

C#

Para automatización
electrónica e industrial



Aprenda con ejemplos prácticos a incorporar interfaces electrónicas, conectarse a equipos industriales, agregar multimedia interactiva y controlar los periféricos.

2 capítulos dedicados a la integración de:

ADOBE

FLASH

Diseñe animación
interactiva para sus
pantallas de monitoreo.

MICROCONTROLADORES

PIC

Arme proyectos electrónicos
integrando comunicación
RS232 con C#.

Aaron Castro Bazua

C# para automatización electrónica e industrial

Aaron Castro Bazúa

Editor: Aaron Castro Bazua

ISBN-13 978-607-00-5217-0

Registro obtenido 13 de Diciembre de 2011

Editor independiente: 1era edición

www.multitecnologia.com

aaroncb@multitecnologia.com

Primera edición : Enero de 2012, México

Registro obtenido con fundamento en los artículos 53, fracción IV de la Ley Federal del Derecho de Autor y 86, 88, 89 fracción I, 94 fracción II y 96 de su Reglamento.

Derechos reservados

Esta obra es propiedad intelectual del autor, se encuentra prohibida su venta o reproducción sin consentimiento del autor.

Importante

La información tiene un enfoque completamente didáctico, los ejemplos plantean un conocimiento estrictamente básico como introducción a los lenguajes y plataformas descritas, los posibles errores y omisiones no serán jurídicamente responsabilidad del editor.

Autorización para su venta e impresión en México

Estimado lector:

Agradezco su interés por aprender este interesante lenguaje, durante este proceso trataré de explicar de la manera mas amena como puede mejorar sus aplicaciones con C#.NET y diferentes componentes de hardware, también narraré algunas experiencias de aplicaciones en las que he participado como desarrollador.

Atte.
Ing. Aaron Castro Bazúa

Acerca del libro:

El libro nace de la necesidad de aplicar C#.NET en proyectos de automatización con un enfoque práctico, los ejercicios le permitirán combinar diversas tecnologías y acelerar su aprendizaje.

La presentación de los temas se plantean como una platica personal entre instructor y lector.

Acerca del autor:

Egresado de Ingeniería Electrónica por el Instituto Tecnológico de Sonora en 2003, estudió programación a partir de 1995 con Turbo C y VisualBasic, desde entonces ha desarrollado prototipos y proyectos involucrando hardware electrónico e industrial, módulos multimedia con Flash y bases de datos, actualmente se desempeña como Freelance en el campo de soluciones a la medida integrando C#.NET.

Einstein solía decir...

“Si no puedes explicar algo de forma sencilla, entonces tú no lo entiendes bien”

*A partir de esta frase surge la idea de facilitar el aprendizaje de estas tecnologías,
este libro es el resultado de una labor de 11 meses de trabajo y entrega.*

Dedico esta obra a mi familia

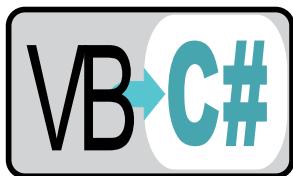
Simbología

Para una lectura amena utilizaremos símbolos dando énfasis a notas especiales.



Tips

Representa algún tip interesante que debemos tener en cuenta.



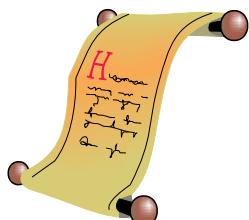
VB (90's) comparado con C#.NET

Significa una comparación entre el antiguo Visualbasic y C#.



En Lenguaje Terrícola

Este símbolo aparece cuando explicamos un concepto con lenguaje sencillo o una analogía.



Anécdotas

En estos párrafos narraremos alguna anécdota relacionada con el tema tratado.

Contenido

Capítulo 1: Introducción a C#

Evolución de los lenguajes y antecedentes.....	2
Origen de .NET.....	3
Conceptos básicos de la plataforma y POO.....	4
Métodos, propiedades e interfaz.....	5
Interfaz de desarrollo IDE	6
Controles estándar y opcionales	6
El editor de código	8
Propiedades y eventos en controles	10
Argumentos en controles	10
Explorador de soluciones	11
Comentarios, nodos y regiones	12
El depurador (debugger)	14
Tipos de dato y sus rangos	15
Tipo Carácter y String	16
Tipo Int, Byte, Bool y Float	17
Operadores aritméticos y lógicos	18
Operadores bits, asignación y creadores de objetos.....	19
Conversión de datos.....	20
Flujo de programa, ciclos y decisiones	22
Declaración de funciones	23
Sentencia If.....	25
Sentencia GoTo.....	26
Operador AND y sentencia If	27
Operador OR y sentencia If.....	29
Práctica 1: "La prueba de Turing".....	30
Sentencia Switch	33
Ciclo For	35
Ciclo For Each y While.....	38
Ciclo Do While	39
Manejo de arreglos en C#.NET.....	40
Apuntadores	40
Práctica 2: Manejo de arreglos en C#.NET.....	41
Excepciones Try, Catch, Finally	42
Errores lógicos.....	45
Práctica 3: Algoritmo de sondeo con sentencia If y ciclo For.....	46
Práctica 4: Cálculo de promedio para 10 calificaciones.....	47
Práctica 5: Conversión de binario a decimal.....	48
Práctica 6: Conversión de decimal a binario.....	49
Práctica 7: VideoJuego "El Ahorcado".....	51
Convención de nombres para variables y controles	54
Ámbito de variables.....	55
Conociendo los controles.....	56
Control Button.....	58
Control Imagelist	59
Control TextBox.....	61
Control Label	62
Control ListBox y CheckBox.....	63
Control NumericUpDown y RadioButton	64
Control GroupBox y PictureBox	65
Control DataGridView.....	66
Control ComboBox y Timer.....	67

Contenido

Control HScrollBar, VScrollBar y ProgressBar.....	69
Control TabControl	70
Control SerialPort.....	70
Control Shockwave Flash.....	71
Control Windows Media Player.....	72
Control Chart.....	73
Propiedades Focus y Tabindex.....	74

Capítulo 2: Programación orientada a eventos

Mensajes y cuadros de dialogo	76
Acceso a controles desde otros formularios.....	78
Práctica 8: Eventos periódicos con control Timer	80
Práctica 9: Gráfica de señal simulada.....	81
Práctica 10: Simulación de cronómetro.....	89
Práctica 11: Manejo de alarmas.....	95
Práctica 12: Simulación de monitoreo domótico.....	105
Práctica 13: Videojuego "Pong" con Flash y C#.NET.....	114
Concepto de "Padre" (parent).....	119
Práctica 14: Eventos compartidos.....	120
Creación de controles por código.....	122
Práctica 15: Arreglos de controles.....	122
Liberación de recursos en C#.NET.....	125
Práctica 16: Control Chart para impresión en papel.....	125
Práctica 17: Manejo básico de usuarios y diseño de User Control.....	131

Capítulo 3: Programación orientada a objetos

Introducción a la Programación Orientada a Objetos.....	140
Origen de la POO.....	141
Enemigos de la POO.....	142
El problema para comprender la POO	143
Conceptos básicos de programación orientada a objetos.....	144
Herencia.....	145
Abstracción, Polimorfismo	146
Encapsulamiento.....	147
Aplicaciones de la programación orientada a objetos.....	147
Aprendiendo a modelar con POO.....	148
Práctica 18: Modelando una compuerta lógica.....	152
Práctica 19: Clase para obtener la ecuación de la recta de un sensor.....	158
Práctica 20: Herencia de eventos y controles de usuario.....	162
Práctica 21: Ecosistema a base de objetos.....	166
Diseño de objetos para tomar muestras de datos.....	176
Práctica 22: Clase genérica para tomar muestras de sensores.....	178
Concepto de ArrayList.....	179
Concepto de Tag.....	181
Patrones de diseño de software.....	187
Práctica 23: Patrones de diseño por capas	189
Anexos: Clases capa negocio.....	198
Anexos: Clases capa de datos.....	202

Contenido

Capítulo 4: Introducción a Adobe Flash

¿Ingenieros haciendo gráficos?.....	206
Antecedentes de Flash.....	207
Flash aplicado a la automatización.....	208
ActionScript y el entorno de desarrollo	209
Herramientas.....	210
Documento de Flash y Menús.....	211
Línea de tiempo.....	212
Las capas (Layers).....	213
Herramientas de edición.....	214
Primera animación.....	218
Concepto de Acción.....	221
Práctica 23: Jugando con la animación.....	232
Práctica 24: Animación de un carro.....	234
Práctica 25: Creando MovieClips.....	239
Práctica 26: Diseño de botones personalizados.....	243
Práctica 27: Máscaras en Flash.....	246
Práctica 28: Comunicación bidireccional con C#.NET.....	253
Diferencias entre ActionScript 2.0 y 3.0.....	258
Práctica 29: Videojuego "Pong" con Flash y C#.NET.....	258
Práctica 30: Manejo de sonido en Flash.....	268
Práctica 31: Diseño del medidor de temperatura.....	273
Práctica 32: Diseño de la casa automatizada (domótica).....	275

Capítulo 5: Aplicaciones móviles y control telefónico

Aplicaciones para dispositivos móviles.....	281
Requisitos de VisualStudio.....	282
Práctica 34: Ejemplo básico en un dispositivo móvil.....	283
Práctica 35: Reproduciendo audio con nuestro móvil.....	286
Práctica 36: Monitoreo y control inalámbrico con SmartPhone.....	291
Código del servidor de alarmas.....	301
Aplicaciones de telefonía con Módem.....	305
Requerimientos para ejercicios.....	306
Conexión del modem PCI sin afectar enlace a Internet y teléfono.....	306
Trabajando con DTMF (Dual-Tone Multi-Frequency).....	307
Aplicaciones de automatización telefónica.....	308
Práctica 37: Reconociendo el módem de la PC	310
Práctica 38: Marcador telefónico básico.....	315
Práctica 39: Contestador telefónico básico.....	320
Práctica 40: Sistema de control por medio de tonos.....	330
Formato de audio para reproducción en módem.....	342
Archivos de audio gratuitos para reproducción por módem.....	344

Contenido

Capítulo 6: Comunicación con PLC's y configuración de servidor OPC

¿Como debe ser un monitoreo industrial?.....	350
La complejidad de los procesos.....	351
La delimitación de objetivos.....	352
Una historia sobre procesos.....	353
El arte de cotizar un proyecto.....	355
Consejos para cotizar un proyecto.....	357
Cotizar en base a la "Ley de Murphy".....	357
Etapas para tener éxito como desarrollador.....	358
Diferencia entre HMI y SCADA.....	358
Introducción al OPC.....	359
Aplicaciones Cliente-Servidor.....	360
Aplicaciones Cliente con C#.....	360
Virus en sistemas de automatización.....	361
Virus enfocado a sistemas SCADA.....	361
Servidor Kepware.....	362
Instalación y configuración de servidor OPC.....	363
Configurando un PLC KOYO DL06 de Automation Direct	365
Configurando un PLC Siemens S7-300.....	373
Práctica 41: Aplicación cliente con C#.NET.....	384
Archivos de audio gratuitos.....	396
Archivos de imágenes vectoriales gratuitos.....	400

Capítulo 7: Lenguaje C para Microcontroladores PIC con el entorno CCS

Introducción a CCS.....	412
Entorno de desarrollo.....	413
Tipos de dato, Comentarios y Operadores.....	415
Tipos de operadores.....	415
Sentencia GoTo	417
Sentencia If y Switch.....	418
Ciclo For, While y Do While.....	419
Directivas para el compilador.....	420
Conversiones entre tipos.....	421
Comandos para entrada y salida.....	422
Práctica 42: Sencillo ejemplo parpadeando leds.....	423
Práctica 43: Corrimiento de un bit en el puerto A.....	429
Práctica 44: Lectura de puerto analógico RE0 y despliegue en puerto A.....	431
Práctica 45: Manejo de un Display LCD	432
Práctica 46: Rotando un mensaje en el Display LCD.....	434
Práctica 47: Desplegando el valor del potenciómetro en el Display LCD.....	436
Práctica 48: Conversión de binario a decimal.....	437
Práctica 49: Generando pulsos con período variable.....	439
Práctica 50: Envío de datos por puerto serie.....	443
Práctica 51: Recepción de datos por puerto serie.....	446
Práctica 52: Comunicación serie bidireccional con C#.NET.....	448
Práctica 53: Manejo de interrupciones en el puerto B.....	450
Práctica 54: Juego para sumar números usando interrupciones.....	453

Contenido

Práctica 55: Frecuencímetro sencillo con el timer1 y una entrada por hardware.....	457
Práctica 56: Manejo de interrupciones combinadas con el puerto serie.....	459
Práctica 57: Control de Led RGB con PWM por software.....	462
Práctica 58: Control de Led RGB desde C#.NET.....	465
Aplicaciones con puertos COM virtuales por USB.....	470
Práctica 59: Comunicación bidireccional por USB.....	470

Capítulo 8: Manejo de periféricos

Introducción al puerto serial.....	479
Práctica 60: Chat por puerto serie entre 2 computadoras.....	479
Comunicación con Microcontroladores PIC.....	484
Práctica 61: Recepción de datos por puerto serie desde el PIC.....	485
Práctica 62: Envío de datos por puerto serie al PIC.....	488
Práctica 63: Comunicación serial bidireccional con PIC 16f887.....	490
Práctica 64: Envío de datos para control de led RGB.....	495
Práctica 65: Comunicación por USB con el PIC.....	498
Práctica 66: Control del puerto paralelo en C#.....	502
Práctica 67: Lectura y escritura básica de puerto paralelo.....	503
Práctica 68: Identificación de datos de lectura en puerto paralelo.....	508
Práctica 69: Monitoreo con puerto paralelo y manejo de multimedia.....	512
Ventajas y desventajas del puerto paralelo.....	521
Práctica 70: Control para semáforos de proceso con puerto paralelo.....	522
Introducción a TCP/IP.....	533
El concepto de Socket.....	533
Práctica 71: Comunicación básica Cliente-Servidor.....	534
Práctica 72: Identificando la IP de mi PC.....	541
Práctica 73: Chat entre dos computadoras por UDP.....	542

IMPORTANTE

VisualStudio, C# y .NET Framework son marcas registradas propiedad de Microsoft Corporation Inc.

