestro programa tiene algún error y todo parece estar bien, seguramente olvidamos algún punto y coma. Es bueno reconocer estos tipos de errores ya que nos permite reducir el tiempo de corrección necesario.

32 www.redusers.com 

Los lenguajes de programación

Cualquier persona puede pensar en una actividad y mencionar la lista de pasos y el orden en el que se deben cumplir. Podemos hacer sin problema la lista de pasos para cambiar el neumático del automóvil o para alimentar a los peces del acuario. Con esto vemos que no encontraremos problema alguno para listar los pasos de cualquier actividad que hagamos.

Ahora, seguramente nos preguntamos: ¿qué tiene esto que ver con la programación de computadoras? y la respuesta es simple. Cualquier persona que pueda hacer una lista de pasos puede programar una computadora. El programa de la computadora no es otra cosa que una lista de los pasos que la computadora tiene que seguir para hacer alguna actividad. A esta lista de pasos la llamamos **algoritmo**. Un algoritmo son los pasos necesarios para llevar a cabo una acción. Una característica importan te del algoritmo es que tiene un punto de **inicio** y un punto de **fin**, lo que indica que los pasos se llevan a cabo de forma **secuencial**, uno tras otro. El algoritmo se ría equivalente a la receta de la torta y los ingredientes necesarios generalmente se rán los datos o la información que usaremos.

Hagamos un pequeño programa para ver cómo funcionaría esto. Para esto creamos un proyecto, tal y como vimos en el **Capítulo 1**. Nuestro programa mostrará los pa sos para hacer una torta o un pastel. Colocamos el siguiente código en nuestro pro grama y veremos cómo se corresponde con lo que hemos aprendido.

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace AplicacionBase**

**{**

**class Program**

**{**

**// Esta es la funcion principal del programa**

**// Aqui inicia la aplicacion**

**static void Main(string[] args)**

LAS FUNCIONES EN C#

Las funciones son secciones del programa en las que podemos tener código especializado. La función **Main()** es invocada por el sistema operativo cuando queremos que nuestro programa se inicie. Es la única función que deben tener todos los programas que desarrollemos.

www.redusers.com 33

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

**{**

**Console.WriteLine(“1- Precalentar el horno”);**

**Console.WriteLine(“2- Batir margarina con azucar”);**

**Console.WriteLine(“3- Agregar huevos”);**

**Console.WriteLine(“4- Cernir harina”);**

**Console.WriteLine(“5- Agregar a la mezcla y leche”);**

**Console.WriteLine(“6- Hornear por 40 minutos”);**

**Console.WriteLine(“7- Decorar y comer”);**

**}**

**}**

**}**

C# es un **lenguaje orientado a objetos** y necesita que definamos una **clase** para po der crear un programa. En este momento no veremos las clases, pero regresaremos a ellas en un capítulo posterior. Lo primero que debemos encontrar es una **función** que se llama **Main()**, que es muy importante, como vimos en el **Capítulo 1**.

Ya sabemos que los algoritmos necesitan un inicio y un fin. En nuestro programa el punto donde siempre inicia es la función **Main()**. Esta función indica el punto de arranque de nuestro programa y aquí colocaremos los pasos que debe seguir nuestro programa en el orden adecuado.

Para indicar todos los pasos que pertenecen a **Main()** necesitamos colocarlos aden tro de su **bloque de código**. Un bloque de código se define entre llaves **{ }**. To do lo que se encuentre en ese bloque de código pertenece **a Main()**, lo que se en cuentre afuera de él, no pertenece.

Es importante que notemos que la **clase** también tiene un bloque de código y **Main()** se encuentra adentro de él. De igual forma, el **namespace** también tiene su propio bloque de código y la clase se encuentra adentro, aunque por ahora sólo nos con centraremos en el bloque de código de **Main()**.

Un error común cuando se crean bloques de código es olvidar cerrarlos. Como re comendación, una vez que abrimos un bloque de código con **{**, inmediatamente de bemos de colocar el cierre con **}** y luego programar su contenido. De esta forma, no

ASIGNACIÓN DE VALORES FLOTANTES

Cuando colocamos un valor con decimales en C# automáticamente se lo interpreta como de tipo **double**. Sin embargo, si necesitamos colocar ese valor en una variable de tipo **float**, debemos usar el sufijo **f**. Éste le dice a C# que use ese valor como flotante. Ejemplo: **resultado = a \* 3.57f;**.

34 www.redusers.com 

Los lenguajes de programación

olvidaremos cerrarlo. Dentro de **Main()** encontramos los pasos de nuestro algorit mo. En este momento nos pueden parecer extraños, pero aprenderemos qué signi fican. Tenemos que recordar que debemos aprender el idioma de la computadora y éstas son una de sus primeras frases.

Lo que encontramos adentro de la función **Main()** es una serie de **sentencias**. Las sentencias nos permiten colocar instrucciones que nuestro programa pueda eje cutar, que son utilizadas para mostrar un mensaje en la pantalla. El mensaje que nosotros queremos mostrar en la pantalla se muestra al invocar el método

**WriteLine()**. Los métodos son similares a las funciones, existen adentro de las cla ses y nos permiten hacer uso de las funcionalidades internas. Este método perte nece a la clase **Console**, y en esa clase encontraremos todo lo necesario para que podamos trabajar con la consola, ya sea colocando mensajes o leyendo la infor mación proporcionada por el usuario. Como **WriteLine()** pertenece a **Console** cuando queremos utilizarlo, debemos escribir:

**Console.WriteLine(“Hola”);**

**WriteLine()** necesita cierta información para poder trabajar. Esta información es el mensaje que deseamos mostrar. A los métodos se les pasa la información que nece sitan para trabajar por medio de sus **parámetros**. Los parámetros necesarios se co locan entre **paréntesis()**. **WriteLine()** necesitará un parámetro de tipo **cadena**.

Una **cadena** es una colección de caracteres, es decir, letras, números y signos. La ca dena se delimita con **comillas dobles “**. En el ejemplo, el mensaje que queremos que se muestre es: **Hola**. Por eso pasamos el parámetro de tipo cadena **“Hola”**. Las sentencias se finalizan con **punto y coma ;**. Esto es importante y no debemos olvidarlo ya que este carácter en particular es reconocido por C# para indicar que la sentencia escrita hasta allí, ha finalizado.

En cada una de las sentencias invocadas hasta aquí por nuestro programa, hemos enviado un mensaje tras otro con cada paso necesario para cocinar nuestro pas tel o nuestra torta. Por último, sólo resta que ejecutemos el programa escrito y veamos el resultado que aparece en la pantalla.

PROBLEMAS CON LOS MÉTODOS

Cuando se invoca un método es necesario no solamente colocar su nombre, sino que también se deben poner los paréntesis. Éstos se deben poner aun cuando el método no necesite parámetros. No hay que olvidar usar el paréntesis de cierre **)**. Si olvidamos colocar los paréntesis, el compilador nos marcará errores y no podremos ejecutar el programa.

www.redusers.com 35

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

*Figura 1. Podemos observar los mensajes*

*que enviamos a la pantalla por medio del método* ***WriteLine()****.*

Podemos observar que la ejecución ha mostrado los mensajes en el orden en el que los hemos puesto. Ahora podemos hacer un pequeño experimento. Cambia remos el orden de las sentencias y observaremos qué es lo que sucede. El progra ma quedará de la siguiente forma:

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace AplicacionBase**

**{**

**class Program**

**{**

**// Esta es la funcion principal del programa**

**// Aqui inicia la aplicacion**

**static void Main(string[] args)**

**{**

**Console.WriteLine(“1- Precalentar el horno”);**

36 www.redusers.com 

Los lenguajes de programación

**Console.WriteLine(“2- Batir margarina con azucar”);**

**Console.WriteLine(“6- Hornear por 40 minutos”);**

**Console.WriteLine(“5- Agregar a la mezcla y leche”);**

**Console.WriteLine(“7- Decorar y comer”);**

**Console.WriteLine(“3- Agregar huevos”);**

**Console.WriteLine(“4- Cernir harina”);**

**}**

**}**

**}**

Ejecutémoslo y veamos qué ocurre.

*Figura 2. Aquí vemos que los mensajes se colocan en el orden*

*en que se encuentran en la función* ***main()*** *y no en el orden en que esperábamos.*

Podemos observar que cuando se ejecuta, las sentencias se ejecutan en el mismo orden como están en el código del programa, no se ejecutan como nosotros sa bemos que se deben de ejecutar. Este punto es importante, ya que nos muestra que colocar las sentencias del programa en el orden de ejecución correcto es res ponsabilidad del programador. La computadora no puede leernos la mente y sa ber qué es lo que realmente deseamos, ésta sólo ejecuta fielmente las sentencias

www.redusers.com 37

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

en el orden en que las colocamos. Si en la vida real primero horneamos y luego batimos, no podemos esperar obtener un pastel o una torta. Por lo tanto, si en la computadora no ponemos las sentencias del programa en el orden adecuado, no podemos esperar tener un programa funcional.

