Universidad Nacional Autónoma de México
Programa de Tenología en Cómputo
Lo que siempre quisiste saber de C# y nunca te atreviste a preguntar

Elaborado por:

Rivera Negrete Manuel Armando

28 de Junio del 2018

Índice general

¿Por qué aprender C#?

Parte I

C# Básico

Capítulo 1

La arquitectura .NET

1.1. C# y su lugar dentro de .NET

C# (leído «C sharp») es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como partes de su plataforma .NET. Aunque esta plataforma permite desarrollar aplicaciones en otros lenguajes de programación, C# ha sido creado específicamente para .NET, adecuando todas sus estructuras a las características y capacidades de dicha plataforma.
.NET es un framework de Microsoft que hace un énfasis en el desarrollo sencillo de aplicaciones, independencia de hardware y transparencia de redes. Es una implementación de Common Language Infraestructure (Estandar CLI). Un Framework es un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación.

1.2. Common Language Runtime

El Common Language Runtime o CLR es un entorno de ejecución que ejecuta el código y proporciona servicios que facilitan el proceso de desarrollo de los programas que corren sobre la plataforma Microsoft .NET.
Los compiladores y las herramientas exponen la funcionalidad del tiempo de ejecución del idioma común y le permiten escribir código que se beneficia de este entorno de ejecución administrada. El código que desarrolla con un compilador de lenguaje que se dirige al tiempo de ejecución se denomina código administrado; se beneficia de características tales como la integración entre idiomas, manejo de excepciones entre idiomas, seguridad mejorada, soporte de versiones e implementación, un modelo simplificado para la interacción de componentes y servicios de depuración y creación de perfiles. Para enternder

el proceso de compilación en C# es necesario definir algunos conceptos:
Compilación: La tarea de compilar se refiere al proceso de traducción del código fuente a código entendible por la computadora, entendiéndose por código fuente las líneas de código que se han escrito en un lenguaje de programación, en este caso un lenguaje de programación de alto nivel.
Código Máquina: Es el sistema de códigos directamente interpretable por un circuito microprogramable, como el microprocesador de una computadora, es decir, es un lenguage que entiende la computadora.
ByteCode: Es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario producido por el compilador cuyo contenido es el código objeto o código máquina.

1.3. Microsoft Intermediate Language

MSIL significa Microsoft Intermediate Language. Podemos llamarlo Lenguaje Intermedio (IL) o Lenguaje Intermedio Común (CIL). Durante el tiempo de compilación, el compilador convierte el código fuente en Microsoft Intermediate Language (MSIL). Microsoft Intermediate Language (MSIL) es un conjunto de instrucciones independiente de la CPU que se puede convertir de manera eficiente al código nativo. Durante el tiempo de ejecución, el compilador Just In Time (JIT) de Common Language Runtime (CLR) convierte el código de Microsoft Intermediate Language (MSIL) en código nativo al sistema operativo.
El código fuente escrito en C# se compila en un lenguaje intermedio (IL) que guarda conformidad con la especificación de CLI. El código y los recursos IL, como mapas de bits y cadenas, se almacenan en disco en un archivo ejecutable denominado ensamblado, normalmente con la extensión .exe o .dll. Un ensamblado contiene un manifiesto que proporciona información sobre los tipos, la versión, la referencia cultural y los requisitos de seguridad del ensamblado.
Cuando se ejecuta el programa de C#, el ensamblado se carga en el CLR, el cual podría realizar diversas acciones en función de la información en el manifiesto. Luego, si se cumplen los requisitos de seguridad, el CLR realiza la compilación Just in time (JIT) para convertir el código IL en instrucciones máquina nativas. El CLR también proporciona otros servicios relacionados con la recolección de elementos no utilizados, el control de excepciones y la administración de recursos. El código que se ejecuta en el CLR se conoce a veces como çódigo administrado", a diferencia del çódigo no administra-