Los temas tratados en este curso no afectan los derechos de autor de Microsoft.

¡Compre VisualStudio!

Para proyectos de automatización industrial la plataforma Windows ha comprobado ser la más conveniente, aprenda los conceptos básicos con este libro.

Capítulo 1

Introducción a C#.NET

En este capítulo aprenderemos los conceptos básicos de programación, ejemplos en consola y una introducción a los controles de VisualStudio.

Requisitos: VisualStudio 2008
Windows XP (preferencia) o Windows 7

Evolución de lenguajes y antecedentes de .NET

De Turbo C, Pascal a C#.NET

Si usted aprendió a programar con lenguajes como Turbo C, Pascal o Clipper este libro le ayudará a emigrar a la programación orientada a objetos de una manera sencilla con ejemplos prácticos en el área de control.

De VisualBasic (98) a C#.NET

La sencillez de VisualBasic permitió adaptarlo a los sistemas industriales desde su concepción, fueron buenos tiempos cuando el compilador nos dejaba jugar a programar sin muchas restricciones y poco nos preocupaba la gestión interna del lenguaje, dado que este libro tiene un enfoque ingenieros de control permítame compartirle algo...

En el 95 fue mi primer contacto con Turbo C y debo decir que desde que sentí que podía hacer que la maquina “pensara” no me pude despegar de este mundo tan interesante llamado programación, crear videojuegos fue la mejor manera de ganar entusiasmo, durante estos días teníamos la limitante de ejecutar en ambiente DOS, agregar mouse y gráficos era una tarea laboriosa y casi artesanal, fueron 3 años de aplicar el lenguaje en diferentes proyectos escolares y salvar mi vida gracias al legendario Turbo C... hasta que llegó el día que me presentaron a VisualBasic, al ver la interfaz pensé... ¿y donde está el editor? después al arrastrar controles y configurar sus gráficos sin programar fue un autentico “Shock”.

VisualBasic ¿Cómo puede ser tan sencillo?

Cualquier persona que tuvo la experiencia de pasar de lenguajes por procedimientos a un lenguaje como VisualBasic se hace esta pregunta, crear la interfaz sin tener que programar fue un gran avance en el mundo de la programación y durante el 93 al 2000 fue una época de rediseño de sistemas muy interesante, la moda era emigrar de ambiente DOS a Windows, también tomaron auge otros competidores como Delphi y Java, sin embargo los programadores de automatización optaron por VisualBasic y gran cantidad de sistemas de monitoreo comunicados con PLC's están muy bien implementados con este lenguaje, hago énfasis por que fue muy criticado en ambientes de informática e ingeniería en sistemas, lo consideraban muy limitado y un “lenguaje de juguete” la razón: demasiada sencillez genera malos hábitos de programación.

Los problemas de VisualBasic

Una situación común en el área de sistemas es cuando se tiene que retomar el código fuente de otro programador, como desarrollador me ha pasado en muchas ocasiones, a veces es por que los sistemas no se cotizan bien y el programador entra en conflictos económicos con su cliente, la verdad es complicado cotizar y poner límites a un sistema, implementar una propuesta al pie de la letra en tiempo y costo es un verdadero arte. El problema fue que VisualBasic complicó esta situación, retomar un código de otra persona sin tener documentación o “Los Planos” de un sistema es realmente caótico, al final la mayoría prefería decir al cliente que tenía que reiniciar el sistema con una frase estilo Hollywood... voy a hacer su sistema... “pero a mi manera”.

El Fin de VisualBasic (98)

VisualBasic volvió a nacer con la plataforma .NET, muchas son las razones por las que Microsoft decidió crear su nueva generación de lenguajes y plataforma, personalmente creo que fue una gran herramienta que solo tenía el error de consentir demasiado al programador restando crédito a su eficiencia, los sistemas han evolucionado en gran parte por el auge de Internet, los antiguos diagramas de flujo ya no pueden representar la totalidad de la operación de un sistema, otros estándares como UML (Unified Modeling Language) se han diseñado especialmente para la programación orientada a objetos y las nuevas tendencias, esto fue la razón primordial del gran cambio en Visualbasic y su hermano C#.NET.

Tipos de programación y la transición a objetos

Emigrando de VisualBasic a C#.NET:

Cuando se programa por eventos solo se ponen a trabajar los controles disponibles, en VisualBasic no era común crear nuestros objetos por que la bibliografía de entonces no promovía esta práctica.

¿Puedo programar en C# sin crear clases?

La respuesta es si, pero trataremos de erradicar esa práctica y dejar a un lado los malos hábitos que pudo dejar VB, aún en esta nueva plataforma y sus restricciones todavía podemos crear un caos de código.

Una nueva forma de razonar...

Con C#.NET conoceremos una forma mas natural de pensar en un sistema de control o monitoreo, también cuestiones internas del compilador que no se mostraban en VB y algunos conceptos nuevos como multihilos, herencia, polimorfismo, encapsulamiento, abstracción, etc.

La plataforma .NET

La primera versión de .NET vio la luz entre el 2000 y 2002, durante esta etapa se puso a prueba su versiones Beta y 1.0, con la moda se dio a conocer una nueva palabra “Framework”, en Java esto es similar a la “Maquina Virtual”, sin embargo no solo es un intérprete, permite que gran variedad de lenguajes generen binarios compatibles entre si, de todos ellos el que ha ganado mas popularidad es sin duda C#.NET.

¿De dónde surge C#.NET?

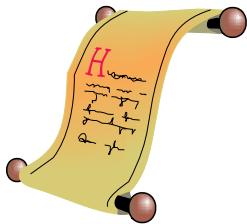
Por el concepto de máquina virtual para algunos tiene mucha similitud con Java, personalmente creo que viene inspirado en Delphi por dos razones:

- 1.- Mismo arquitecto: Anders Hejlsberg
- 2.- Delphi incorporó un IDE para formularios Web y de Escritorio antes de la plataforma .NET (1997).

Cabe destacar que gran cantidad de programadores en Delphi emigraron a C# por estas razones.

C# y el Opensource

El mundo del OpenSource también se sintió atraído por C#, el Mexicano Miguel de Icaza y su proyecto MONO se encuentran actualmente implementando en forma paralela su compilador C# para Linux, otra buena razón para confiar en este lenguaje.



Anécdota:

En una conferencia le preguntaron a Icaza, ¿un programador puede pasar de .NET a Mono?
En ese tiempo Mono no tenía debugger y la dificultad al programar era mayor, su respuesta fue...
"Depende que tan hombre eres".

La Filosofía de .NET

Hasta el momento tenemos claro que C#.NET es una evolución a los entornos de programación y sigue cobrando fuerza, pero... ¿a qué se refiere la plataforma y el tan nombrado framework?.



En lenguaje terrícola...

Las ventajas primordiales de .NET es que se pueden elegir varios lenguajes y combinar los ejecutables que genera cada uno, es decir una parte de un proyecto puede estar hecho en C#.NET, otra en VisualBasic.NET o C++.NET y compartir el código sin problema, esto se debe a que todo se convierte a un lenguaje intermedio (MSIL) que estandariza el código antes de convertirlo en lenguaje máquina.

Acerca del Framework

Lo que en Java se conoce como “máquina virtual” en .NET se convierte en el “Framework”, son simplemente archivos que permiten que se ejecute la plataforma en las PC's con Windows, cada cierto tiempo salen nuevas versiones del Framework (1.1, 2.0, 3.5, 4, etc), es necesario tenerlos instalados con sus respectivos parches (SP1, SP2, etc) para que funcionen nuestros proyectos.

Programación Orientada a Objetos en C#.NET

La popularidad de la programación orientada a objetos se debe a los buenos resultados al momento de construir, rediseñar y dar soporte a un sistema.

Un programador que viene de un lenguaje por procedimientos con Turbo C o Ensamblador debe acostumbrarse a nuevos vocablos y una forma diferente de idealizar el proyecto.

Consideraciones generales si usted está acostumbrado a programar por procedimientos...

- Las librerías reutilizables ahora las llamaremos Clases.
- Las funciones ahora las llamaremos Métodos.
- Las variables que tiene cada Método de una Clase se llaman Propiedades.

¿Qué es una Clase?

Una clase es la definición de las características concretas de un determinado tipo de objetos, es decir, es una especie de molde representado por código que define métodos y propiedades de un concepto.



Clases en lenguaje terrícola...

Una Clase define el comportamiento y descripción de cualquier cosa, un objeto material (carro, martillo, mesa), un concepto (cuenta bancaria, deuda, impuestos) o un ser vivo (humano, perro, delfín).

Métodos de una Clase

Los métodos son las funciones privadas o públicas que dan “vida” a nuestro código, es común que el nombre de estos métodos sea un verbo como “Encender”, “Apagar”, “Encontrar”, etc. Las Clases pueden tener infinidad de métodos en los cuales se debe razonar muy bien su labor y como interactúa con otros métodos y propiedades internos.

Propiedades de una Clase

Las propiedades son todas las variables privadas o públicas con las que realiza sus operaciones cada Clase, por ejemplo para un Objeto “Persona” sus propiedades pueden ser: Edad, Peso, Estatura, etc. En el caso de un proceso industrial donde modelamos el comportamiento de un Sensor las propiedades pueden ser Corriente, Voltaje, Temperatura, etc.

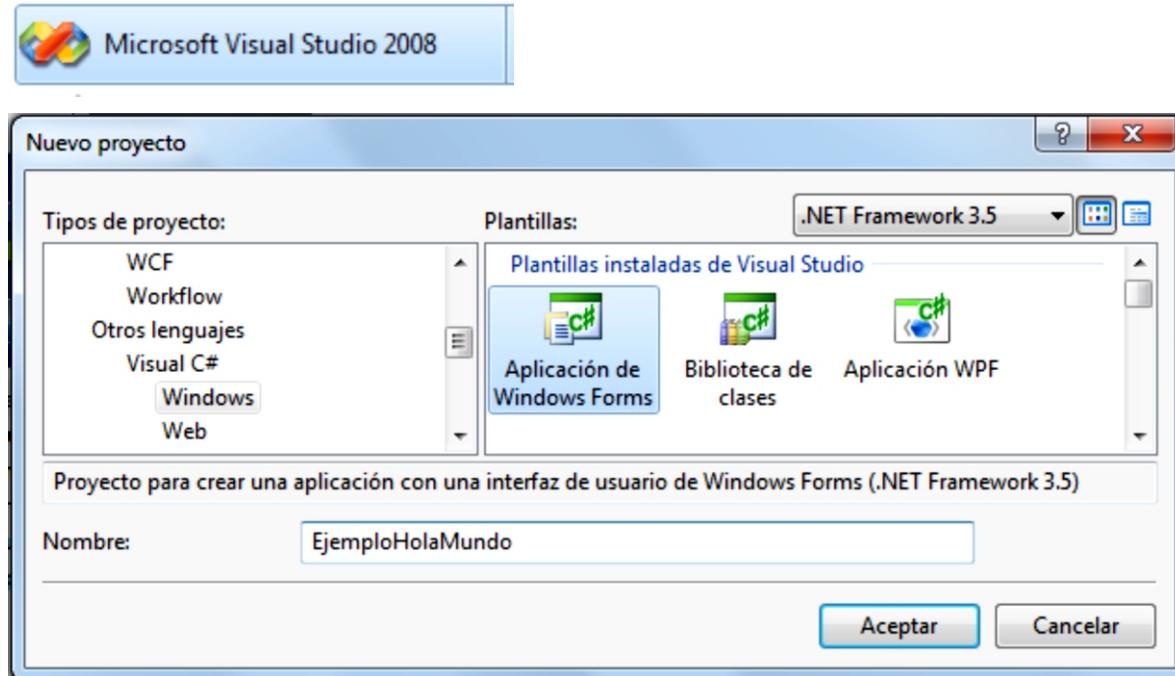
Vamos a centrarnos en el tema de clases cuando dominemos la programación estructurada.