Errores en los programas

Una parte importante de la programación es reconocer los errores que podemos tener con el fin de corregirlos y permitir que el programa trabaje correctamente. Podemos tener dos tipos de errores: los **errores de sintaxis** y los **errores de lógi ca**. Los errores de sintaxis son fáciles de corregir y se deben a que escribimos en el orden incorrecto las sentencias del programa. Estos errores se muestran cuando el programa se compila, y de esa forma se pueden buscar fácilmente para corregirlos. Los errores de lógica se deben a que hemos creado nuestro algoritmo en forma erró nea y aunque todas las sentencias estén correctas, el programa no lleva a cabo lo que deseamos. Estos errores son más difíciles de corregir y la mejor forma de evi tarlos es por medio de un análisis previo adecuado del problema a resolver. Más adelante aprenderemos cómo llevar a cabo este análisis.

Empecemos a reconocer los errores de sintaxis y los mensajes que da el compila dor cuando los tenemos. El reconocer estos mensajes y cómo fueron ocasionados nos permitirá resolverlos rápidamente en el futuro. Para esto empezaremos a ha cer unos experimentos. En el código de nuestro programa, en cualquiera de las sentencias, eliminamos el punto y coma que se encuentra al final, y luego lo com pilamos para ver qué sucede. Debido a que hemos eliminado el punto y coma, te nemos un error de sintaxis y el compilador nos dice que el programa no se ha compilado exitosamente.

Dentro de nuestro compilador se mostrará la ventana de lista de errores. En esta ventana se colocan los errores encontrados por el compilador. Siempre es conve niente solucionar el problema que se encuentra en la parte superior de la lista y no el último, debido a que muchas veces, solucionando el primero, quedan soluciona dos otros que arrastraban su problema. El mensaje que vemos nos indica que se es peraba punto y coma, tal y como se ve en la **figura 3**.

ERROR COMÚN DE LAS CADENAS

Algunas veces sucede que olvidamos colocar la cadena entre comillas o que solamente se ha colocado la comilla del inicio. Es importante no olvidar colocar las cadenas siempre entre las dos comillas. Si no lo hacemos de esta forma, el compilador no puede encontrar correctamente la cadena o dónde finaliza. La sentencia nos marcará un error de compilación.

38 www.redusers.com 

Los lenguajes de programación

*Figura 3. Podemos ver la ventana con la lista*

*de errores y el error de sintaxis encontrado por el compilador.*

La lista de errores nos da una descripción del error y también nos permite ver el nú mero de línea donde se encuentra el error. Una forma sencilla de llegar a éste es sim plemente hacer doble clic sobre la descripción del error, así el entorno C# nos lle vará directamente a dónde se encuentra éste. Coloquemos nuevamente el punto y coma en el lugar donde estaba en un principio y compilemos. En esta ocasión, el programa puede compilar y ejecutar sin problemas, en base a la correción que he mos realizado con antelación. El error se encuentra solucionado. La solución que nos brinda C# para corregir errores es práctica y sencilla de implementar.

Ahora experimentemos otro error. A la primera sentencia le borramos el paréntesis de cierre y luego compilamos para ver qué ocurre:

ERRORES CON LOS NOMBRES DE VARIABLES

Un error común que los principiantes hacen cuando nombran sus variables es que olvidan cómo nombraron a la variable y la escriben a veces con minúscula y a veces con mayúscula. Esto hace que se produzcan errores de lógica o de sintaxis. Para evitarlo, podemos utilizar una tabla que contenga la información de las variables de nuestro programa.

www.redusers.com 39

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

*Figura 4. Aquí vemos el mensaje de error generado por no cerrar el paréntesis.*

*Figura 5. Ahora nuestra lista de errores muestra tres errores en el programa.*

40 www.redusers.com 

Los lenguajes de programación

Corrijamos el error y compilemos de nuevo. Todo debe de estar bien. Pero lo que haremos ahora es borrar las comillas de cierre de la cadena en la primera sentencia. Este error nos presenta un escenario más interesante y nos conviene entender qué es lo que sucede. Compilemos el programa y veamos nuestra lista de errores. Ésta será mucho más extensa que las anteriores:

Aquí ha sucedido algo interesante, a pesar de que solamente hemos creado un error: en la lista de errores aparecen tres errores. Esto se debe a que un error pue de ocasionar errores extras en el código siguiente. Al no cerrar la cadena, el com pilador no sabe dónde termina la sentencia y cree que el paréntesis de cierre y el punto y coma forman parte de la cadena. Por eso también nos muestra que fal tan el paréntesis y el punto y coma.

Debido a que los errores pueden ser en cadena, como en este ejemplo, siempre ne cesitamos corregir el error que se encuentra en primer lugar. Si nosotros intentára mos corregir primero el error del punto y coma, no resultaría evidente ver cuál es el error en esa línea, ya que veríamos que el punto y coma sí está escrito.

A veces sucede que nuestro programa muestra muchos errores, pero cuando corre gimos el primer error, la gran mayoría desaparece. Esto se debe a factores similares a los que acabamos de ver.

La diferencia entre los métodos Write() y WriteLine()

Hemos visto que el método **WriteLine()** nos permite mostrar un mensaje en la con sola, pero existe un método similar que se llama **Write()**. La diferencia es muy sen cilla. Después de escribir el mensaje **WriteLine()** inserta un salto de línea, por lo que lo próximo que se escriba aparecerá en el renglón siguiente. Por su parte, el méto do **Write()** no lleva a cabo ningún salto de línea y lo siguiente que se escriba será en la misma línea. Veamos todo esto en el siguiente ejemplo:

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace AplicacionBase**

**{**

**class Program**

**{**

**// Esta es la funcion principal del programa**

**// Aqui inicia la aplicacion**

**static void Main(string[] args)**

**{**

**Console.Write(“Mensaje 1 “);**

****www.redusers.com 41

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

**Console.Write(“Mensaje 2 “);**

**Console.WriteLine(“Mensaje 3 “);**

**Console.WriteLine(“Mensaje 4 “);**

**Console.Write(“Mensaje 5 “);**

**}**

**}**

**}**

Al ejecutarlo, vemos cómo en efecto después de usar **Write()** lo siguiente que se es cribe se coloca en el mismo renglón, pero al usar **WriteLine()** lo siguiente aparece en el renglón siguiente. Esto es un ejemplo claro de lo explicado anteriormente:

*Figura 6. Podemos observar la diferencia entre los métodos* ***Write()*** *y* ***WriteLine()****.*

Las variables

Las computadoras no solamente escriben mensajes en la pantalla, sino que deben hacer cosas útiles y para poder hacerlas necesitan información. La información de be almacenarse y manipularse de alguna forma.

Cuando queremos guardar algo en la vida cotidiana podemos utilizar cajas, por lo que cualquier cosa que deseamos almacenar puede ser colocada en una caja de

42 www.redusers.com 

Los lenguajes de programación

cartón, por ejemplo. Sin embargo, el tamaño de la caja dependerá de lo que de seamos almacenar. Unos zapatos pueden caber en una caja pequeña, pero en esa caja no podemos colocar un televisor. Si empezamos a almacenar muchas co sas en cajas, pronto tendremos demasiadas y será difícil poder encontrar lo que hemos guardado. Entonces, una forma de solucionar esto es por medio de eti quetas. A cada caja le ponemos una etiqueta con el contenido y de esta forma po demos encontrar rápidamente lo que buscamos.

Las **variables** en la computadora funcionan de manera similar a nuestras cajas de almacenaje. Las podemos imaginar como pequeñas cajas que existen en la memo ria de la computadora y su tamaño dependerá de la información que deben guar dar. Esto se conoce como tipo de la variable. Para poder acceder a esa caja o varia ble le ponemos un nombre que sería la etiqueta con la que la identificamos.

Para hacer uso de una variable, lo primero que tenemos que hacer es **declararla**. La declaración de éstas es algo muy sencillo. Como primer paso tenemos que co locar el **tipo** y luego el **nombre**. Las variables en C# deben nombrarse de acuerdo con unas recomendaciones sencillas:

• Los nombres de las variables deben empezar con letra.

• Es posible colocar números en los nombres de las variables, pero no empezar el nombre con ellos.

• Los nombres de las variables no pueden llevar signos a excepción del guión bajo **\_** . • Las variables no pueden llevar acentos en sus nombres.

Cuando nombramos las variables, hay que tener en cuenta que C# es **sensible** a las **mayúsculas** y **minúsculas**, por lo que una variable llamada **costo** no es la misma que otra variable llamada **COSTO** u otra llamada **Costo**.

Es recomendable nombrar a las variables con nombres que hagan referencia a la in formación que guardarán. Si nos acostumbramos a hacer esto desde que empeza mos a programar, evitaremos muchos problemas en el futuro y será mucho más sen cillo corregir nuestros programas cuando tengan errores.