do"que se compila en lenguaje de máquina nativo destinado a un sistema específico. En el siguiente diagrama se ilustran las relaciones de tiempo de compilación y tiempo de ejecución de archivos de código fuente de C#, las bibliotecas de clases de .NET Framework, los ensamblados y el CLR.
Ninguno de los compiladores que generan código para la plataforma .NET produce código máquina para CPUs x86 ni para ningún otro tipo de CPU concreta, sino que generan código escrito en el lenguaje intermedio conocido como Microsoft Intermediate Lenguage (MSIL) El CLR da a las aplicaciones la sensación de que se están ejecutando sobre una máquina virtual, y precisamente MSIL es el código máquina de esa máquina virtual. Es decir, MSIL es el único código que es capaz de interpretar el CLR, y por tanto cuando se dice que un compilador genera código para la plataforma .NET lo que se está diciendo es que genera MSIL.
MSIL ha sido creado por Microsoft tras consultar a numerosos especialistas en la escritura de compiladores y lenguajes tanto del mundo académico como empresarial. Es un lenguaje de un nivel de abstracción mucho más alto que el de la mayoría de los códigos máquina de las CPUs existentes, e incluye instrucciones que permiten trabajar directamente con objetos (crearlos, destruirlos, inicializarlos, llamar a métodos virtuales, etc.), tablas y excepciones (lanzarlas, capturarlas y tratarlas).

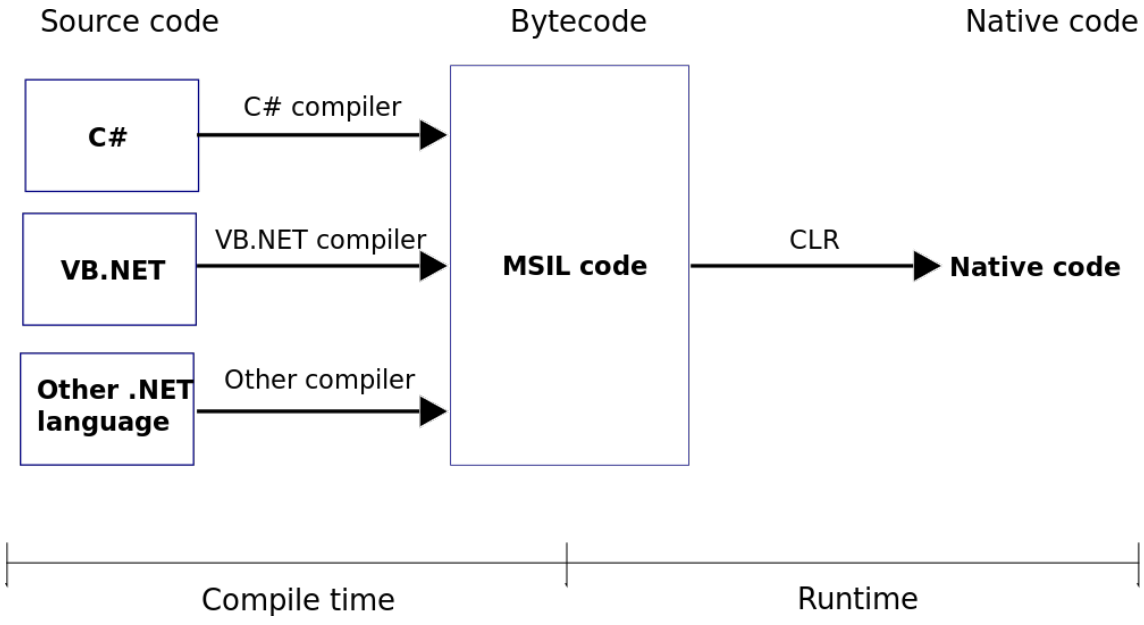

Figura 1.1: Compilación en C#

1.4. Assemblies

Los assemblies son los bloques de construcción de las aplicaciones del framework .NET. Un assembly es una colección de tipos y recursos que están construidos para trabajar en conjunto y formar la unidad lógica de la funcionalidad del programa.
Todos los tipos en .NET Framework deben existir en assemblies; el tiempo de ejecución de idioma común no admite tipos fuera de ensamblados. Cada vez que crea una aplicación de Microsoft Windows, un servicio de Windows, una biblioteca de clases u otra aplicación con Visual Basic .NET, está creando un solo assembly. Cada assembly se almacena como un archivo .exe o .dll.
Aunque es técnicamente posible crear assemblies que abarquen múltiples archivos, no es probable que use esta tecnología en la mayoría de las situaciones.
.NET Framework usa assemblies como la unidad fundamental para varios propósitos:
•Seguridad
•Tipo de identidad
•Versiones
•Desarrollo