¿Donde obtener C#?

Existen versiones “express” de VisualStudio que son gratis y las podemos obtener por Internet en el portal de Microsoft, para este libro nos basamos en la versión 8 y el Framework 3.0.

¡Iniciemos!

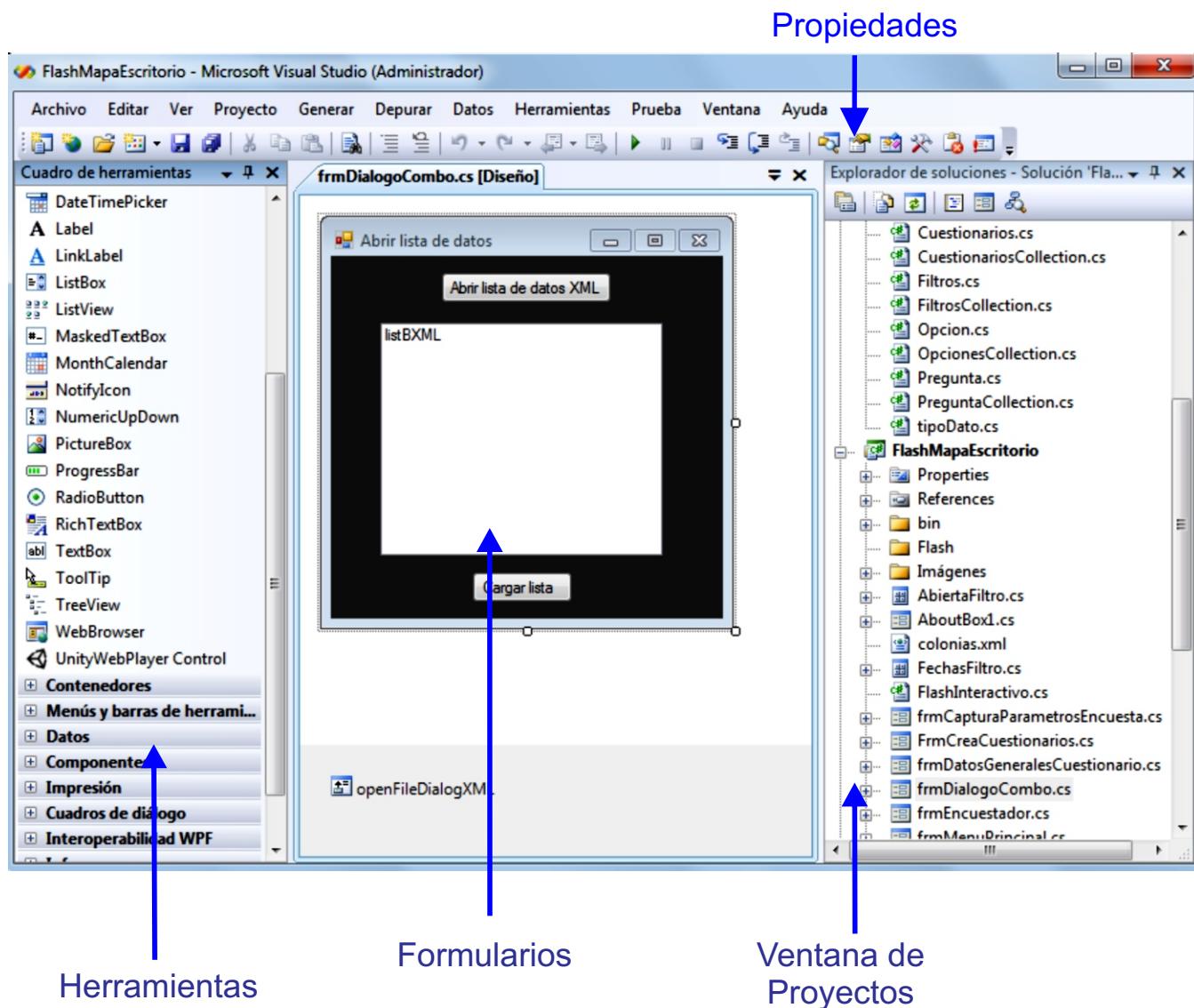
Abrimos VisualStudio y elegimos un proyecto en C# para Windows Forms.



Visualstudio y su entorno de desarrollo integrado (IDE)

El IDE es nuestro entorno de desarrollo para integrar los componentes necesarios de nuestro sistema: formularios, panel de propiedades y proyectos, editor, depurador, acceso a servidores de datos, componentes, clases, etc.

VisualStudio facilita mucho la programación por que proporciona mensajes de ayuda al momento de codificar, depurar y configurar.

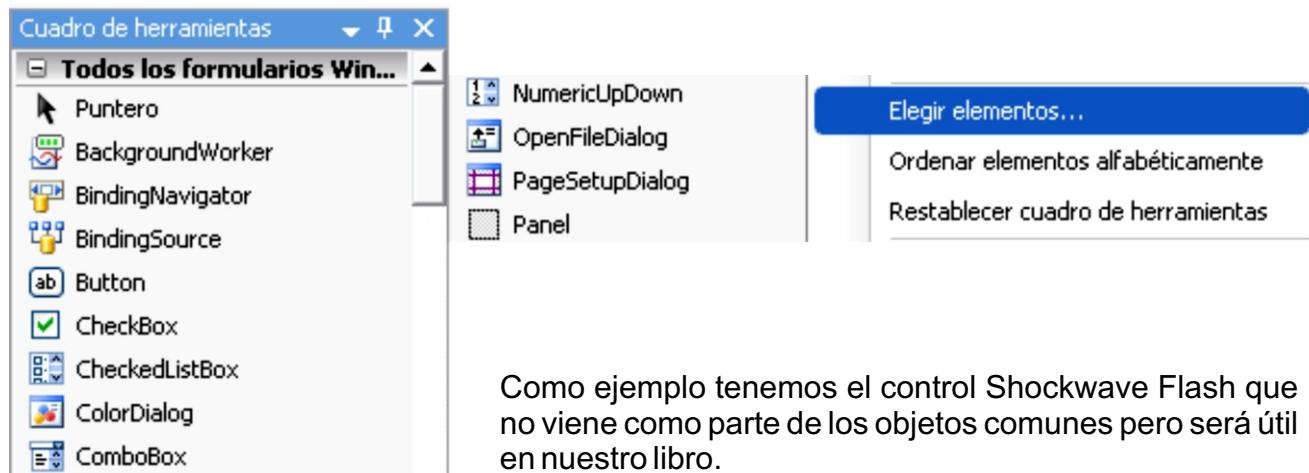


El IDE, es fácil de personalizar, podemos ajustar tamaños y acomodar los paneles, en lo personal recomiendo ajustar la pantalla como se muestra en la imagen.

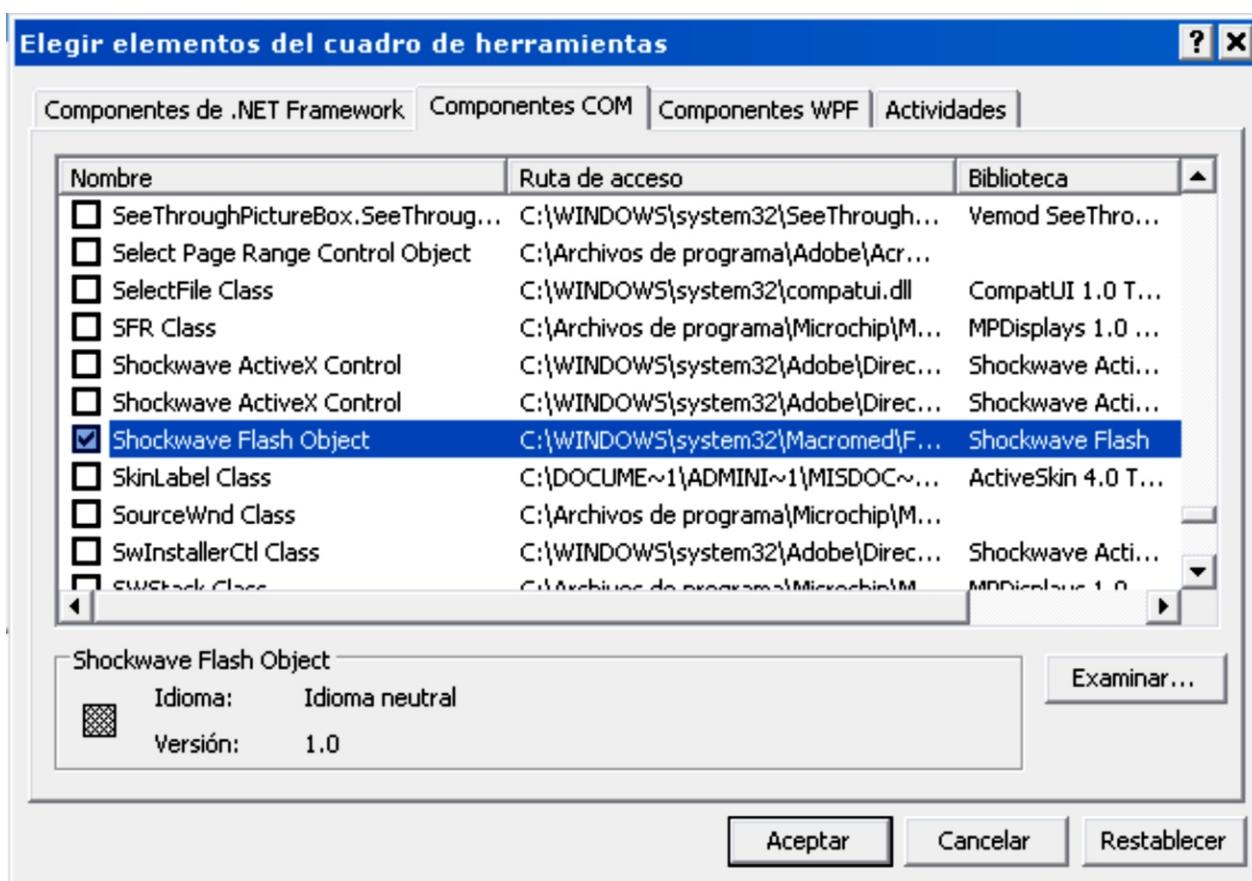
Colocar el panel de proyectos y propiedades al lado derecho es la forma mas sencilla de trabajar.

Barra de herramientas

En la barra de herramientas tenemos todos nuestros controles comunes y aparte podemos agregar otros mas especializados dando clic derecho sobre la barra y la opción “elegir elementos”.



Como ejemplo tenemos el control Shockwave Flash que no viene como parte de los objetos comunes pero será útil en nuestro libro.



Primeros pasos con C#.NET

Antes de empezar a crear nuestras interfaces es importante definir dos conceptos:



Tiempo de diseño: Es cuando nos dedicamos a dibujar la interfaz arrastrando y configurando objetos sin ejecutar el programa.

Tiempo de ejecución: Es cuando compilamos con éxito y nuestro programa esta ejecutándose.

Glosario de palabras comunes

Compilar: Proceso de convertir el texto o código de nuestro programa a un lenguaje máquina.

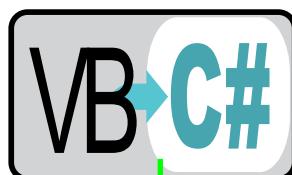
Instancias: Son objetos que derivan de una clase.

Framework: Es un conjunto de ensamblados que permiten que funcione .NET en una PC.

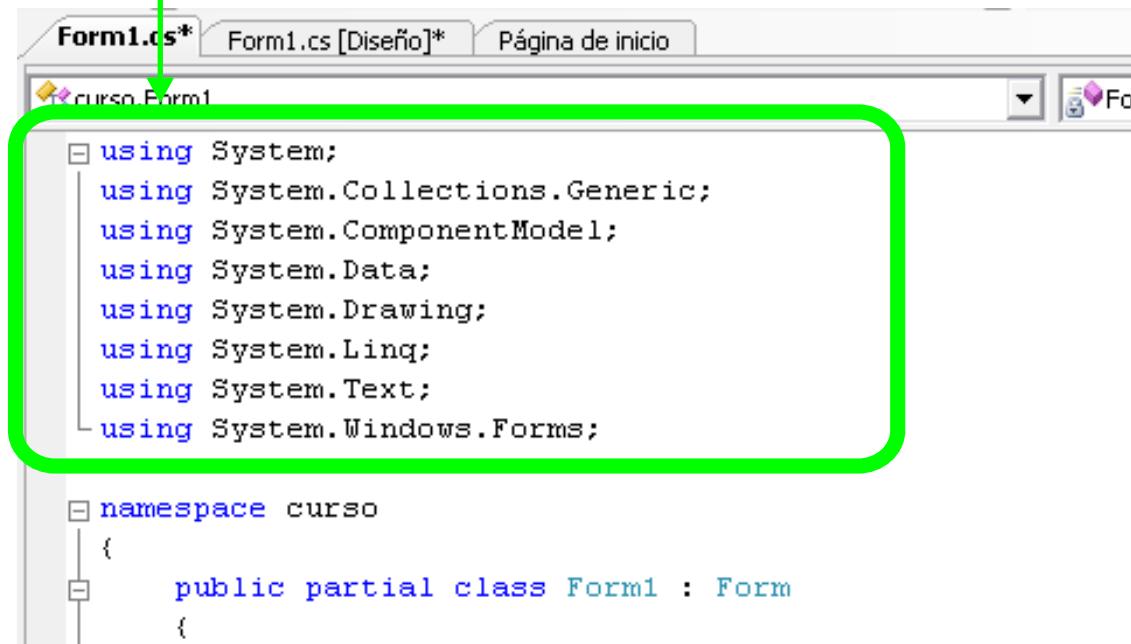
Cast: Es el vocablo utilizado para conversión de datos.