Veamos un ejemplo de cómo podemos declarar una variable que guarde valores **nu méricos enteros**. El tipo que guarda estos valores se conoce como **int**.

SELECCIÓN DEL TIPO

Es importante conocer los tipos y la información que pueden guardar, ya que esto nos permitirá guardar la información necesaria y utilizar la menor cantidad de memoria. Podemos aprender los rangos de los tipos o imprimir una tabla y tenerla a mano. Con la práctica podremos utilizar los tipos sin tener que verificar los rangos constantemente.

www.redusers.com 43

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

**int costo;**

Vemos que al final de la sentencia hemos colocado el punto y coma. Si necesitamos declarar más variables lo podemos hacer de la siguiente forma:

**int costo;**

**int valor;**

**int impuesto;**

Pero también es posible declarar las variables en una sola línea. Para esto simple mente separamos los nombres de las variables con comas. No hay que olvidar co locar el punto y coma al final de la sentencia.

**int costo, valor, impuesto;**

C# nos provee de muchos tipos para poder usar con nuestras variables, que pode mos conocer en la siguiente tabla:

TIPO INFORMACIÓN QUE GUARDA

**bool** Es una variable booleana, es decir que solamente puede guardar los valores verdadero o falso (**true** o **false**) en términos de C#.

**byte** Puede guardar un byte de información. Esto equivale a un valor entero positivo entre 0 y 255. **sbyte** Guarda un byte con signo de información. Podemos guardar un valor entero con signo desde –128 hasta 127.

**char** Puede guardar un carácter de tipo Unicode.

**decimal** Este tipo puede guardar un valor numérico con decimales. Su rango es desde ±1.0 ? 10?28 hasta ±7.9 ? 1028.

**double** También nos permite guardar valores numéricos que tengan decimales. El rango aproximado es desde ±5.0 ? 10?324 hasta ±1.7 ? 10 308.

FORMA CORRECTA DE HACER UNA ASIGNACIÓN

Un punto importante a tener en cuenta es que la asignación siempre se lleva a cabo de derecha a izquierda. Esto quiere decir que el valor que se encuentra a la derecha del signo igual se le asigna a la variable que se encuentra a la izquierda del signo igual. Si no lo hacemos de esta forma, el valor no podrá ser asignado o será asignado a la variable incorrecta.

44 www.redusers.com 

TIPO INFORMACIÓN QUE GUARDA

Los lenguajes de programación

**float** Otro tipo muy utilizado para guardar valores numéricos con decimales. Para este tipo el rango es desde ±1.5 ? 10?45 hasta ±3.4 ? 1038.

**int** Cuando queremos guardar valores numéricos enteros con signo, en el rango de -2,147,483,648 hasta 2,147,483,647.

**uint** Para valores numéricos enteros positivos, su rango de valores es desde 0 hasta 4,294,967,295. **long** Guarda valores numéricos enteros realmente grandes con un rango

desde –9,223,372,036,854,775,808 hasta 9,223,372,036,854,775,807.

**uling** Guarda valores numéricos enteros positivos. Su rango de valores varía desde 0 hasta 18,446,744,073,709,551,615.

**short** Guarda valores numéricos enteros, pero su rango es menor y varía desde -32,768 hasta 32,767. **ushort** Puede guardar valores numéricos enteros positivos con un rango desde 0 hasta 65,535. **string** Este tipo nos permite guardar cadenas.

*Tabla 1. Ésta es la tabla de los tipos más utilizados en C#.*

*Es útil para seleccionar el tipo adecuado para las variables.*

Una vez que hemos declarado la variable, debemos **inicializarla**. Inicializar una va riable es asignarle un valor por primera vez. Esto nos permite darle un valor inicial que puede ser útil en la ejecución de nuestro programa (no sólo podemos asignarle a una variable un valor fijo, sino también puede ser desde un texto ingresado por el usuario o desde el registro de una base de datos).

Para asignarle un valor a una variable, ya sea durante la inicialización o durante el transcurso del programa, usamos un **operador de asignación**, el signo **igual**. Veamos a continuación un ejemplo de cómo asignar valores:

**costo = 50;**

**valor = 75;**

**impuesto = 10;**

En este caso, la variable **costo** almacena el valor de 50, la variable **valor** almacena el 75, y la variable **impuesto** guarda el valor 10 .

LOS COMENTARIOS Y EL COMPILADOR

Los comentarios solamente nos sirven a nosotros como seres humanos para facilitarnos a comprender el código del programa. El compilador ignora todos los comentarios y no los toma en cuenta en el momento de la compilación. Los comentarios no hacen el programa ni más lento ni más rápido. Sin embargo, sí son muy útiles cuando nosotros vemos el código.

www.redusers.com 45

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

Una asignación no válida sería la siguiente:

**50 = costo;**

En este caso nada queda asignado a la variable **costo**. Siempre la variable que recibe el valor se coloca a la izquierda del signo igual.

Si lo necesitamos, es posible inicializar la variable al momento de declararla. Esto es muy cómodo y si lo hacemos así evitamos olvidar la inicialización de la variable. Pa ra hacerlo colocamos el tipo de la variable seguido del nombre a usar en la variable e inmediatamente con el operador de asignación colocamos el valor.

**int costo = 50;**

Comentarios en el programa

Los programas de computadora pueden ser muy grandes, por lo que es necesario colocar **comentarios** de codigo en el programa. Los comentarios sirven para docu mentar las partes del programa, explicar lo que hace cierta parte del código o sim plemente colocar un recordatorio.

Podemos hacer comentarios de una sola línea, que se colocan por medio de la do ble barra **//** seguido del texto del comentario. El texto que se encuentre a partir de la doble barra **//** y hasta el fin del renglón es ignorado por el compilador. Algunos ejemplos de comentarios pueden ser:

**// Declaro mis variables**

**int ancho, alto;**

**int tarea; // Esta variable guarda el área calculada.**

**// La siguiente sentencia nunca se ejecuta**

**// Console.WriteLine(“Hola”);**

Si llegáramos a necesitar colocar mucha información en nuestros comentarios, pue de ser útil explayarla en varios renglones, es decir, hacer un **bloque de comenta rios**. Para esto, abrimos nuestro bloque de comentarios con una barra seguida de un asterisco, **/\***, y finalizamos éste con un asterisco en primer lugar, seguido de una barra **\*/**. Todo lo que se encuentre entre estos delimitadores será tomado como un comentario y el compilador lo pasará por alto.

Veamos un ejemplo de un comentario de múltiples líneas:

46 www.redusers.com 

Los lenguajes de programación

**/\* La siguiente sección calcula el área y usa las variables alto – para guardar la altura del rectángulo**

**ancho – para guardar la base del rectángulo**

**\*/**

Así vemos cómo los comentarios colocados en varios renglones, se toman como un solo comentario. Esto es algo muy útil para recordatorios futuros.

Mostrar los valores de las variables en la consola

Ya que podemos guardar valores, también nos gustaría poder mostrarlos cuando sea necesario. Afortunadamente, por medio del método **WriteLine()** o el método **Write()** podemos hacerlo. En este caso no usaremos la cadena como antes, lo que usaremos es una **cadena de formato** y una **lista de variables**. La cadena de formato nos per mite colocar el mensaje a mostrar e indicar qué variable usaremos para mostrar su contenido y también en qué parte de la cadena queremos que se muestre. En la lis ta de variables simplemente colocamos las variables que queremos mostrar.

La cadena de formato se trata como una cadena normal, pero agregamos **{}** donde deseamos que se coloque el valor de una variable. Adentro de **{}** debemos colocar el **índice** de la variable en nuestra lista. Los índices inician en **cero**. Si queremos que se muestre el valor de la primera variable en la lista colocamos **{0}**, si es la segunda entonces colocamos **{1}** y así sucesivamente. La lista de variables simplemente se co loca después de la cadena, separando con comas las diferentes variables.

Por ejemplo, podemos mostrar el valor de una variable de la siguiente forma: **Console.WriteLine(“Se tiene {0} en la variable”, costo);**

Como solamente tenemos una variable en nuestra lista de variables entonces colo camos **{0}**. Pero también podemos mostrar el valor de dos variables o más, como lo muestra el siguiente ejemplo:

**Console.WriteLine(“La primera es {0} y la segunda es {1}”, costo, valor);**

Ahora veamos un ejemplo con lo que hemos aprendido.