Capítulo 2

Introducción a Visual Studio 2017 y C#

2.1. Características de C#

C# es un lenguaje de programación orientado a objetos, al ser posterior a C++ y Java. los lenguajes de programación orientados a objetos más conocidos hasta entonces, C# combina y mejora gran parte de las características más interesantes de ambos lenguajes. Por tanto, un programador que conozca C# a fondo no tendrá problemas para programar tanto en C++ como en Java, sus anteccesores.
El nombre fue inspirado por la notación musical '#' (llamada sostenido, en inglés Sharp) que indica que la nota es de un tono más alto. Se puede utilizar C# para crear aplicaciones cliente de Windows, servicios Web XML, componentes distribuidos, aplicaciones cliente-servidor, aplicaciones de base de datos, y mucho, mucho más.
Para poder crear programas en C# y ejecutarlos posteriormente, es necesario tener instalado en el PC los siguientes paquetes:

.NET Framework SDK: Es el kit de desarrollo e incluye un compilador de línea de C# y bibliotecas que contienen una amplia colección de clases previamente definidas que podemos utilizar en nuestras aplicaciones; es decir, contiene todo lo necesario para poder crear y compilar nuestros programas.
.NET Framework Redistributable Package: Permite la ejecución de programas creados en C#. Esto es necesario porque la compilación de C# no genera, como habitualmente para otros lenguajes, código máquina, sino un código escrito en un lenguaje propio de Microsoft: MSIL (Microsoft Intermediate Language). El CLR (Common Language Runtime) es el núcleo de

la plataforma .NET y se encarga de gestionar la ejecución de los programas escritos en MSIL, ambos se pueden descargar gratuitamente desde la página web de la Red de Desarrolladores para Microsoft (Microsoft Developers Network) :
https://www.microsoft.com/es-mx/download

También pueden crearse programas mediante la herramienta Visual Studio(Incluye en su instalación dichos paquetes), que ofrece un interfaz gráfico muy amigable y cómodo de utilizar, esta herramienta cuenta con una versión de paga y una gratuita (Community) y se puede descargar directamente desde la página oficial.
https://visualstudio.microsoft.com/es/

En la figura 2.2 se muestra un menú de paquetes complementarios que podemos descargar, si sólo queremos desarrollar en C# basta con seleccionar los primeros dos paquetes.

Figura 2.1: Página oficial para descargar Visual Studio

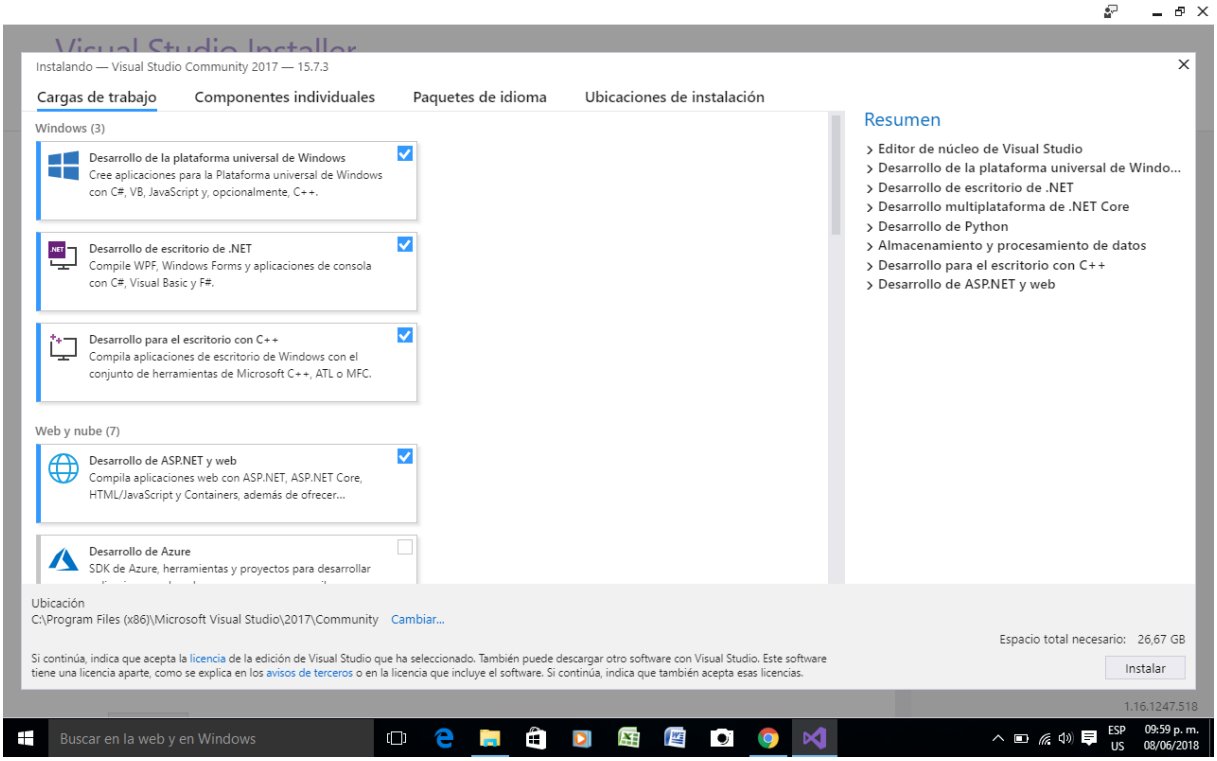
Visual Studio Installer
Instalando — Visual Studio Community 2017 — 15.7.3
Cargas de trabajo
Componentes individuales
Paquetes de idioma
Ubicaciones de instalación
Windows (3)
Desarrollo de la plataforma universal de Windows
Cree aplicaciones para la Plataforma universal de Windows con C#, VB, JavaScript y, opcionalmente, C++.
Desarrollo de escritorio de .NET
Compile WPF, Windows Forms y aplicaciones de consola con C#, Visual Basic y F#.
Desarrollo para el escritorio con C++
Compila aplicaciones de escritorio de Windows con el conjunto de herramientas de Microsoft C++, ATL o MFC.
Web y nube (7)
Desarrollo de ASP.NET y web
Compila aplicaciones web con ASP.NET, ASP.NET Core, HTML/JavaScript y Containers, además de ofrecer...
Desarrollo de Azure
SDK de Azure, herramientas y proyectos para desarrollar
Resumen
Editor de núcleo de Visual Studio
Desarrollo de la plataforma universal de Windo...
Desarrollo de escritorio de .NET
Desarrollo multiplataforma de .NET Core
Desarrollo de Python
Almacenamiento y procesamiento de datos
Desarrollo para el escritorio con C++
Desarrollo de ASP.NET y web
Ubicación
C:\Program Files (x86)\Microsoft Visual Studio\2017\Community
Cambiar...
Si continúa, indica que acepta la licencia de la edición de Visual Studio que ha seleccionado. También puede descargar otro software con Visual Studio. Este software tiene una licencia aparte, como se explica en los avisos de terceros o en la licencia que incluye el software. Si continúa, indica que también acepta esas licencias.
Espacio total necesario: 26,67 GB
Instalar
1.16.1247.518
Buscar en la web y en Windows
ESP
US
09:59 p. m.
08/06/2018