El editor de código

Visualstudio tiene muchas ventajas, es muy cómodo y fácil de organizar, nos ayuda al momento de programar ya que muestra posibilidades a medida que vamos tecleando, tiene otras bondades como mostrarnos las partes del código que genera, acomodar las llaves, crear regiones y organizar mejor nuestro código.



Al momento de aprender a usar VS provoca confusión todo el código "extra" que nos muestra el editor, en los tiempos de VisualBasic (98) esto no se mostraba al usuario, ahora Microsoft decidió abrir parte de su código y dejarnos ver sus interiores.



```

Form1.cs* Form1.cs [Diseño]* Página de inicio
curso.Form1

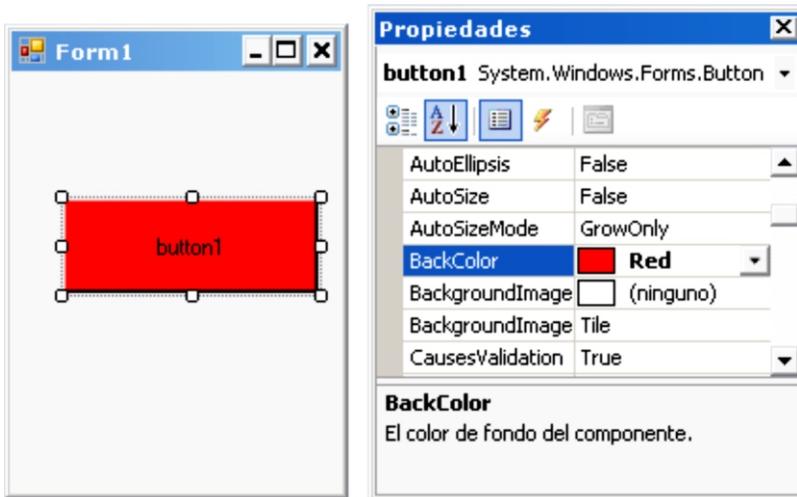
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace curso
{
    public partial class Form1 : Form
    {
}

```

Concepto de Propiedad

Las propiedades son características que tienen todos los controles, por ejemplo un Botón (Button) puede ser color rojo, tener una coordenada x=50, y=100 , estas propiedades nos definen la vista gráfica pero también existen otras para dar comportamiento a nuestro control, por ejemplo:



Enabled.- nos dice si el control esta habilitado para operar, sus valores son (true, false).

Visible.- Permite ocultar el diseño en tiempo de ejecución, sus valores son (true, false).

AutoSize.- Permite que el control se ajuste al tamaño del texto.

Todas estas propiedades se pueden establecer mientras diseñamos la pantalla o bien durante la ejecución.

La sintaxis para leer o establecer propiedades por medio de código en C# es la siguiente:

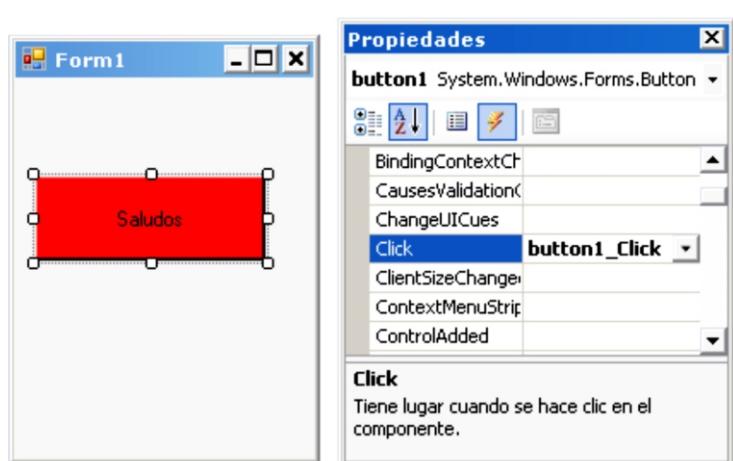
```
button1.BackColor = Color.Blue; //asignamos la propiedad "color de fondo" en azul
button1.AutoSize = true; //elegimos que el control se ajuste al tamaño del texto
button1.Text = "Ver Monitoreo"; //asignamos una cadena de caracteres a la propiedad texto
```

El ícono nos permite elegir las propiedades del objeto, mientras que el nos permite ver los eventos posibles en ese control.

Concepto de Evento

Un evento son las posibles reacciones que tiene un control y permiten ejecutar código, por ejemplo el evento clic de un usuario, arrastrar el mouse sobre un control, etc. Cada control tiene variedad de eventos en común con otros objetos o especialmente diseñados.

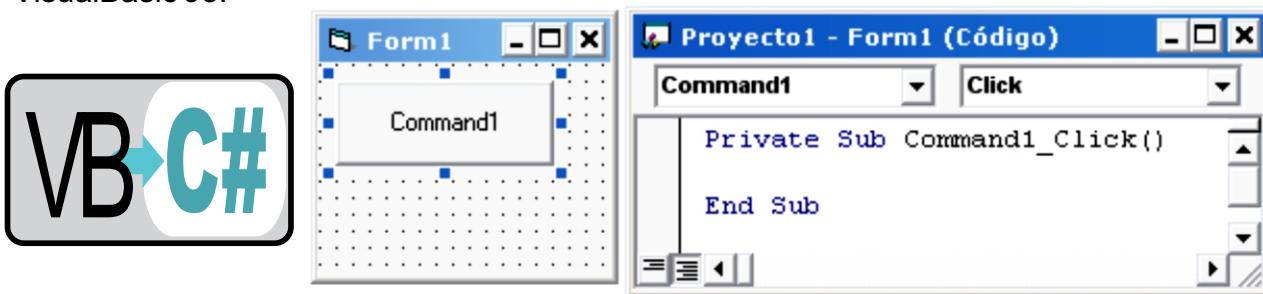
En el caso de un Botón es sencillo acceder a su evento Click, seleccionamos la casilla y hacemos doble Click para entrar al código.



```
private void button1_Click(object sender, EventArgs e)
{
}
```

Como podemos ver VS nos escribe parte del código.

Vamos a hacer un ligero paréntesis comparándolo con la programación orientada a eventos de VisualBasic 98:



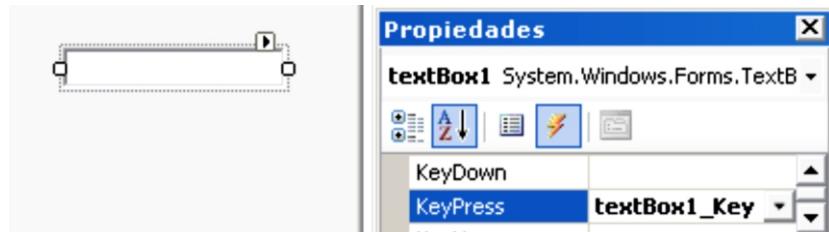
Si usted manejo VB recuerda que los botones de llamaban comandos y el código generado en un evento Click solo mostraba el nombre "Command1" seguido de _Click(), por alguna razón no era necesario conocer como el compilador trabajaba internamente, posiblemente esa fue la clave de la sencillez y éxito de VisualBasic, con C# cambia nuestra manera de ver el código, ahora podemos entender mas a fondo que hace el compilador, para empezar nos muestra dos parámetros extra en el evento Click: (object sender, EventArgs e)



En lenguaje terrícola... ¿para que nos sirven?
object sender: nos permite saber cual objeto esta generando el evento, en este caso el botón.

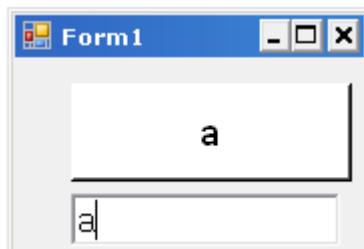
EventArgs e: Nos permite conocer datos que trae el objeto que nos provocó en evento.

Por ejemplo en un textBox los argumentos que manda el evento keypress son los caracteres:



Para probar estos argumentos vamos a utilizar el botón previamente dibujado y en el evento KeyPress del textBox1 escribimos:

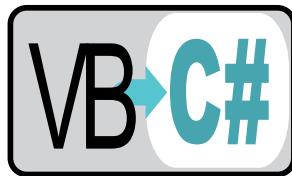
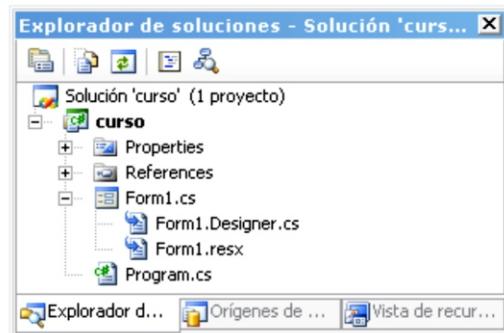
```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    button1.Text = e.KeyChar.ToString();
    //por ser un evento KeyPress cada vez que se escribe un carácter
    //recibimos un argumento "e" por parte del sender(textBox1), el
    //argumento tiene la propiedad KeyChar la cual convertimos en cadena
    //y la asignamos a la propiedad text del botón.
}
```



Ejecutando nuestro programa con el botón ➤ observamos como pasamos el valor de un control a otro.

El IDE también genera líneas en la clase del diseñador Form1.Designer.cs, el código generado corresponde a la configuración de propiedades del botón y el evento que tenemos registrado (Click), ahora VisualStudio nos permite ver y modificar su código interno.

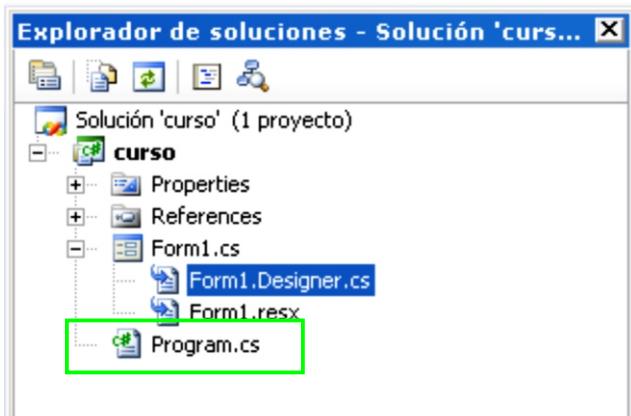
```
// button1
//
this.button1.BackColor = System.Drawing.Color.Red;
this.button1.Location = new System.Drawing.Point(12, 57);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(134, 49);
this.button1.TabIndex = 0;
this.button1.Text = "Saludos";
this.button1.UseVisualStyleBackColor = false;
this.button1.Click += new System.EventHandler(this.button1_Click);
```



En Visualbasic el código del diseñador era completamente oculto al usuario.

Explorador de Soluciones

El explorador de soluciones nos permite acceder a todos nuestros formularios, clases y variedad de elementos para nuestro sistema.



Para este ejemplo solo contamos con un formulario (Form1.cs) pero se pueden tener mayor cantidad, inclusive crear una solución con varios proyectos que a su vez tengan sus propios formularios.

La carpeta de Referencias contiene las librerías que ocupa nuestro programa, en adelante vamos a poner énfasis en esta carpeta ya que incorporaremos controles como Flash y componentes para conexión con PLC's.



El Program.cs es como la chispa de encendido de cualquier programa en C#, crea la primera instancia de nuestro formulario de inicio, a su vez se invoca el método que dibuja los gráficos iniciales de nuestro programa, es decir ejecuta las líneas de Form1.Designer.cs.

Sobre el Editor

El editor de VisualStudio posee muchas ventajas para estructurar un programa, le presento 3:

Comentarios en C#: Los comentarios es una buena práctica al desarrollar, nos permite explicar el funcionamiento de alguna instrucción o función de nuestro programa, en VisualStudio tienen un color verde y para agregarlos tecleamos dos diagonales “//”, con esto VS pone en color verde el texto y lo deja sin validez para la lógica del compilador.

Si deseamos agrupar varias líneas de comentarios agregamos /* al inicio y */ al final, ejemplo:

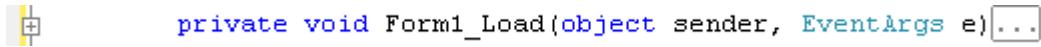
```
// comentarios de una sola linea

/* comentarios
de varias
lineas */
```

Uso de nodos:

Los nodos son otra característica de VS para hacer mas sencilla la lectura del código, cada vez que se agrupa un evento o función por medio de las llaves "{}" se genera un nodo con el símbolo , si presionamos el código encerrado se reduce a una sola línea, con esto es más sencillo encontrar porciones del programa en las que estamos trabajando y dejar en una sola línea las que ya están validadas.