**using System;**

**using System.Collections.Generic;**

****www.redusers.com 47

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

**using System.Text;**

**namespace AplicacionBase**

**{**

**class Program**

**{**

**// Esta es la función principal del programa**

**// Aquí inicia la aplicación**

**static void Main(string[] args)**

**{**

**// Declaramos variables**

**int costo;**

**int valor, precio;**

**// Inicializamos las variables**

**costo = 50;**

**valor = 75;**

**precio = 125;**

**// Declaramos e inicializamos**

**int impuesto = 10;**

**// Mostramos un valor**

**Console.WriteLine(“El valor adentro de costo es {0}”, costo);**

**// Varias variables**

**Console.WriteLine(“Valor es {0} y precio es {1}”, valor, precio);**

**// Dos veces la misma variable**

**Console.WriteLine(“Valor es {0} y precio es {1} con valor de {0}”, valor, precio);**

**/\* No olvidemos**

**mostrar el valor**

**de la variable impuesto**

**\*/**

**Console.WriteLine(“Y el valor que nos faltaba mostrar {0}”,impuesto);**

48 www.redusers.com 

Los lenguajes de programación

**}**

**}**

**}**

Una vez que hemos compilado la aplicación, obtendremos el resultado que mos tramos en la figura a continuación.

*Figura 7. En la consola observamos que los valores guardados*

*en las variables fueron desplegados de acuerdo con lo colocado en la cadena de formato.*

Operaciones aritméticas

Poder guardar la información es muy importante, pero es mucho más importante que podamos recuperar y manipular esa información para hacer algo útil con ella. Para esto, debemos comenzar a realizar las operaciones aritméticas básicas: suma, resta, multiplicación y división, algo muy común en cualquier sistema informático.

Cuando queremos o necesitamos realizar una operación aritmética debemos ha cer uso de un **operador**. Cada operación tiene su propio operador, que es el en cargado de procesar los números, realizando el determinado cálculo, para luego devolvernos el resultado deseado.

www.redusers.com

49

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

OPERADOR DESCRIPCIÓN

**=** Asignación. Este operador ya es conocido por nosotros.

**+** Suma. Nos permite sumar los valores de las variables o los números

**-** Resta. Para restar los valores de las variables o los números.

**\*** Multiplicación. Multiplica los valores de las variables o los números.

**/** División. Divide los valores de las variables o los números.

**%** Módulo. Nos da el residuo de la división.

*Tabla 2. Ésta es la tabla de operadores aritméticos en C#.*

Veamos algunos ejemplos de cómo utilizar estos operadores. Imaginemos que ya he mos declarado e inicializado las variables. Para guardar en la variable **resultado** la su ma de dos números, hacemos lo siguiente:

**resultado = 5 + 3;**

Como hemos visto, la expresión a la derecha se le asigna a la variable, por lo que en este caso la expresión **5 + 3** se evalúa y su resultado que es **8** es almacenado en la variable **resultado**. Al finalizar esta sentencia, la variable **resultado** tiene un va lor de **8**, que podrá ser utilizado dentro de la misma función o en el procedi miento donde fue declarado.

Las operaciones también se pueden realizar con las variables y los números. Suponga mos que la variable **a** ha sido declarada y se le ha asignado el valor de **7**.

**resultado = a – 3;**

De nuevo se evalúa primero la expresión **a-3**, como el valor adentro de **a** es **7**, en tonces la evaluación da el valor de **4**, que se le asigna a la variable **resultado**. Si queremos podemos trabajar únicamente con variables. Ahora supondremos que la variable **b** ha sido declarada y que le hemos asignado el valor **8**.

**resultado = a \* b;**

Entonces, se evalúa la expresión **a\*b** que da el valor **56**, que queda asignado a la va riable **resultado** al finalizar la expresión.

Si lo que deseamos es la división, podemos hacerla de la siguiente forma: **resultado = a / b;**

50 www.redusers.com 

Los lenguajes de programación

En este caso, se divide el valor de **a** por el valor de **b**. El resultado es **0,875**. Como el número tiene valores decimales, debemos usar variables de algún tipo que nos per mita guardarlos como **float** o **double**.

Y por último nos queda el módulo.

**resultado = a % b;**

En este caso recibimos en resultado el valor del residuo de dividir **a** por **b**. Veamos un pequeño programa donde se muestra lo aprendido en esta sección.

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace AplicacionBase**

**{**

**class Program**

**{**

**// Esta es la función principal del programa**

**// Aquí inicia la aplicación**

**static void Main(string[] args)**

**{**

**// Declaramos las variables, ahora de tipo flotante**

**float a, b, resultado;**

**// Inicializamos las variables**

**a = 7;**

**b = 8;**

CÓMO EVITAR ERRORES EN LAS EXPRESIONES MATEMÁTICAS

Los errores más comunes en las expresiones matemáticas son olvidar el punto y coma, hacer la asignación errónea, escribir mal el nombre de la variable y agrupar incorrectamente las operaciones. Utilizar los paréntesis para agrupar ayuda mucho a reducir errores.

www.redusers.com 51

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

**resultado = 0;**

**// Sumas**

**Console.WriteLine(“Sumas”);**

**resultado = 3 + 5;**

**Console.WriteLine(“Resultado = {0}”,resultado);**

**resultado = a + 3;**

**Console.WriteLine(“Resultado = {0}”, resultado);**

**resultado = a + b;**

**Console.WriteLine(“Resultado = {0}”, resultado);**

**// Restas**

**Console.WriteLine(“Restas”);**

**resultado = a - b;**

**Console.WriteLine(“Resultado = {0}”, resultado);**

**resultado = b - 5;**

**Console.WriteLine(“Resultado = {0}”, resultado);**

**resultado = b - a; // A la variable b se le resta a Console.WriteLine(“Resultado = {0}”, resultado);**

**// Multiplicaciones**

**Console.WriteLine(“Multiplicaciones”);**

**resultado = a \* 5;**

**Console.WriteLine(“Resultado = {0}”, resultado);**

**resultado = a \* 3.5f;**

**Console.WriteLine(“Resultado = {0}”, resultado);**

**resultado = a \* b;**

**Console.WriteLine(“Resultado = {0}”, resultado);**

**// Divisiones**

**Console.WriteLine(“Divisiones”);**

52 www.redusers.com 

**resultado = a / 3;**

Los lenguajes de programación

**Console.WriteLine(“Resultado = {0}”, resultado);**

**resultado = a / b;**

**Console.WriteLine(“Resultado = {0}”, resultado);**

**resultado = b / 2.5f;**

**Console.WriteLine(“Resultado = {0}”, resultado);**

**// Modulo**

**Console.WriteLine(“Modulo”);**

**resultado = a % b;**

**Console.WriteLine(“Resultado = {0}”, resultado);**

**}**

**}**

**}**

Podemos ver la ejecución en la siguiente figura:

*Figura 8. Vemos los resultados de las operaciones aritméticas sobre nuestras variables.*

**www.redusers.com 53

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

Precedencia de operadores

Hasta el momento hemos utilizado expresiones aritméticas sencillas, pero en la vi da real se pueden necesitar expresiones más complicadas. Entonces, si estas expre siones no se colocan adecuadamente, esto nos puede ocasionar problemas.

Por ejemplo, veamos la siguiente expresión.

**resultado = 3 \* 5 + 2;**

Esta expresión presenta un problema para nosotros en este momento. No sabemos si primero hace 3\*5 y le suma 2, que da como resultado 17, o si suma 5+2 y luego lo multiplica por 3, lo que da como resultado 21. ¿Cuál es la respuesta correcta? La forma de saberlo es por medio de la **precedencia de operadores**. Ésta nos in dica el orden en el que se llevan a cabo las operaciones. Este orden depende del tipo de operador que se utiliza. Algunos operadores tienen más precedencia que otros, es decir, se ejecutan primero.

La siguiente tabla muestra la precedencia de los operadores que hemos visto hasta el momento en C#. Se encuentran listados de mayor a menor precedencia.

OPERADOR DESCRIPCIÓN

**\*** Multiplicación

**/** División

**%** Módulo

**+** Adición

**-** Sustracción

*Tabla 3. Esta tabla nos muestra la precedencia de operadores*

*en C#. La multiplicación tiene más precedencia y se ejecuta primero.*

Esto quiere decir que cuando tenemos una expresión aritmética como la del ejem plo, primero se llevaría a cabo la multiplicación y luego la suma, entonces con esto podemos deducir cómo C# evaluaría la expresión.

Pero algunas veces sería mejor si nosotros pudiéramos organizar nuestra expresión con el fin de hacerla más fácil de leer para nosotros o para que podamos indicar de forma precisa cómo hacer la operación. Para que podamos organizar una expresión hacemos uso de los paréntesis. Cada sección que tengamos en los paréntesis se eva lúa como una expresión y el resultado se pasa al resto de la expresión. Supongamos que tenemos la siguiente expresión:

**resultado = (3\*5) + 2;**

54 www.redusers.com 

Los lenguajes de programación

Vemos que por medio de los paréntesis se ha agrupado una parte de la expresión, por lo que se evalúa y equivale a:

**resultado = 15 + 2;**

Lo que al final evalúa en:

**resultado = 17;**

Veamos otro ejemplo de cómo podemos usar los paréntesis para indicar cómo de seamos que se evalúe la expresión.