Figura 2.2:

2.2. Estructura de un programa

C# es un lenguaje orientado a objetos. En cualquier programa en C# debe existir al menos una clase que contenga un método llamado **Main**. Este método constituye lo que se denomina punto de entrada, y define por dónde ha de comenzar a ejecutarse la aplicación: la primera instrucción ejecutada será la primera instrucción del método **Main**.
El siguiente ejemplo muestra el programa más simple que puede crearse en C#.

```
1 using System;
2 //Usamos el espacio de nombres System
3 class Ejemplo1 {
4   static void Main() {
5
6   }
7 }
```

Para crear una clase hay que escribir la palabra reservada **class** seguida del nombre que queremos darle. A continuacion entre llaves, aparecerán los métodos y atributos de dicha clase. En este ejemplo hemos creado la clase *Ejemplo1*, que no incluye ningún atributo y contiene un solo método, de nombre **Main**. Como hemos dicho, todo programa en C# debe contener al menos una clase con un método llamado **Main**.

Es importante aclarar que C# se distinguen las mayúsculas de las minúsculas, algo que no curre en todos los lenguajes de programación. Así, el punto de entrada ha de llamarse *Main* y no *main* o *MAIN* o ninguna otra variante.

Como para cualquier otro método existen varias alternativas válidas oara crear la declaración del método **Main**. Algunas son:

public static int Main()
public static void Main(string[] args)
static int Main(string[] args)

La palabra **public** indica que el método es público, es decir, puede ser utilizado por otra clase (si no se pone nada, se considerará privado por defecto). La palabra **static** indica que que el método está asociado a la clase a la que pertebece y no a los objetos que se creen de dicha clase. En tercer lugar, aparece el tipo de la información que devuelve el método: **int** indoca que devuelve un dato de tipo entero; **void** indica que no se devuelve nungún valor.

A continuación del nombre del método, que para el punto de netrada siempre es **Main**, aparecen entre paréntesis los argumentos o datos de entrada de este método, es decir, la información de la partida que requiere. Esta sección puede estra vacía.

Por el momento dado los escasos conocimientos que aún tenemos, será suficiente con que el punto de entrada sea estático, no tenga argumentos y no devuelva ningún valor, tal y como se declaró en el ejemplo.