Ejemplo para un evento Load:

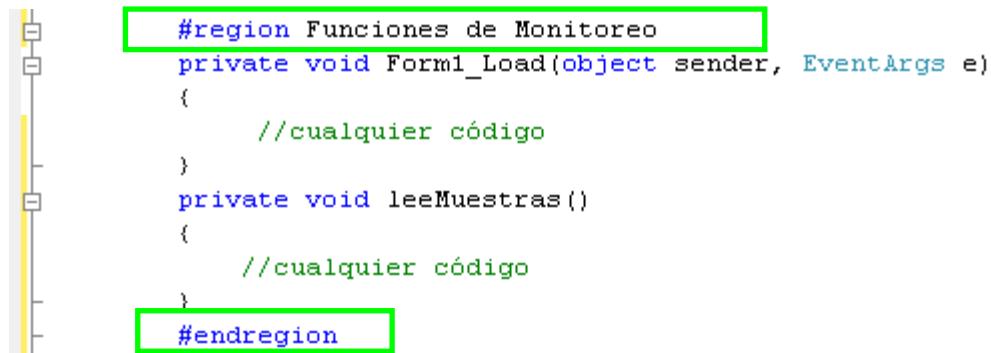


```
private void Form1_Load(object sender, EventArgs e){...}
```

Uso de Regiones:

A diferencia de los nodos, las regiones las define el usuario y pueden agrupar secciones de nodos, las sintaxis es:

```
#region nuestro texto
#endregion
```



```
#region Funciones de Monitoreo
private void Form1_Load(object sender, EventArgs e)
{
    //cualquier código
}
private void leeMuestras()
{
    //cualquier código
}
#endregion
```

Al presionar el nodo  nos queda

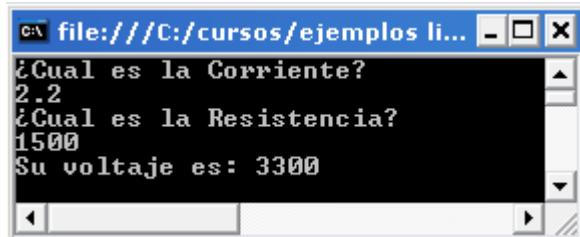


Como podemos observar toda la región queda acotada a una sola línea y conserva el texto que definimos: "Funciones de Monitoreo".

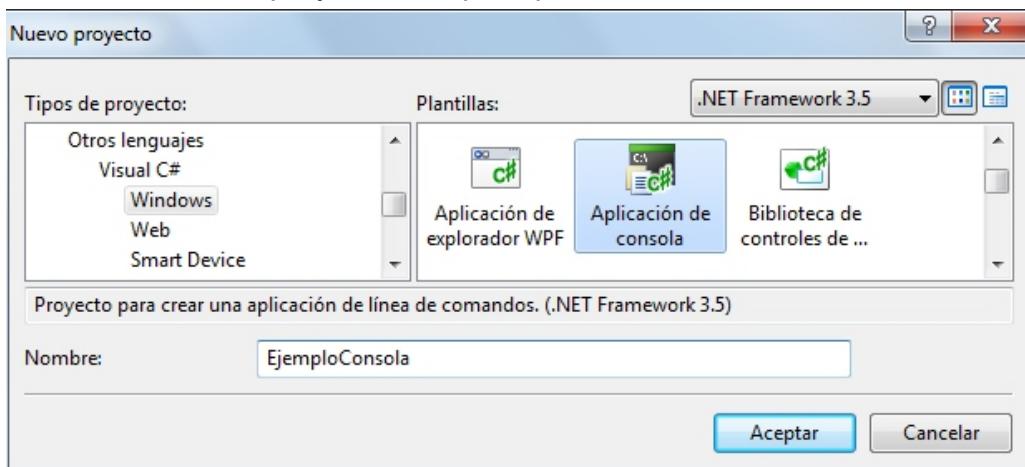
El uso de regiones es otra buena práctica para organizar nuestro programa, corregir errores y agregar funciones.

Iniciando con la consola de C#.NET

Vamos a pausar la explicación de los formularios de windows para centrarnos en los tipos de datos, sentencias y ciclos al estilo antiguo, con la consola de C#.NET.



Creamos un nuevo proyecto del tipo “Aplicación de Consola”.



```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

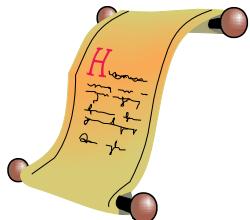
namespace EjemploConsola
{
    class Program
    {
        static void Main(string[] args)
    }
}

```

Trabajar con la consola de C# es parecido al ambiente MS-DOS antiguo, como se puede observar el proyecto solo se compone de las propiedades, referencias y la clase básica: Program.cs

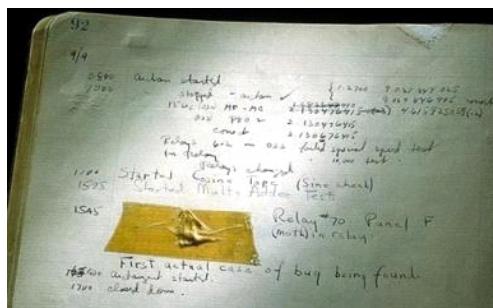
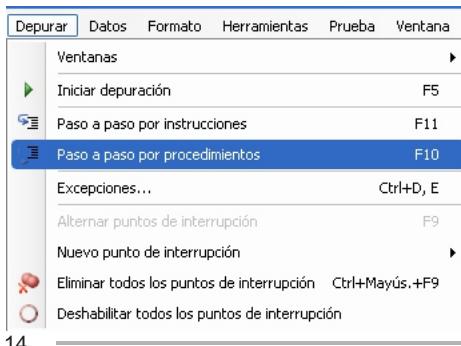
Dejamos por un momento este proyecto, ahora vamos a una parte interesante en todos los compiladores: El Depurador.

El Depurador (debugger)



En 1947 Grace Hooper se encontraba trabajando en la "supercomputadora" electromecánica Mark II, tal día al estar operando se registró un error que impedía el avance de los comandos, los operadores buscaron entre los "pasillos" de la máquina y alguien dijo ¡Es un bicho!, refiriéndose a una palomilla atrapada en un relevador, desde entonces se conocen como "bugs" a los errores que impiden que se ejecute un programa correctamente y el nombre "debugger" a la herramienta para encontrarlos.

En VisualStudio el Depurador se encuentra en el menú principal, es una herramienta avanzada que nos permite localizar errores y ejecutar los programas paso por paso.



El “Bicho” quedó almacenado para la Historia en el libro de apuntes del sistema.

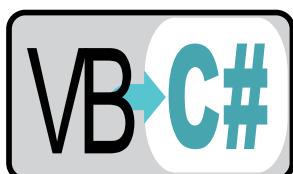
Fuente: U.S. Naval Historical Center Online Library
Photograph NH 96566-KN

Tipos de datos y prácticas con el depurador

En este capítulo estudiaremos conceptos básicos del lenguaje concentrados en sus tipos de datos, operadores, conversión y consejos sobre la mejor manera de asignar un nombre a los controles. Es muy importante tener bien claro estos conceptos por que sentarán las bases del lenguaje, en C# todas las variables son objetos por lo cual analizaremos métodos implícitos que nos ahorrarán tiempo de codificación.

Me interesa que a usted le quede claro como se va ejecutando cada línea de código, para esto vamos aplicar el depurador (debugger) de VisualStudio.

Detalles a considerar sobre la codificación en C#:



Diferencias con VB (98)

El lenguaje si distingue entre mayúsculas y minúsculas

En C# no existe el tipo Variant

El tipo Float es el equivalente a Single o Double en VisualBasic

Tipos de dato en C#

El tipo de dato depende de la información que vamos a guardar, es importante una buena elección para no desperdiciar memoria y hacer lento el sistema.

En esta tabla se presentan los tipos de C# y sus rangos posibles de valor.

Tipo	Rango
bool	true/false
sbyte	-128 a 127
short	-32768 a 32767
int	-2147483648 a 2147483647
long	-9223372036854775808 a 9223372036854775807
byte	0 a 255
ushort	0 a 65535
uint	0 a 4294967295
ulong	0 a 18446744073709551615
float	Aprox. $\pm 1.5 \times 10^{-45}$ a $\pm 3.4 \times 10^{38}$ con 7 decimales
double	Aprox. $\pm 5.0 \times 10^{-324}$ a $\pm 1.7 \times 10^{308}$ con 15 o 16 decimales
decimal	Aprox. $\pm 1.0 \times 10^{-28}$ a $\pm 7.9 \times 10^{28}$ con 28 o 29 decimales
char	Cualquier carácter unicote, ocupa 16 bit

¿Qué tipo de dato elegir?

Si usted va a guardar información de tipo texto el indicado es el `string`, observe los ejemplos...

La sintaxis básica es:

Tipo de dato Nombre de variable

```
//Algunos ejemplos de declaración de variables con un dato inicial
string nombre = "Verónica Michel";
char letra = 'A';
int metros = 100;
byte edad = 5;
float temperatura = 7.2f;
bool autorización = true;
```

Tipo de dato Char (Carácter)

Nos permite almacenar un carácter “Unicode”, por lo regular guardamos letras. Ejemplo:

```
Char letra= 'A';
```

¿Qué es Unicode?

Es una tabla que asigna un código a cada uno de los más de cincuenta mil símbolos para alfabetos europeos, ideogramas chinos, japoneses, coreanos y muchas otras formas de escritura, aparte tiene mas de mil símbolos especiales.

Tipo de dato String (Cadena)

Los tipos String nos permiten almacenar cadenas de caracteres donde el tamaño depende de la cantidad que ocupe, la sintaxis de declaración es:

```
String NuestroDatos;
```

Algo interesante con C# es que todos los tipos de dato tienen algún método que nos puede ser útil, para el tipo String tenemos el substring() que nos permite reemplazar datos de una cadena, ejemplo: En la clase program de nuestro proyecto de consola escriba el siguiente código:

```
class Program
{
    static void Main(string[] args)
    {
        string texto = "Es muy sencillo programar en C# utilizando VisualStudio";
        //con el método "replace" aplicado al objeto dato "texto" lo modificamos
        string nuevotexto = texto.Replace("VisualStudio", "VS");
    }
}
```

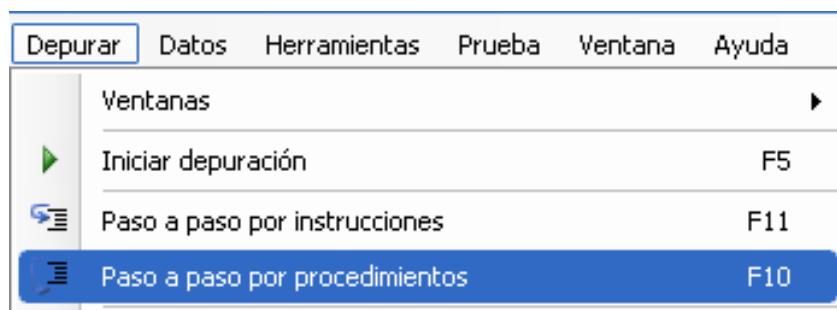
Vamos a practicar con el Depurador.

Agregue un punto de interrupción en el costado izquierdo dando Click y en el menú principal elija Depurar/ Paso a paso por procedimientos o bien F10.



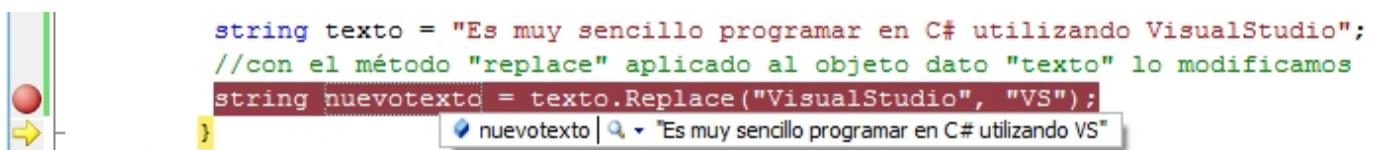
```
string nuevotexto= texto.Replace("VisualStudio", "VS");
```

La depuración nos permite ver “en cámara lenta” como funciona nuestro programa.



Recuerda en C# todos los datos son objetos, por lo tanto tienen sus métodos y nos permiten ahorrar mucho tiempo.

Cuando se ejecuta la línea del punto de interrupción posicionamos el Mouse en la variable “nuevotexto” y podemos ver como se reemplaza la palabra “VisualStudio” por “VS”.



Tipo de Dato Int (Entero)

Los datos enteros nos sirven para almacenar números entre -2147483648 y 2147483647, por lo cual son muy utilizados para información que no tenga puntos decimales, comúnmente se aplica para guardar el índice en una base de datos, almacenar números de pedidos y cualquier valor que no implique decimales.