**resultado = 3 \* (5+2);**

Lo primero que sucede es que se evalúa la expresión contenida en el paréntesis. **resultado = 3 \* 7;**

Que nos da el valor:

**resultado = 21;**

Veamos un ejemplo un poco más complicado.

**resultado = (3+7) \* (36 + 4 \*(2+5));**

Veamos qué sucede:

**resultado = 10 \* (36 + 4 \* 7);**

Luego se continúa evaluando:

**resultado = 10 \* (36 + 28);**

****www.redusers.com 55

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

Lo que da como resultado:

**resultado = 10 \* 64;**

Y al final obtenemos:

**resultado = 640;**

Afortunadamente, C# hace todas estas evaluaciones y todos estos cálculos por no sotros, pero ahora sabemos cómo sucede y sabemos que podemos usar los parénte sis para organizar nuestras expresiones aritméticas.

Cómo pedirle datos al usuario

Hemos utilizado variables para guardar valores y con los operadores aritméticos hemos podido realizar cálculos matemáticos. Sin embargo, todos los valores que hemos usado se encuentran colocados directamente en el código del programa. Esto no es muy cómodo, ya que si deseamos hacer un cálculo con diferentes va lores, es necesario modificar el código y compilar nuevamente. Sería más útil si pudiéramos hacer que el usuario colocara los valores que necesita cuando el pro grama se ejecuta. Para esto, C# nos provee de un método que pertenece a la cla se **Console**. El método se llama **ReadLine()**. Éste no necesita ningún parámetro, por lo que sus paréntesis permanecerán vacíos, y nos regresa una cadena que con tiene lo que escribió el usuario con el teclado.

Ésta es una forma sencilla para pedirle información al usuario. Supongamos que dese amos pedirle al usuario su nombre, que utilizaremos a posteriori para saludarlo. Para eso colocamos el siguiente código:

**string entrada = ” ”;**

AGRUPACIÓN DE LOS PARÉNTESIS

Un error común con expresiones grandes es olvidar cerrar un paréntesis en forma adecuada. En Visual Studio es fácil ver cuáles paréntesis son pareja ya que al seleccionar uno de ellos su pareja se ilumina. Para cada paréntesis de apertura forzosamente debe de existir un paréntesis de cierre. Cuando esto no se logra se denomina desbalance de paréntesis.

56 www.redusers.com 

Los lenguajes de programación

**Console.WriteLine(“Escribe tu nombre”);**

**entrada=Console.ReadLine();**

**Console.WriteLine(“Hola {0}, como estas?”,entrada);**

En este fragmento de programa es importante notar cómo el valor obtenido por **ReadLine()** se le asigna a la variable de tipo cadena **entrada**. Si no colocamos una variable que recibe el valor de lo escrito por el usuario, éste se perderá. Cuando usemos **ReadLine()** debemos hacerlo como se muestra. Una vez que recibimos la entrada del usuario podemos trabajar con ella sin problemas.

Conversión de variables

El método de **ReadLine()** nos presenta una limitación en este momento ya que só lo regresa el valor como cadena, pero no podemos utilizar una cadena en operacio nes aritméticas o asignarla directamente a una variable numérica. Lo que debemos hacer es convertir ese dato numérico guardado como cadena a un valor numérico útil para nuestros propósitos. C# nos provee de una clase que per mite hacer conversiones entre los diferentes tipos de variables. Esta clase se cono ce como **convert** y tiene una gran cantidad de métodos para la conversión. Aquí sólo veremos cómo convertir a enteros y flotantes, ya que son los tipos con los que trabajaremos en este libro principalmente. Sin embargo, los demás métodos se usan de forma similar y podemos encontrar la información necesaria sobre éstos en **MSDN (***MicroSoft Developer Network***)**.

Si queremos convertir del tipo **string** al tipo **int** usaremos el método **ToInt32()**. És te necesita un parámetro, que es la cadena que deseamos convertir. El método re gresa un valor de tipo **int**, por lo que es necesario tener una variable que lo reciba.

Un ejemplo de esto es:

**a = Convert.ToInt32(entrada);**

MSDN

**MSDN** es un sitio web donde encontraremos información técnica sobre todas las clases y funciones, y todos los métodos que conforman los lenguajes que componen Visual Studio. También es posible encontrar ejemplos y descripciones de los errores de compilación. Tiene un buen sistema de búsquedas que resulta muy útil. La dirección es **www.msdn.com**.

www.redusers.com 57

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

De forma similar podemos convertir la cadena a una variable de tipo **float**. En este caso usaremos el método **ToSingle()**. Éste también necesita un parámetro, que es la cadena que contiene el valor a convertir. Como regresa un valor flotante, entonces debemos tener una variable de tipo **float** que lo reciba.

**valor = Convert.ToSingle(entrada);**

Ya que tenemos los valores convertidos a números, entonces podemos hacer uso de ellos. Veamos un ejemplo donde le pedimos información al usuario.

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace AplicacionBase**

**{**

**class Program**

**{**

**// Esta es la función principal del programa**

**// Aquí inicia la aplicación**

**static void Main(string[] args)**

**{**

**// Declaramos variables**

**string entrada = “”;**

**int a = 0, b = 0, resultado = 0;**

**// Leemos una cadena**

**Console.WriteLine(“Escribe tu nombre”);**

**entrada = Console.ReadLine();**

**Console.WriteLine(“Hola {0}, como estas?”, entrada);**

**// Leemos dos valores y los sumamos.**

**Console.Write(“Dame un entero:”);**

**entrada = Console.ReadLine();**

**// Convertimos la cadena a entero**

**a = Convert.ToInt32(entrada);**

58 www.redusers.com 

Los lenguajes de programación

**Console.Write(“Dame otro numero entero:”);**

**entrada = Console.ReadLine();**

**// Convertimos la cadena a entero**

**b = Convert.ToInt32(entrada);**

**// Sumamos los valores**

**resultado = a + b;**

**// Mostramos el resultado**

**Console.WriteLine(“El resultado es {0}”, resultado);**

**}**

**}**

**}**

Ahora compilemos y ejecutemos la aplicación. Para introducir los datos, simplemen te los escribimos con el teclado y al finalizar oprimimos la tecla **ENTER** o **RETURN**.

*Figura 9. Éste es el resultado de la ejecución, donde vemos*

*que la suma se lleva a cabo con los valores introducidos por el usuario. *www.redusers.com 59

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

Cómo resolver problemas en la computadora Ya que conocemos algunos elementos del lenguaje, ahora es importante apren der cómo resolver los problemas en la computadora. Existen diferentes metodo logías para hacerlo, pero como éste es un libro dirigido a gente sin experiencia en programación aprenderemos un método sencillo. Como ya hemos mencio nado, C# es un lenguaje orientado a objetos. Sin embargo, primero aprendere mos una metodología estructurada, lo que nos permitirá tener experiencia en la resolución de problemas y luego podremos aprender la metodología orientada a objetos de manera más sencilla.

La metodología que aprendemos se basa en el concepto de subdivisión de problemas. Es decir que tomamos el problema general y luego lo dividimos en problemas más pequeños y si es necesario, dividimos estos subproblemas en problemas más peque ños, hasta que podamos resolver el subproblema directamente.

Para resolver cualquier problema, lo primero que tenemos que hacer es entenderlo. Hay que entender de forma precisa qué es lo que se pide. Los problemas tendrán información. En muchos casos, el problema nos da directamente la información. A este tipo de información se lo conoce como **información explícita**. Pero también a veces sucede que el problema no nos da directamente la información, por eso es importante entenderlo bien. En este caso nosotros debemos inferir la información de otros datos que nos proporciona o buscarla en algún otro lugar. A este tipo de información se lo conoce como **información implícita**. Afortunadamente, la in formación implícita en muchos casos es fácil de encontrar o forma parte de los co nocimientos generales de cualquier persona.

Ya que tenemos el problema subdividido y los datos reconocidos, procedemos a re alizar el algoritmo. Recordemos que el algoritmo es la serie de pasos necesarios pa ra resolver el problema. Además, éste puede definirse por medio de un diagrama de flujo o escribirse en algo que denominaremos **pseudocódigo**. El programador pue de crear el algoritmo con su método de preferencia.

Cuando se ha terminado el algoritmo, procedemos a probarlo. En esta etapa pode mos probar en el papel si todo funciona bien. Si hubiera algún resultado erróneo, entonces se puede corregir el algoritmo y el proceso se repite hasta que tengamos un algoritmo que funcione adecuadamente.

CONVERSIÓN DE NÚMEROS A CADENAS

Tambien es posible convertir de números a cadenas. Para esto se usa el método **ToString()**, que está disponible en todos los tipos numéricos de C#. De esta forma podemos utilizar la información de las variables numéricas en funciones o métodos que trabajen con cadenas. El método no cambia el tipo, simplemente regresa una cadena que contiene el texto de ese valor.