2.3. Compilación y ejecución

La compilación y ejecución usando Visual Studio resulta ser muy sencilla sin embrago se explicará el proceso con dicha herramienta y sin ella.
Para compilar nos vamos al menú que se encuentra en la parte superior de la ventana de C# y damos click en la pestaña donde dice Compilar, posteriormente se nos abrirá un menú en el cual seleccionaremos la opción de compilar solución. Al hacer dicho paso en la parte inferior aparecerá otra ventana conocida como la ventana de salida o de output donde se nos informará de todos los errores en nuestro programa antes de ejecutarlo. Una vez compilado nuestro programa nos vamos a la ventana superior y damos click en la pestaña donde dice depurar, al abrirse el menú seleccionamos la opción de iniciar sin depurar y se nos abrirá la ventana de comandos con nuestro programa en ejecución.

Capítulo 3

Tipos predefinidos y control de flujo

3.1. Tipos predefinidos de C#

3.2. Tipos de referencia

3.3. Sentencias condicionales

3.4. Ciclos de repetición

Capítulo 4

Clases y Objetos

4.1. Conceptos básicos de POO

4.2. Creación de clases

4.3. Constructores

4.4. Propiedades

4.5. Atributos y métodos de instancia

4.6. Miembros estáticos

4.7. Estructuras

4.8. Tipos de referencia vs Tipos de valor

4.9. Clases estáticas y métodos de acceso

Capítulo 5

Control de Acceso

5.1. Namespaces

5.2. Encapsulamiento y modificadores de acceso

5.3. Métodos accesores vs propiedades

Capítulo 6

Arreglos

6.1. Sintaxis y uso de arreglos

6.2. Arreglos multidimensionales

6.3. Clase Array

Capítulo 7

Objetos y métodos

Capítulo 8

Polimorfismo

8.1. Concepto de polimorfismo

8.2. Interfaces y su implementación

8.3. Relación de subtipos y supertipos

Capítulo 9

Herencia

9.1. Herencia (is-a relationship)

9.2. Métodos virtuales

9.3. Palabras reservadas virtual y override

9.4. Clases abstractas y clases selladas

9.5. Clase Object

Parte II

C# Intermedio

Capítulo 10

Excepciones

10.1. Clase Exception

10.2. Bloque try-catch-finally

10.3. Definición de una excepción

10.4. Relanzar Excepciones

Capítulo 11

Strings

11.1. String vs StringBuilder

11.2. Formato de una cadena

Capítulo 12

Manejo de archivos

12.1. Archivos y flujos

12.2. Manejo de sistema de archivos

12.3. Clases File, FileInfo, Directory, DirectoryInfo

12.4. Lectura y escritura de archivos

Capítulo 13

Genéricos

13.1. Necesidad de tipos genéricos

13.2. Métodos genéricos

13.3. Clases genéricas

Capítulo 14

Colecciones

14.1. Listas y Diccionarios

Capítulo 15

Concurrencia

15.1. Clases Thread y Parallel

15.2. Tasks

15.3. Sincronización

Capítulo 16

Lambdas, Delegados y Eventos

16.1. Expresiones Lambda

16.2. Introducción a delegados y eventos

16.3. Creación y uso de delegados

16.4. Multicast delegate

16.5. Uso de eventos

16.6. Clase EventArg

Capítulo 17

LINQ (Checar Entity Framework)

17.1. Introducción a LINQ

17.2. Query syntax

17.3. Métodos de extensión

17.4. Operaciones estándar de consulta

Parte III

C# Avanzado

Capítulo 18

Interfaces gráficas de usuario con Windows Forms

18.1. Introducción a las GUIs

18.2. Manejo básico de eventos

18.3. Propiedades de los controles y Layouts

Capítulo 19

Controles de Windows Forms

Capítulo 20

Introducción a Programación Asíncrona

20.1. Métodos asíncronos

20.2. Palabras async y await

Capítulo 21

WPF (Windows Presentation Foundation)

21.1. ¿Qué es WPF?

21.2. Diferencias entre WPF y Windows Forms

21.3. Mi primera aplicación con WPF

Capítulo 22

Bases de datos con LINQ

22.1. Introducción a las bases de datos relacionales

22.2. LINQ to Entities y ADO.NET

22.3. Operaciones CRUD

Capítulo 23

Control de versiones con Team Explorer y Git

23.1. Configuración de Git y Team explorer

23.2. Manejo de ramas

23.3. Commit