Ejemplo: `int contador=32; //variable para almacenar un conteo`



Detalle importante:

Cuando dividimos un dato entero entre un entero siempre nos regresa un entero, esto puede generar errores difíciles de detectar como lo muestra el ejemplo:

```
int a = 7;
int b = 10;
float respuesta = b / a;
// error lógico, la respuesta es 1.42 pero se convierte en 1 por
// dividir un entero entre un entero.
```

Tipo de Dato Byte

Los datos tipo Byte tiene un rango de valor entre 0 y 255 con números enteros positivos, es recomendable guardar valores que no excedan ese rango, ejemplos:

```
byte edad = 5; //Edad de una persona
byte mes = 2; //Mes del año
```

Por el enfoque electrónico de este curso la variable tipo Byte la utilizaremos bastante.

Tipo de Dato Bool

Cuando ocupamos almacenar un valor que solo tiene dos opciones: true/false el tipo Bool es el mas indicado, la sintaxis es la siguiente:

```
bool autorizacion = true;
bool secuenciaActiva = false;
```

Tipo de Dato Float (Punto flotante)

Las variables tipo Float tienen un rango entre 1.5*10-45 a 3.4*1038 con 7 decimales, su utilidad se da con cualquier valor que implique decimales. La sintaxis es la siguiente:

```
float temperatura = 7.22f;
float porcentaje = 0.25f;
float ganancia = 1.756f;
```

Constantes en C#

Cuando ocupamos declarar una constante escribimos const antes de la declaración del tipo de dato:

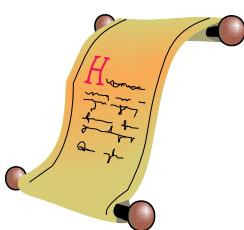
```
const float pi = 3.1415926f; //valor de π
const float f = 96484.5561f; //constante de faraday
```

El bloque **finally** siempre se ejecuta no importando el error, es opcional utilizarlo, debe incluirse después de todos los bloques **catch**. Ejemplo:

```
try //operación riesgosa
{
    int a = 0;
    int respuesta = 10 / a;
}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
finally
{
    Console.WriteLine("siempre se ejecuta esta linea");
}
```

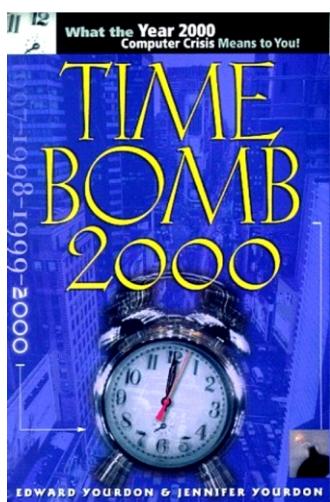
Errores lógicos

El error lógico es básicamente un error humano, se caracteriza por ser indetectable para el depurador y complicado de visualizar por el programador, este tipo de errores se presentan ante condiciones no previstas en el análisis del sistema, son detalles que por lo regular son arreglados hasta el momento que surgen durante el arranque o producción de un sistema.

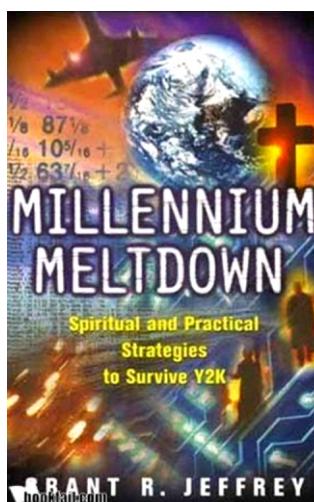


Un poco de historia: el error lógico más famoso...

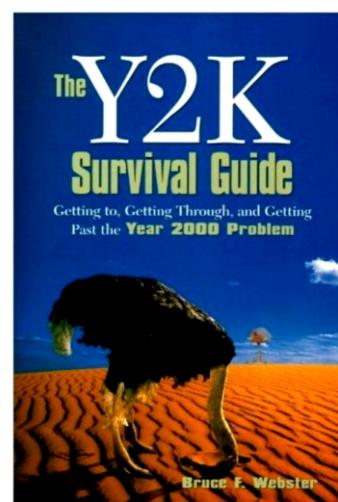
Antes del año 2000 se hizo popular la frase Y2K (Year 2000), este término pronosticaba el error que sufrirían sistemas antiguos por almacenar la fecha del año con solo dos dígitos, ejemplo: 1999 = 99, la gran incógnita era... **¿que pasaría con estos sistemas a la hora de almacenar el año 2000?**, durante los 90's se manejaron varias teorías sobre el posible fin del mundo por la activación de viejas bombas nucleares, colapso financiero, etc. Afortunadamente los efectos negativos fueron mínimos por la gran difusión que se dio al problema.



Autor: Jennifer Yourdon
Prentice Hall
(Diciembre, 1997)



Autor: Grant R. Jeffrey
Tyndale House Pub
(Diciembre, 1998)



Autor: Bruce F. Webster
Prentice Hall
(Diciembre, 1998)

EJERCICIOS CON LA CONSOLA DE C#.NET

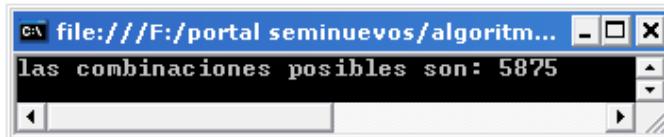
Práctica 3: Algoritmo de sondeo combinando sentencia IF y ciclo For “El número de la suerte: 21”

Fuente: www.multitecnologia.com/practicas/practica3.rar

Hace algunos siglos durante mi época de estudiante recuerdo que cada vez que me daban un boleto de autobus sumaba cada dígito para ver si coincidía con 21, tiempo después vino la interrogante sobre cuantas posibilidades podría tener de acertar en un tiraje de 100,000 boletos diarios, en aquel entonces Turbo C me ayudo a resolverlo, hoy recordaremos aquel viejo algoritmo con C#.NET.

```
//Este programa calcula combinaciones entre 0 y 100,000 que suman el numero 21,
//solo sumamos por separado cada dígito. Ejemplo: 399 --> 3 + 9 + 9 = 21
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace algoritmo
{
    class Program
    {
        static void Main(string[] args)
        {
            funcionCombinaciones(); //mandamos llamar a la función
        }
        private static void funcionCombinaciones()
        {
            int i;
            int numCombinaciones = 0;
            for (i = 0; i <= 100000; i++)
            {
                //lo convertimos en string para sacar el número de letras
                string numLetras = i.ToString();
                int digito = 0, j, suma = 0;
                //este ciclo separa y suma cada dígito
                for (j = 0; j < numLetras.Length; j++)
                {
                    digito = Convert.ToInt32(numLetras.Substring(j, 1));
                    suma = suma + digito;
                }
                if (suma == 21)//cuando suman 21 se incrementan las combinaciones
                {
                    numCombinaciones++;
                }
            }
            Console.WriteLine("las combinaciones posibles son: " +
                numCombinaciones.ToString());
            Console.ReadLine(); //detiene la ejecución esperando un carácter
        }
    }
}
```

Al ejecutar obtenemos:



Según el resultado de cada 100,000 boletos solamente 5,875 forman el número 21, lo que equivale a un 5.875%

Práctica 4: Cálculo de promedio para 10 calificaciones

Fuente: www.multitecnologia.com/practicas/practica4.rar

En este ejercicio practicamos el uso de ciclos para repetir la captura de 10 calificaciones y calcular el promedio al final del ciclo, se utilizan tipos de dato flotantes para dar un resultado con decimales.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace PromedioCalificaciones
{
    class Program
    {
        static void Main(string[] args)
        {
            funcionInicio();
        }
        private static void funcionInicio()
        {
            float calificacion; // variable para almacenar calificación
            float suma = 0.0f; // variable para sumar el acumulado de las calificaciones
            float promedio = 0.0f; // variable el promedio de las 10 calificaciones
            Console.WriteLine("Programa para calcular el promedio de 10 calificaciones");
            //preguntamos 10 veces la calificación y la almacenamos en el arreglo
            for (int j = 0; j < 10; j++)
            {
                Console.WriteLine("Escriba su calificación en el examen #" + (j + 1).ToString());
                //guardamos las calificaciones en el arreglo
                calificacion = (float)Convert.ToDouble(Console.ReadLine());
                suma = suma + calificacion;//vamos sumando cada valor
            }
            promedio = suma / 10; //la suma de todas las calificaciones entre 10
            Console.WriteLine("Su promedio final es: " + promedio.ToString());
            Console.ReadLine();
        }
    }
}
```

Al ejecutar obtenemos:

```
Programa para calcular el promedio de 10 calificaciones
Escriba su calificación en el examen #1
8.2
Escriba su calificación en el examen #2
5.3
Escriba su calificación en el examen #3
9.3
Escriba su calificación en el examen #4
8.6
Escriba su calificación en el examen #5
7.4
Escriba su calificación en el examen #6
5.8
Escriba su calificación en el examen #7
6.6
Escriba su calificación en el examen #8
8.7
Escriba su calificación en el examen #9
7.3
Escriba su calificación en el examen #10
9.8
Su promedio final es: 7.7
```

Ahora el código:

Como vamos a mandar un correo desde nuestra aplicación ocupamos algunas librerías de .NET, en la parte superior donde se encuentran las directivas “using”, agregamos el código que se muestra y la variable string ruta.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Mail; ←
using System.Net.Mime;
using System.Collections;
namespace MonitoreoTanque
{
    public partial class frmTanque : Form
    {//grabamos la ruta donde se ejecuta el programa
        string ruta = Application.StartupPath;

    public frmTanque()
    {
        InitializeComponent();
    }
}
```

Importante:

Agregar estas referencias para tener acceso a recursos que nos permiten enviar correos.

Dando clic en el formulario nos vamos a su evento Load y escribimos:

```
private void frmTanque_Load(object sender, EventArgs e)
{
    //a la ruta le cortamos la carpeta bin\\Debug reemplazandola con nada
    ruta = ruta.Replace("bin\\Debug", "");
    //cargamos el archivo de Flash colocado en la carpeta Multimedia
    flashTanque.Movie = ruta + "Multimedia\\tanque.swf";
}
```

En el evento click del btnReinicio:

```
private void btnReinicio_Click(object sender, EventArgs e)
{
    //regresa todo a los valores iniciales
    gbAlarms.BackColor = Color.DarkRed;
    timerParpadeo.Enabled = false;
    trackBarNivel.Value = 51;
    flashTanque.FrameNum = 50;
    lblNivel.Text = (trackBarNivel.Value - 1).ToString();
    lblMensaje.BackColor = Color.DarkRed;
    lblMensaje.Text = "Mensaje";
    WMPAlarma.URL = null; //detiene la reproducción
}
```

Para el evento Tick del timerParpadeo:

```
private void timerParpadeo_Tick(object sender, EventArgs e)
{
    //permite que el groupbox parpadee cada segundo con dos colores
    if (gbAlarmas.BackColor == Color.Red)
    {
        gbAlarmas.BackColor = Color.Black;
        lblMensaje.BackColor = Color.Black;
    }
    else
    {
        gbAlarmas.BackColor = Color.Red;
        lblMensaje.BackColor = Color.Red;
    }
}
```

En el evento Scroll del trackBarNivel:

```
private void trackBarNivel_Scroll(object sender, EventArgs e)
{
    //al trackBarNivel.Value le restamos uno por que
    // la animación tiene 101 fotogramas,
    //el primer fotograma indica 0% (nivel vacío)
    // el fotograma 101 indica 100% (lleno)
    lblNivel.Text = "Nivel: " + (trackBarNivel.Value - 1).ToString();
    flashTanque.FrameNum = trackBarNivel.Value - 1;
    ActivaAlarmas();
}
```

Creamos una función tipo Void con el nombre “ActivaAlamas”.

```
private void ActivaAlarmas()
{
    if ((trackBarNivel.Value - 1) == 0) //si el tanque se queda sin agua
    {
        lblNivel.Text = "TANQUE VACIO!!";
        timerParpadeo.Enabled = true; //activa el parpadeo de luces en pantalla
        if (rbOpcion1.Checked) //opción que solo activa luces de alarma
        {
            lblMensaje.Text = "acabo de activar las luces de alarma";
        }
        //si opción 2 o 3 esta seleccionada se activa la alarma auditiva
        if ((rbOpcion2.Checked) || (rbOpcion3.Checked))
        {//concatenamos la carpeta Multimedia y asignamos el audio al reproductor
            WMPAlarma.URL = ruta + "\\Multimedia\\AlarmaSirenaNuclear.mp3";
            lblMensaje.Text = "acabo de activar las luces y sirena de alarma";
        }
        //si la opción 3 esta seleccionada manda también el correo
        if (rbOpcion3.Checked)
        {
            lblMensaje.Text = "active todas las alarmas";
            MandarCorreo();
        }
    }
}
```

Finalmente escribimos otra función tipo Void con el nombre “MandarCorreo”.