60 www.redusers.com 

Los lenguajes de programación

El último paso consiste en pasar el algoritmo a programa de computadora. Si to do se ha realizado correctamente, el programa deberá ejecutarse al primer inten to. Mucha gente se sorprende que para hacer un programa de computadora, el úl timo paso sea colocarlo en la computadora, pero ésta es la forma correcta. Algu nos programadores no profesionales intentan hacer el programa directamente en la computadora, pero esto acarrea muchos errores de lógica y datos, y el tiempo en que resuelve el mismo problema es mayor y es difícil que se ejecute correcta mente al primer intento. Aunque la metodología que acabamos de presentar pa rece larga, en realidad puede ahorrar hasta un 50% del tiempo de resolución en comparación con alguien que programe directamente en la computadora. Ahora que empezamos, es bueno adquirir hábitos correctos de programación desde el inicio. En el futuro, en especial cuando los problemas que se resuelvan sean com plicados, se apreciarán los beneficios.

Elementos de un diagrama de flujo

Es momento de aprender los componentes básicos de un **diagrama de flujo**. Con estos diagramas podemos representar el algoritmo y son fáciles de hacer y entender. Otra ventaja que tienen es que son independientes del lenguaje, lo que nos da la fle xibilidad de resolver el problema y usar nuestra solución en cualquier lenguaje.

El diagrama de flujo es una **representación gráfica del algoritmo** y tiene diferentes componentes. Cada componente es una figura y ésta representa un tipo de actividad. En el interior de la figura colocamos una descripción del paso que está sucediendo ahí. Por medio de flechas indicamos cuál es el paso siguiente y nos da la idea del flujo general del algoritmo. En algunas ocasiones, a las flechas les agregamos información extra, pero esto lo veremos más adelante. Aunque en este momento no utilizaremos todos los componentes del diagrama de flujo, sí conoceremos los más importantes, ya que en los capítulos posteriores los utilizaremos.

El primer componente tiene forma de ovalo y se lo conoce como **terminador**. És te nos sirve para indicar el inicio o el final del algoritmo. Al verlo podemos saber inmediatamente dónde inicia el algoritmo y poder continuar a partir de ahí. Hay que recordar que los algoritmos deben tener un final para que sean válidos. Si el ter minador es de inicio, colocamos en su interior el mensaje: **Inicio**. Si el terminador indica dónde se acaba el algoritmo, entonces se coloca el mensaje: **Final**.

Inicio Final

*Figura 10. En esta figura podemos observar el terminador tanto*

*en su papel como punto de inicio como punto final del algoritmo.*

**www.redusers.com 61

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

El siguiente elemento se conoce como **proceso**. Éste tiene la forma de un rectán gulo y nos sirve para indicar que precisamente un proceso debe llevarse a cabo en ese lugar. La descripción del proceso se indica en su interior.

A = b + c

*Figura 11. El proceso se indica por medio*

*de un rectángulo. En su interior indicamos lo que se tiene que procesar.*

Similar al proceso, tenemos el **proceso predefinido**. Su forma también es un rec tángulo, pero éste tiene dos franjas a los lados. Se usa para indicar que haremos uso de un proceso que ya ha sido definido en otra parte. Un ejemplo de esto sería un método, como los que ya hemos usado. Principalmente lo usaremos para indicar llamadas a funciones o métodos definidos por nosotros mismos. En un capítulo pos terior aprenderemos cómo hacer nuestros propios métodos.

Tolnt32()

*Figura 12. Éste es el símbolo usado para el proceso predefinido.*

Nuestro próximo elemento es conocido como **condición**. Utilizaremos este ele mento cuando aprendamos cómo hacer que el programa tome sus propias

DATOS DE TRABAJO

Algunas veces necesitaremos variables que nos apoyen para resolver algún cálculo o proceso. A éstas las llamaremos datos de trabajo. Los datos de trabajo se descubren cuando hacemos el análisis del problema. Si llegamos a omitir alguno durante el análisis siempre es posible agregarlo durante el desarrollo, pero esto debe ser la excepción a la regla.

62 www.redusers.com 

Los lenguajes de programación

decisiones. Básicamente, sirve para hacer una selección entre las posibles rutas de ejecución del algoritmo, que dependerá del valor de la expresión evaluada. Su forma es la de un rombo y a partir de las esquinas de éste surgirán las flechas ha cia las posibles rutas de ejecución del algoritmo.

A > B

*Figura 13. La decisión se representa con un rombo.*

Luego nos encontramos con un componente conocido como **datos**. Su forma es la de un paralelogramo y podemos usarlo para indicar la salida o la entrada de los datos. Un ejemplo de esto sería cuando llevamos a cabo la petición de un va lor al usuario o cuando mostramos un valor en pantalla. Adentro de él indica mos qué dato se pide o se presenta.

Pedir A

*Figura 14. En este caso usamos el dato*

*para indicar la petición del valor de una variable.*

Hemos visto los elementos principales del diagrama de flujo. Ahora podemos hacer un ejercicio que muestre la solución de un problema en la computadora.

C# EXPRESS...

Es una versión gratuita de C# con las herramientas básicas para programar en este lenguaje. Esta versión nos provee todo lo necesario para los temas que aprenderemos en este libro. Aunque la versión es gratuita, es necesario registrarla para poder utilizarla por más de treinta días. La podemos descargar de **http://msdn2.microsoft.com/en-us/express/aa700756.aspx**.

www.redusers.com 63

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

Resolución de problemas en la computadora

Empecemos por un problema sencillo y fácil de entender. Para este problema se guiremos todos los pasos que hemos mencionado hasta llegar al programa de cóm puto final. En primer lugar, tenemos la descripción del problema. Hacer un programa de cómputo que calcule el área y el perímetro de un rectángulo dados sus lados. Una vez que tenemos el problema y lo hemos entendido, entonces procedemos a subdividirlo en problemas más pequeños que sean fáciles de resolver. Esta subdivisión puede ser hecha por medio de un diagrama y para esto hay que re cordar que si alguno de los problemas aún continua siendo difícil, lo podemos seguir subdividiendo. A modo de ejemplo exageraremos un poco la subdivisión en este ca so. El problema quedaría subdividido como se muestra en la siguiente figura.

Calcular área y perímetro 

del rectángulo

Pedir

Pedir datos Calcular área Pedir

Calcular 

perímetro

Mostrar

Mostrar 

resultados

Mostrar

ancho

alto

área

perímetro

*Figura 15. Aquí tenemos nuestro problema*

*subdividido en problemas menores y fáciles de resolver.*

Ya que conocemos el problema y lo hemos subdividido en varios subproblemas, po demos identificar de manera más clara los datos que nos hacen falta para poder tra bajar en él. La descripción del problema nos proporciona dos datos explícitos: el

MSDN

Si queremos encontrar información sobre cualquier clase o método de C#, el lugar apropiado para hacerlo es el sitio web MSDN que nos provee Microsoft. Cuando realicemos la búsqueda debemos filtrar para buscar únicamente contenidos del lenguaje C# y así evitar confusiones con información de otros lenguajes. La dirección es **www.msdn.com**.

64 www.redusers.com 

Los lenguajes de programación

**área** y el **perímetro**. Pero también reconocemos inmediatamente dos datos implíci tos que son necesarios: el **alto** y el **ancho** del rectángulo.

Nosotros sabemos que necesitamos una cadena para recibir el valor escrito por el usuario, pero como tal, esta cadena no forma parte del problema. Sin embargo, nos ayuda a resolverlo, por lo que la cadena será un dato de trabajo. Los datos que he mos identificados serán colocados en una tabla de la siguiente forma:

NOMBRE TIPO VALOR INICIAL

**área** Float 0.0

**perímetro** Float 0.0

**ancho** Float 1.0

**alto** Flota 1.0

**valor** String “”

*Tabla 4. Tabla de variables para el programa que estamos resolviendo.*

Hemos decidido usar el tipo **float**, ya que podemos tener valores decimales en el cál culo del perímetro. Como un rectángulo con ancho o alto de 0 no tiene sentido, hemos seleccionado como valor inicial para estas variables a 1. Ahora viene el paso más importante: la creación del algoritmo. Sin embargo, si ob servamos la forma como subdividimos el problema, podemos ver que nos da una indicación de los pasos que son necesarios colocar en el algoritmo.