Cabe mencionar que esta rutina funciona para cuentas de Hotmail, pero cambiando el código puede trabajar con otros servidores de correo.

```
//I M P O R T A N T E

//funciona para cuentas de Hotmail

private void MandarCorreo()
{ //cuando se llama a esta función se manda un correo
    //Se prepara un nuevo mensaje
    System.Net.Mail.MailMessage msg = new System.Net.Mail.MailMessage();

    try
    {
        msg.To.Add("CorreoDelDestino@x.com"); //el destinatario
        msg.DeliveryNotificationOptions = DeliveryNotificationOptions.OnSuccess;
        msg.Priority = MailPriority.High;
        //Se establece el remitente
        msg.From = new MailAddress("suCorreo@hotmail.com",
                                   "suNombre", System.Text.Encoding.UTF8);
        msg.Subject = "Alarma"; //Se establece el asunto del mail
        //Formato de codificación del Asunto
        msg.SubjectEncoding = System.Text.Encoding.UTF8;
        //establece el cuerpo del mail
        msg.Body = "Informandole que a las " + System.DateTime.Now.ToString() +
                   " se dio una alarma en el tanque";
        //Se establece la codificación del Cuerpo
        msg.BodyEncoding = System.Text.Encoding.UTF8;
        //Se indica si al cuerpo del mail, se interpretara como código HTML
        msg.IsBodyHtml = false;
        SmtpClient client = new SmtpClient(); //creamos un objeto cliente
        client.Host = "smtp.live.com"; //apuntamos al servidor de hotmail
        client.Port = 587;
        client.UseDefaultCredentials = false;
        client.EnableSsl = true;
        //Credenciales para autenticarse con la cuenta de correo y la contraseña
        client.Credentials = new System.Net.NetworkCredential("sucorreo@hotmail.com",
                                                               "contraseña");
        client.Send(msg);
        MessageBox.Show("Ya mande el correo!!");
    }

    //en caso de no mandar el correo capturamos los errores
    catch (System.Net.Mail.SmtpException ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error,
                       MessageBoxDefaultButton.Button1);
    }
    catch (FormatException ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error,
                       MessageBoxDefaultButton.Button1);
    }
}
```

Al ejecutar observamos que se puede elegir las acciones que realiza el sistema cuando existe una alarma.

En la opción 2 reproduce un audio y 2 controles parpadean, para la 3ra opción hace lo mismo y aparte manda un correo.



Si usted desea utilizar otra cuenta de correo solo cambie el servidor SMTP.

Práctica 12: Simulación de monitoreo domótico

Fuente: www.multitecnologia.com/practicas/practica12.rar



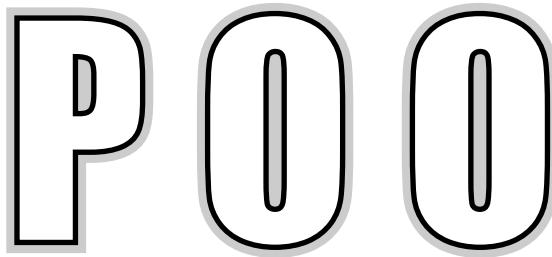
La domótica es otra rama de control enfocada al control inteligente de viviendas, una prioridad es el uso eficiente de energía, la seguridad, conectividad y comodidad.

En este ejercicio vamos a simular un control domótico de luces, audio y alarmas en pantalla, para hacerlo utilizaremos un archivo de Flash previamente programado en ActionScript.

Al ejecutarse el ejemplo usted podrá activar las animaciones de Flash con código en C#.NET, también se activarán 4 audios .mp3 en diferentes tiempos para ejemplificar música programada.

Es importante mencionar que el archivo para esta práctica contiene programación en ActionScript que en el Manual 4 estudiaremos.

PROGRAMACIÓN ORIENTADA A OBJETOS

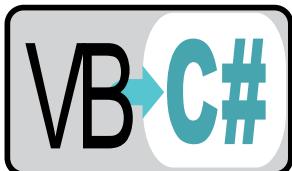


Estimado lector:

Llego el momento de cambiar nuestra manera de razonar un sistema, vamos a entrar a la POO, para esto debemos cambiar nuestros antiguos conceptos de programación.

Durante décadas los programadores desarrollaron aplicaciones que resolvían los mismos problemas una y otra vez. Con la llegada de la POO la ventaja de reutilizar código es un hecho que tarda muy poco en conquistar la mente de un desarrollador.

El tiempo que se ahorra, la capacidad de seccionar los problemas y resolverlos por piezas es el objetivo de esta metodología.



Hasta ahora los ejercicios del curso se han enfocado a programación en base a eventos con llamadas a funciones (estilo Visualbasic 98), en realidad aún no hemos explotado las mejores características del lenguaje, ahora que ya conocemos el entorno, los controles y comandos vamos a diseñar nuestros propios objetos.



En lenguaje terrícola...

La POO es una nueva forma de razonar los sistemas, sin embargo, aún siguiendo los lineamientos no garantiza que los objetos estén bien razonados, depende de la forma en la que el programador ve los procesos y el entorno a modelar, para razonar con sentido común y plasmarlo en código toma tiempo y mucha práctica.

Origen “Todo nace de una necesidad”

A medida que se complica el diseño de software las metodologías evolucionan, un gran paso se dio en los 60's con el Simula 67.

¿Qué es el Simula 67?

Es un lenguaje diseñado para simulación diseñado en el “Centro de Cómputo Noruego” en Oslo, sus creadores fueron Ole-Johan Dahl y Kristen Nygaard, su labor era programar el comportamiento de naves con diferentes cualidades.

La Idea

La idea fue agrupar cada tipo de nave en una clase de objetos, al mismo tiempo cada clase tendría sus propiedades y métodos, con este concepto era mas sencillo razonar la simulación.

El resultado

Al ejecutar sus rutinas observaron como las características particulares de cada nave afectaban al comportamiento del grupo, este fenómeno se daba al momento de combinarse los objetos, mas tarde daría inicio a un nuevo lenguaje: Smalltalk.

¿Smalltalk?

La primera versión fue escrita en Basic, después se codificó con Simula en el XEROX PARC, su principal característica fue crear y modificar objetos en tiempo de ejecución.

¿Crear y modificar objetos en tiempo de ejecución?

En este Manual vamos a comprobar este concepto, en pocas palabras es crear una instancia mientras el programa se ejecuta.

La evolución...

La POO logró posicionarse a mediados de los ochenta con C++, su enfoque a la simulación lo hizo especial para el desarrollo de videojuegos, al mismo tiempo le permitió mejorar las interfaces gráficas.

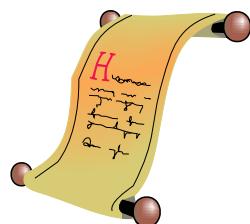
Durante los 70's el equipo de investigación de XEROX PARC acelero la implementación de la POO bajo la tutela de Alan Kay.



Ole-Johan Dahl y Kristen Nygaard
Centro de Cómputo Noruego



Alan Kay
Centro de Investigación XEROX PARC



Una frase popular de Alan Kay es:
“La gente que ama el software quiere construir su propio hardware”

Enemigos de la POO

La POO siempre ha sido motivo de controversia, uno de los casos mas populares se dio con un gran aportador a las ciencias computacionales:

Edsger Wybe Dijkstra fue un científico de computación holandés, sus aportaciones a la informática fueron conceptos de algoritmos como "El camino mas corto", "Algoritmo del banquero", la construcción de un semáforo para coordinar varios dispositivos, computación distribuida, etc. Un hecho curioso es que nunca estuvo de acuerdo en los conceptos y ventajas de la POO, su argumento era que un programa bien estructurado era suficiente y una mejor solución a los problemas de diseño.

Otro gran aporte de Dijkstra fue el libro "The Humble Programmer" donde hizo referencia a la "**Crisis del software**", dicho término fue un tema importante en la reunión de la OTAN en 1968, posteriormente le hizo acreedor al premio "Turing" en 1972.

"Crisis del software" 1968

En este documento agrupo una serie de sucesos que se venían observando en proyectos de software y aún en nuestros días siguen apareciendo.

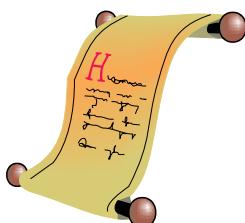
- Los proyectos no terminaban en plazo.
- Los proyectos no se ajustaban al presupuesto inicial.
- Baja calidad del software generado.
- El software no cumplía las especificaciones.
- Código difícil de mantener que dificulta la gestión y evolución del proyecto.

Algunas de sus frases favoritas:

"La **simplicidad** es un prerequisito para la confiabilidad"

"Decir que una computadora puede pensar, es como decir que un submarino puede nadar"

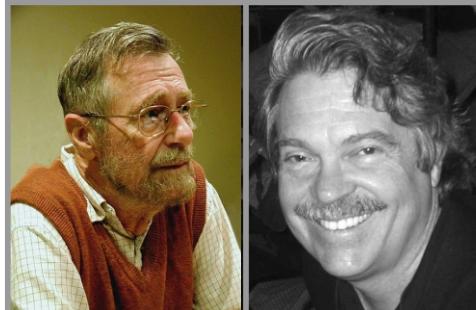
Dijkstra fue un enemigo de la instrucción GO TO y el lenguaje BASIC, su frase favorita era ...
"Mutila la mente más allá de toda recuperación".



Edsger Dijkstra y Alan Kay fueron contemporáneos, su relación fue complicada precisamente por el tema de la POO, sus comentarios más célebres fueron los siguientes...

"La programación orientada a objetos es tan mala idea que solo pudo originarse en California".

Edsger W. Dijkstra



"En ciencias computacionales la arrogancia se mide en nano-Dijkstras".

Alan Kay

Fuente de imágenes: Wikimedia Commons
http://es.wikipedia.org/wiki/Archivo:Alan_Kay2.jpg
http://es.wikipedia.org/wiki/Archivo:Edsger_Wybe_Dijkstra.jpg

Aún con sus opositores la POO ha demostrado adaptarse a los nuevos tiempos y proporcionar más ventajas a los desarrolladores.

El problema para comprender la POO

En POO existe un problema para las carreras relacionadas con electrónica, un cambio en la forma de razonar el concepto de dato.

Un estudiante de electrónica lo primero que aprende es a manejar circuitos lógicos, los estados binarios representados por un nivel lógico (0-5 Volts), permiten “ver” los bits agregando un simple led.

El siguiente paso es guardar ese dato en un flip flop o memoria eprom, ésta es la primera noción de “dato” que se adquiere en el área de electrónica, un estado lógico que representa información.



Posteriormente aprendemos a programar en ensamblador o algún lenguaje como Turbo C, ahora declaramos variables fácilmente y los datos pueden ser de tipo Entero, Flotante, Carácter, String, etc. Sin embargo no dejan de ser estados lógicos almacenados en la memoria del sistema.

```

File Edit Search Run Compile D
[N] --- NELINCPP_ST"1NOLD_C_~1NBO
#include <stdio.h>
#include <conio.h>

int i, j, inpt;
ar[20];

main()
{
    clrscr();

    printf("Enter number (1 to 20) ? ");
    scanf("%d",& inpt);

    ar[0] = ar[1] = 1;
    printf("\n 1 1");
}

```

El siguiente paso fue los lenguajes orientados a eventos como VisualBasic (98), aunque ya era posible manejar algunos conceptos de objetos, la mayoría seguía con el mismo concepto de tipos de dato y el espacio que ocupaban en memoria.

```

TicTacToe - frmTicTacToe (Code)
(General) (Declarations)

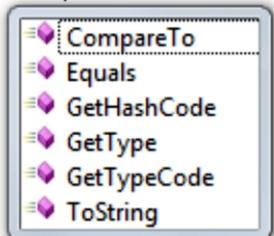
Option Explicit
' Scoring arrays
' These are added to determine when
' a player has won
Dim iXPos(3, 3) As Integer
Dim iOPos(3, 3) As Integer
' Determines who plays next AND
' Used as boolean test variable
Dim sPlaySign As String
' Counts the number of moves
Dim iMove As Integer
' Scoreboard variables
Dim iXScore As Integer
Dim iOScore As Integer

```

Con C#.NET nuestro concepto de dato queda obsoleto, ahora **TODO ES OBJETO**, la vieja idea de un grupo de bits representando información debemos eliminarla, como objetos tienen métodos muy ventajosos que nos ahoran tiempo y facilitan la implementación.

Por ejemplo, si declaramos un objeto entero como:

```
int numero = 0;
int num= numero.
```



Al escribirlo y poner un punto tenemos variedad de métodos a elegir, en este caso “.ToString” nos permite convertir nuestro objeto en una cadena.

¿Cómo debe ser un monitoreo industrial?

En sentido práctico, un operador debe conocer las alarmas aún estando a gran distancia de la pantalla y sin necesidad de llegar a dar un click, bajo condiciones de riesgo no hay tiempo para eso, la interfaz debe mostrar el estado del proceso con la mayor claridad posible, una decisión a tiempo puede salvar vidas.



La complejidad de los procesos...