Calcular área y perímetro 

del rectángulo

Pedir

Pedir datos Calcular área Pedir

Calcular 

perímetro

Mostrar

Mostrar 

resultados

Mostrar

ancho

alto

área

perímetro

*Figura 16. La subdivisión nos muestra una lista*

*de los pasos posibles necesarios en la subdivisión del algoritmo.*

Con esta información creamos el algoritmo con el diagrama de flujo. En éste apare ce la secuencia de pasos en el orden correcto. Veamos cómo queda el diagrama.

www.redusers.com 65

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

Inicio 





Pedir ancho



Pedir alto



Área = ancho • alto

Perímetro = 

(ancho + alto) • 2



Mostrar área



Mostrar

perímetro





Fin

*Figura 17. El diagrama de flujo muestra el algoritmo.*

*Las diferentes figuras nos indican el tipo de actividad que se realiza.*

Podemos observar cómo hemos utilizado el trapezoide cuando es necesario pedirle o mostrarle un dato al usuario. Los cálculos con la fórmula correspondiente están en los rectángulos, ya que son procesos, y el inicio y fin del algoritmo están re presentados por los terminadores.

66 www.redusers.com 

El código de nuestra aplicación

Los lenguajes de programación

Ya tenemos todo lo necesario para poder crear el código del programa. Para esto ha remos uso de la aplicación base que ya tenemos. Como creamos una tabla de datos, ya conocemos todas las variables necesarias para la aplicación. Entonces tomamos esta tabla y colocamos los datos como variables.

**// Declaramos las variables que necesitamos**

**float area = 0.0f;**

**float perimetro = 0.0f;**

**float ancho = 1.0f;**

**float alto = 1.0f;**

**string valor = ””;**

Con las variables definidas, entonces procedemos a codificar cada uno de los pasos del algoritmo. Recorremos paso por paso y en el mismo orden en que se encuen tran, y vemos que lo primero es pedir el ancho del rectángulo.

**Console.Write(“Dame el ancho del rectángulo: “);**

**valor = Console.ReadLine();**

**ancho = Convert.ToSingle(valor); // Convertimos a flotante**

Lo siguiente que hace el algoritmo es pedir el alto del rectángulo, entonces el códi go necesario es el siguiente.

**Console.Write(“Dame el alto del rectángulo: “);**

**valor = Console.ReadLine();**

**alto = Convert.ToSingle(valor); // Convertimos a flotante**

EXPERIMENTANDO

Aprender a programar requiere de mucha práctica y mucha experimentación. Es bueno probar con los programas y ver qué es lo que sucede. No debemos temer en cometer errores cuando estamos aprendiendo. También es bueno verificar la documentación para las funciones, las clases y los métodos ya que es posible encontrar utilidades en éstas para el futuro.

www.redusers.com 67

2. LOS ELEMENTOS BÁSICOS DE UN PROGRAMA

Una vez que hemos obtenido todos los datos de entrada, podemos decir que ya te nemos un proceso. Este proceso será el encargado de calcular el área que ocupa el rectángulo. Veamos el ejemplo a continuación:

**area = ancho \* alto;**

En el próximo proceso que tenemos se calcula el perímetro.

**perimetro = (ancho + alto) \* 2;**

Si vemos con detenimiento el algoritmo y también su paso siguiente, encontra remos que volvemos a trabajar con datos nuevamente, pero en este último caso sólo mostramos en pantalla el valor de área calculado.

Veamos para comprender mejor de qué hablamos.

**Console.WriteLine(“El área es : {0} unidades cuadradas”, area);**

Como último paso del algoritmo que se está ejecutando, sólo queda que mostre mos en nuestra aplicación de consola, el valor calculado del perímetro. A continuación escribamos el último código necesario en nuestra aplicación:

**Console.WriteLine(“El perímetro es : {0} unidades”, perimetro);**

Hemos visto hasta aquí una forma completa y compleja de cómo manejar desde una aplicación de consola, la manipulación de datos, variables y operaciones ma temáticas, con un claro ejemplo estudiado paso a paso. Con esto ya hemos fina lizado el código de la aplicación. Ahora lo podemos compilar y probar, y todo de be funcionar correctamente, si no hemos olvidado de colocar nada en él.

SITIO CON RECURSOS SOBRE C#

Un sitio web que nos presenta diferentes recursos de C# y cuyo contenido está en idioma inglés es: **www.csharphelp.com**. Aquí podremos encontrar información cuando tengamos algún problema con el lenguaje o deseemos aprender cómo se realiza algo en particular. También hay información para programar con formas de Windows.

68 www.redusers.com 

Los lenguajes de programación

*Figura 18. Aquí podemos ver la ejecución de nuestro*

*programa, la forma como pide los datos y los resultados calculados.*

**…** RESUMEN

**Para programar software para computadoras necesitamos conocer un lenguaje de programación. Uno de estos lenguajes es C#. Los programas deben ser ordenados e indicar paso a paso lo que se debe hacer, y para esto hacemos uso de los algoritmos. La información se guarda en variables y el tipo indica la información que puede tener en su interior por lo que es posible mostrar datos en la consola y también pedírselos al usuario. Los operadores aritméticos nos permiten llevar a cabo operaciones matemáticas, siempre que usemos una metodología correcta de resolución de problemas.**

****www.redusers.com 69

ACTIVIDADES

TEST DE AUTOEVALUACIÓN 1 ¿Qué es un algoritmo?

2 ¿Qué características tienen los

algoritmos?

3 ¿Qué es una sentencia?

4 ¿Cómo se finalizan las sentencias?

5 ¿Cuál es la diferencia entre Write() y WriteLine()?

6 ¿Qué es una cadena?

7 ¿Qué es una variable?

8 ¿Cómo mostramos el valor de una variable?

9 ¿Cómo le solicitamos un dato al usuario?

10 ¿Cómo se convierte una cadena a un valor numérico?

11 ¿Qué son los operadores aritméticos?

12 ¿De qué forma podemos agrupar operaciones aritméticas?

EJERCICIOS PRÁCTICOS

1 Hacer un programa que calcule el perímetro de cualquier polígono regular.

2 Generar en este proyecto un error a propósito y ver cómo se comporta el compilador.

3 Hacer un programa que transforme de grados a radianes.

4 Hacer un programa que transforme de grados centígrados a grados Fahrenheit.

5 Hacer un programa que transforme entre dólares y euros y que también pida el tipo de cambio del día.

70 www.redusers.com 

C#

El programa toma

decisiones

En los programas creados hasta ahora, hemos visto que los algoritmos tenían un solo sentido de resolución. En este capítulo aprenderemos otras formas diferentes de hacer que nuestros programas tomen decisiones en base al análisis de diversos parámetros, y también aprenderemos a crear algoritmos con diferentes rutas

de ejecución para poder solucionar problemas en base a un previo análisis.

**SERVICIO DE ATENCIÓN AL LECTOR: usershop@redusers.com**

Capítulo 3

**La toma de decisiones 72** Expresiones relacionales 72 El uso de if 76 El uso de else 84 Cómo usar if anidados 89 Escalera de if-else 92 Expresiones lógicas 96 El uso de switch 105

**Resumen 109 Actividades 110**

3. EL PROGRAMA TOMA DECISIONES

LA TOMA DE DECISIONES

No todos los problemas se resuelven linealmente, a veces es necesario tener que tomar una decisión o ejecutar determinadas acciones cuando una condición se cumple, y otras cuando no lo hace. Supongamos que nuestro problema consiste en mantener la temperatura de un balde con agua tibia. Para hacerlo nosotros po demos agregar agua caliente o agua fría. En este problema necesitaríamos tomar una decisión sobre qué tipo de agua agregar. De igual forma, hay muchos pro blemas en los que para poder resolverlos necesitamos conocer una **condición** o tomar una **decisión**. En C# es sencillo poder lograr esto, ya que el lenguaje nos provee diferentes herramientas para poder lograrlo. Tendremos que utilizar ex presiones y éstas se evaluarán. En este caso usaremos **expresiones relacionales y expresiones lógicas**, que se evaluarán, y dependiendo del resultado de esa eva luación, se llevarán a cabo ciertos pasos del algoritmo u otros.

Empecemos por conocer las expresiones que necesitamos.

Expresiones relacionales

Las **expresiones relacionales** se usan para expresar la **relación** que existe entre dos valores. Los valores pueden estar contenidos adentro de **variables** o ser colocados **explícitamente**. Estas expresiones, al igual que las expresiones aritméticas, tienen sus propios **operadores**. La expresión será evaluada, pero el resultado de la evalua ción tendrá únicamente dos valores posibles: **true** o **false**.

Ambos valores son de tipo **bool** y **true** es usado para indicar que la expresión eva luada es **verdadera**. El valor de **false** por su parte se utiliza para indicar que la expresión evaluada es **falsa**.

Empecemos por conocer primero a los operadores relacionales y luego veremos ejemplos de expresiones relacionales con su resolución.

Operadores relacionales

En la **tabla 1** podemos apreciar los operadores relacionales en C#. La forma como es tán escritos sus signos es la forma como debemos colocarlos en nuestro programa.

ERROR MUY COMÚN CON LA IGUALDAD

Un error muy común que sucede cuando se empieza a programar en C# es el de confundir el operador de igualdad con el de asignación. Si se obtiene un error en una expresión, hay que verificar esto. No olvidemos que la asignación lleva un signo igual y la igualdad doble signo igual.

72 www.redusers.com 

La toma de decisiones

SIGNO OPERADOR

**==** Igualdad

**!=** No igual

**>** Mayor que

**<** Menor que

**>=** Mayor que igual

<= Menor que igual

*Tabla 1. Esta tabla nos muestra los diferentes operadores relacionales de C#.*

El operador de **igualdad** se representa con dos signos igual juntos y sin espacio. Co locar un solo signo igual, indica que el operador será de **asignación**. Esto puede de rivar en errores de lógica, porque una asignación siempre evalúa a verdadero, pero una igualdad no. Este operador es el más sencillo. Supongamos que ya tenemos las variables y las inicializamos con los siguientes valores:

**int a = 5;**

**int b = 7;**

**int c = 5;**

**int d = 4;**

Ahora crearemos nuestra primera expresión y la evaluaremos:

**a == c**

Como el valor contenido en **a** es **5** y el valor contenido en **c** también es **5**, vemos que se cumple **5** igual a **5**, por lo que la expresión se evalúa como **true**. Veamos qué sucede en la siguiente expresión:

**a == d**

En este caso el valor contenido en **d** es **4**, por lo que la expresión **5** igual a **4** no se cumple y el valor de la expresión es **false**. En el lado derecho y en el lado iz quierdo de la expresión podemos colocar cualquier valor, variable o expresión. Si colocásemos una expresión, primero se evaluaría ésta y luego se procedería a evaluar la expresión relacional principal.

**a == (3 + 2)**

****www.redusers.com 73

3. EL PROGRAMA TOMA DECISIONES

En este caso **3 + 2** se evalúa como **5** y luego **5** igual a **5**, lo que nos da como valor final de la expresión **true**. Otro ejemplo es el siguiente:

**(b – 2) == d**

Al evaluar **b - 2** obtenemos **5**, entonces la expresión es **5** igual a **4**, lo que evidente mente nos da como resultado final **false**.

Ésta es la forma como se evalúan las expresiones relacionales y ahora veremos más ejemplos con otros operadores. También tenemos al operador de **desigualdad**, que se crea por medio de un signo de admiración y luego el signo de igual.

**a != 7**

En este caso tenemos **5** no es igual a **7**, lo cual es cierto, y obtenemos el valor **true** como resultado de la evaluación.

**a != c**

Para esta expresión tenemos **5** no es igual a **5**, y se evalúa la expresión como falsa, lo que nos da **false** como valor final. El operador **>** sirve para evaluar una relación del tipo **mayor que**. Si el valor del lado izquierdo es mayor que el valor del lado de recho, regresará **true**, en caso contrario regresará **false**.

**b > a**

Esto es **7** mayor que **5** y da como resultado **true**.

**a > b**

En este caso **5** mayor que **7** es falso y el resultado de la expresión es **false**. **b > b**

Este expresión nos da **6** mayor que **6**, lo cual, si analizamos todos los pasos hasta aquí, nos indica que es falso también y obtenemos **false** como resultado.

74 www.redusers.com 

La toma de decisiones

En el operador **<** evaluamos una relación del tipo **menor que**. Ésta da **true** como resultado si el valor del lado izquierdo es menor que el valor que encontramos en el lado derecho. Si esto no se cumple, entonces el valor regresado será **false**.

**d < a**

Tenemos **4** menor que **5**, lo que resulta cierto y el resultado es **true**. **d < (a - 3)**

La expresión a evaluar es **4** menor que **2**, y se tiene **false** como resultado de la expresión.

**d < d**

Esta expresión también da como resultado **false**, ya que **4** menor que **4** no es ver dadera. Para poder evaluar si un valor es **mayor que o igual** a otro valor usamos el operador **>=**. Su forma de trabajo es similar a la de los otros operadores.

**b >= a**

Para esta expresión tenemos **7** mayor que igual a 5, es cierta y da **true** de resultado. **b >= (a + c)**

En este caso obtendríamos **7** mayor que igual a **10**, lo cual sabemos que es falso y y, por lo cual, como resultado obtendríamos **false**.

COLOCAR LOS OPERADORES ADECUADAMENTE

Los operadores que se escriben con dos signos deben escribirse correctamente. No es lo mismo **>=** que **=>**. Esto puede llevarnos a errores de sintaxis, pero al escribirlos en la forma correcta se verán corregidos. También debemos recordar que se escriben sin espacios entre los signos, ya que esto nos lleva a otro error de sintaxis.

www.redusers.com 75

3. EL PROGRAMA TOMA DECISIONES

**b >= b**

Aquí comparamos la diferencia de resultado con relación a usar **>=** en lugar de **>**. Pa ra esta expresión evaluamos **7** mayor que igual a **7**. Esto es verdadero y el resultado final es **true**. De igual manera tenemos al operador **<=**, pero éste evalúa a la inversa.

**a <= b**

La expresión a evaluar es **5** menor que igual a **7**, y obtenemos **true** como resultado. **(a + c) <= b**

En este caso tenemos **10** menor que igual a **7** y resulta falso.

**(a + 2)<= b**

En esta última expresión **7** menor que igual a **7** es verdadero y nuevamente obte nemos **true** como resultado.

El uso de if

Ya aprendimos a usar las expresiones relacionales y el tipo de valor devuelto. Es el momento de hacer algo práctico con ellas. Este tipo de expresiones se usa en diferentes casos, pero generalmente en las estructuras selectivas o repetitivas. Las estructuras selectivas serán estudiadas en este capítulo, las repetitivas en un capí tulo posterior. Las **estructuras selectivas** son aquellas que nos permiten hacer una selección entre dos o varias rutas de ejecución posibles. La selección se llevará a cabo según el valor de una expresión. Esta expresión puede ser una expresión relacional. Nuestra primera estructura selectiva se conoce como **if**, que es un **si condicional**. Si tal cosa sucede, entonces haz tal cosa. El uso del **if** es sencillo:

**if(expresión)**

**Sentencia a ejecutar**

El uso de **if** requiere que coloquemos una expresión a evaluar entre paréntesis. Si el resultado de esta expresión es **true**, entonces la sentencia a ejecutar se lleva a cabo.

76 www.redusers.com 

La toma de decisiones

Si el resultado de la evaluación es **false**, entonces la sentencia a ejecutar nunca se lle va a cabo, o dicho de otra forma, es ignorada.

En el diagrama de flujo **if** se representa por medio de un **rombo**. En el interior del rombo colocamos la expresión a evaluar, de las esquinas sacamos una ruta de ejecu ción en el caso de que sí se cumpla la condición o en el caso de que no se cumpla.

Expresión

SI NO

*Figura 1. Este rombo simboliza a* ***if*** *con la expresión en su interior.*

Veamos un primer ejemplo donde nos puede servir **if** y las expresiones. Crearemos un programa que le pida al usuario un número, y la computadora debe decir si el número es positivo o negativo. Para ver el algoritmo de este programa, creamos su diagrama de flujo e incluimos los **if** necesarios.

Inicio 



Pedir número



Número>=0 SI



NO 

Número>0 SI

Mostrar "El número es positivo"



NO Mostrar "El número 

es negativo"

Fin

*Figura 2. Éste es el diagrama de flujo del algoritmo para resolver el problema. Podemos observar que tenemos dos rutas de ejecución en cada* ***if****.*

**www.redusers.com 77

3. EL PROGRAMA TOMA DECISIONES

**using System;**

**using System.Collections.Generic;**

**using System.Text;**

**namespace AplicacionBase**

**{**

**class Program**

**{**

**// Esta es la función principal del programa**

**// Aquí inicia la aplicación**

**static void Main(string[] args)**

**{**

**int numero = 0; // Donde guardamos el número**

**string valor = “”; // Para guardar la cadena dada por el usuario**

**// Pedimos el número**

**Console.Write(“Dame un numero entero:”);**

**Valor = Console.ReadLine();**

**numero = Convert.ToInt32(valor); // Convertimos la cadena a entero**

**// Hacemos uso de if con la expresión para el caso de los positivos if (numero >= 0)**

**Console.WriteLine(“El numero {0} es positivo”, numero); // se ejecuta si se cumple numero>=0**

**// Hacemos uso de if con la expresión para el caso de los negativos if (numero < 0)**

**Console.WriteLine(“El numero {0} es negativo”, numero); // se ejecuta si se cumple numero<0**

**}**

EL USO CORRECTO DE LA SENTENCIA CON IF

Cuando hacemos uso de **if**, nunca debemos colocar punto y coma después del paréntesis ya que hacerlo no es un error de sintaxis sino de lógica. El compilador interpreta esto como si deseamos que se ejecute una sentencia vacía cuando **if** se cumple. Si vemos que la sentencia **if** siempre se cumple sin importar cómo se evalúa la expresión, es posible que tengamos este error.

78 www.redusers.com 