En automatización existe una ventaja a la hora de entender un proceso... "lo podemos ver".

Un simple recorrido por un proceso industrial puede dejarle un gran panorama de la complejidad del proyecto, algo que no pasa con sistemas administrativos, financieros, didácticos, videojuegos, etc.

Un ejemplo sencillo es el proceso de una Mina de oro, hasta un niño puede entender como las piedras caen a una quebradora y se van desintegrando hasta llegar a polvo.

¿Por qué es sencillo de entender?

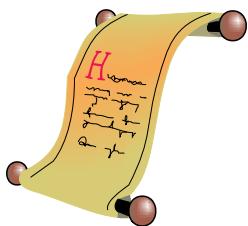
Lo que se puede ver es sencillo recordarlo, existe un gran problema al no ver un proceso con los ojos, los sistemas comunes como los administrativos son mas complejos de memorizar, para empezar son ideas de personas, las ideas no son objetos tangibles, después se deben representar en papel o diagramas, todo esto puede llevar días, semanas o meses.

El problema con el software a la medida...

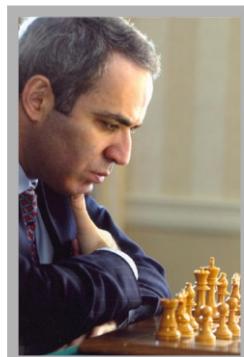
¿Alguna vez ha escuchado esta frase?... "El software a la medida no es negocio", por experiencia puedo afirmarle que es verdadera en un 90%, lamentablemente es una profesión incomprendida, el mercado está saturado tanto de productos como profesionales.

La gran confusión...

La raíz del problema es la ignorancia, lamentablemente el cliente común no conoce sobre programación, en general se tiene la idea que la computadora es realmente inteligente y hace todo el trabajo.



En 1997 surgió uno de los mejores ejemplos, Deep Blue venció al campeón mundial de ajedrez Gary Kasparov, los titulares de noticias del mundo repitieron... "la computadora vence al humano", "la computadora es más inteligente", en ningún momento se dio crédito a los programadores que hicieron el algoritmo, la gente pensó que la máquina evolucionó y pensó por sí misma, lamentablemente continúa ese problema.



VS



Fuente de imágenes:

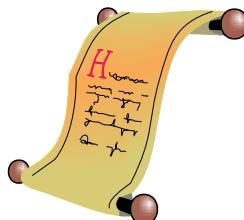
http://upload.wikimedia.org/wikipedia/commons/b/be/Deep_Blue.jpg

<http://upload.wikimedia.org/wikipedia/commons/d/dc/Kasparov-29.jpg>

La delimitación de objetivos

El primer paso es poner por escrito los objetivos de la solución, es mucho mas complejo de lo que parece, primero debe extraer la esencia de la idea de su cliente, lógicamente quiere algo, solo que no puede explicarlo.

Hace tiempo tuve una experiencia que me dejó en claro el tema:



Un cliente tenía años con una idea, con mucho trabajo reunió el presupuesto para llevarla a cabo, ya tenía planeado a quien se lo iba a vender y hasta presumir, también estaba organizada la fiesta de presentación del producto y hasta el “slogan” publicitario, solo faltaba un ligero detalle...
¿Qué iba a hacer exactamente el producto? lo más preocupante es que no existían documentos con descripción, cuando le preguntaba por una idea general escrita en una “servilleta” ¡el cliente no podía! eran conceptos en el aire que ni siquiera el progenitor podía escribir y delimitar, aquí radica el problema del software innovador, es complejo fijar los objetivos.

Como tener éxito con el software a la medida...

Debe aprender a programar módulos reutilizables, no solo en software, también en hardware, la arquitectura de objetos va enfocada precisamente a eso.

La mejor solución...

Colegas, dediquen su conocimiento al diseño de productos propios, integren módulos de hardware y software y podrán competir en innovación, desarrollar en un solo lenguaje no basta, los productos o sistemas grandes llevan programación a varios niveles, no es necesario ser un experto en cada lenguaje, con un conocimiento básico y práctico es suficiente.

Integración de lenguajes

Si va a crear productos es mejor conocer lo práctico de cada lenguaje que ser experto solo en uno, por ejemplo, si usted programa microcontroladores y desea agregar algoritmos para controlar displays gráficos, impresoras, protocolos complejos, audio y guardar grandes cantidades de datos, va a pasar varias semanas complicando su vida, mientras tales tareas son muy sencillas con C#, por otro lado, labores de automatización como un control PID requiere componentes extra para una PC convencional, tal reto para un PLC es muy fácil, simplemente por que está diseñado para eso y representa una mejor opción.



Existe un viejo refrán que dice: “El que mucho abarca, poco aprieta”, en programación no es 100% cierto, quien entiende la lógica puede emigrar de un lenguaje a otro sin problema, lo único que cambia es la sintaxis, el razonamiento es el mismo.

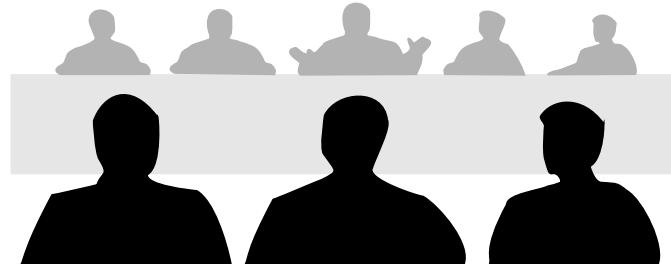
Una historia sobre procesos

¿Qué piensa usted que es más difícil de entender... la lógica de un juego como el Pacman o el proceso de una Mina de Oro?, sin duda, yo le respondería el código del Pacman.

Las razones son las siguientes, lo voy a plantear como una especie de historia...

Tenemos a tres programadores frente a sus clientes, los tres son buenos, manejan el mismo lenguaje y plataforma, ninguno de ellos se conoce, un comité de empresarios les va a asignar el mismo reto:

**“Señores, necesitamos que cada uno programe el videojuego del Pacman”,
debe estar en el menor tiempo posible para nosotros venderlo.**



Para facilitarles el diseño se les proporcionan los mismos gráficos y un detalle de las reglas del juego, no se les entregan diagramas UML ni estándares de programación, el comité no sabe nada de eso, simplemente quiere su juego.

Llega el día que se entregan los requerimientos a los tres, el comité les dice: “**el primero en regresar con el juego gana la competencia**”.

Pasan tres semanas y por coincidencia los tres entregan su juego el mismo día, los programadores nunca compartieron ideas, cada uno trabajo en su reto a su manera, cuando el comité de empresarios juega las tres versiones se da cuenta que cumplen con los requerimientos, como usan los mismos gráficos lucen exactamente igual, sin embargo existe algo curioso con los tres códigos...

¡Son completamente diferentes!, su cantidad de clases con propiedades y métodos van de acuerdo al razonamiento personal de cada programador.

Horas después el comité pide a cada programador que analice el código de sus dos colegas, después de leer cuidadosamente comienzan a intercambiar comentarios entre ellos, ¿Por qué no lo hiciste así?, “Esta interesante pero te ahorrarías mas memoria de esta manera”, “De plano no entiendo esta parte...”. En resumen, se da un acalorado debate de ideas.



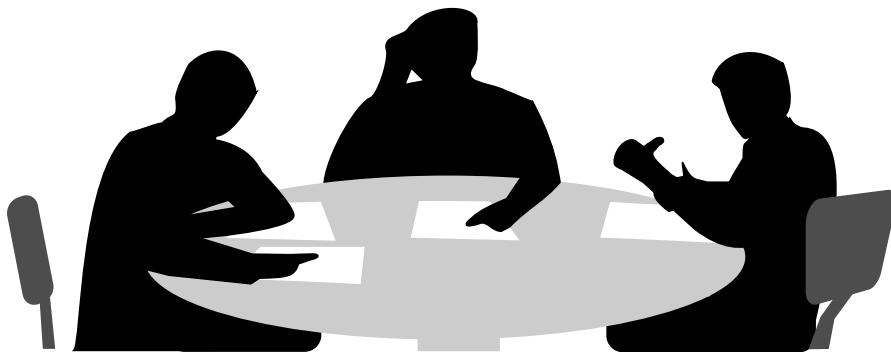
Los tres programadores son contratados y se les asigna otro reto, van a implementar el proceso de control para 3 Minas de Oro.

Como en el reto anterior se les asignan los mismos requerimientos de control y la responsabilidad de una Minera en diferentes lugares, quien implemente el proceso en menor tiempo gana el reto.

Cada programador recibe sus PLC's necesarios y cuenta con toda la obra eléctrica, civil, mecánica y de cableado implementados, solo tiene que preocuparse por la lógica de control en lenguaje ladder.

Comienza el reto, pasan algunas semanas y por segunda ocasión los tres terminan el mismo día sin haberse comunicado entre ellos, el comité arranca los procesos y comprueba que están bien implementados, pero... ¿Cómo luce el código?

Para sorpresa todos los códigos ladder son bastante similares, cada programador puede entender y modificar fácilmente el razonamiento del colega.



¿Qué pasó?

Los procesos industriales son más sencillos de entender y razonar.

- A diferencia de un videojuego o sistema complejo, el proceso se puede explicar sin documentación o pizarrones, por lo regular solo es necesario un recorrido por la planta.
- La cantidad de dispositivos es finita, por lo tanto las variables y objetos, esto limita a que se creen datos innecesarios por programadores inexpertos.
- Las bases de datos son sencillas y con pocas relaciones, solo guardan muestras y datos de operación.
- El lenguaje Ladder tiene controles PID para lazo cerrado que facilitan el control y dan un solo camino para implementarlo.
- Las operaciones a nivel de bit son más sencillas de razonar.
- Los procesos industriales varían poco con el tiempo, primero se realizan de forma manual y después evolucionan.
- Los sistemas donde los procesos son humanos se prestan a tener cambios a medida que cambian las leyes de impuestos, mercantiles, burocráticas, etc.
- El mantenimiento de código industrial es más sencillo, los sensores y actuadores alimentan el sistema, se evitan fallas por captura de personas.

Con esta ligera historia no quiero desmeritar la complejidad de la programación industrial, sólo dar énfasis a que generalmente son procesos más sencillos de entender.



OLE for Process Control

El estándar OPC permite la comunicación entre dispositivos de uso industrial para aplicaciones de control, Microsoft lanza la primera versión de OPC para Windows 3.0, desde entonces a evolucionado hasta adaptarse a los nuevos estándares de comunicación.

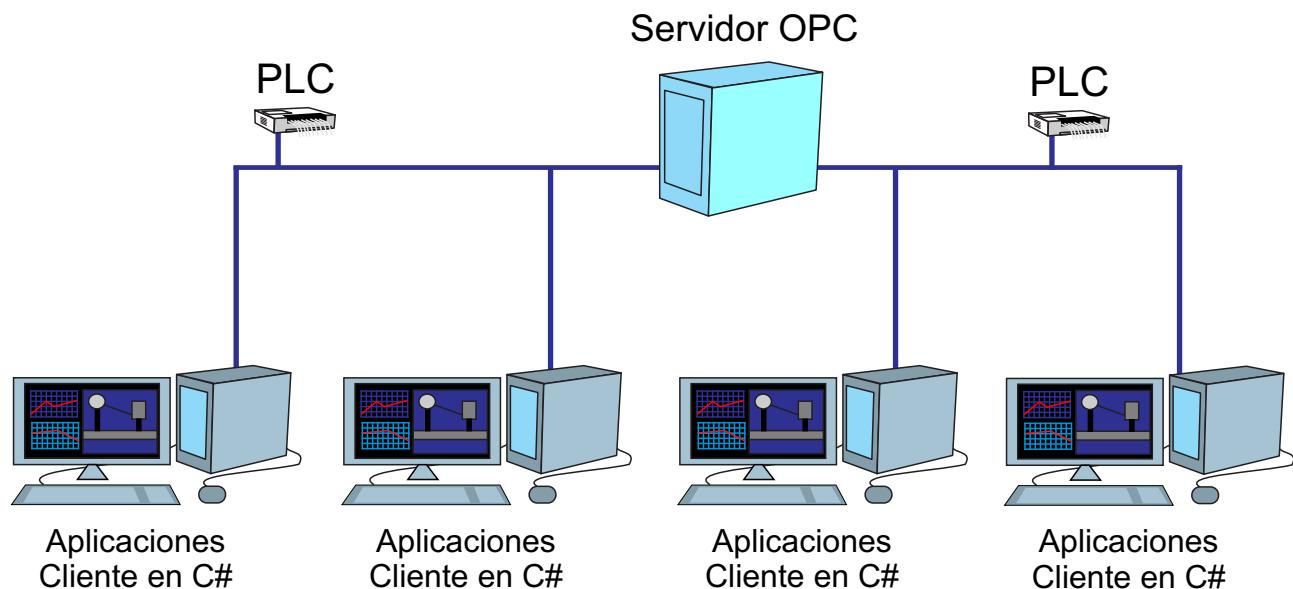


Los fabricantes se preocupan por integrar este protocolo a sus productos, cada uno crea sus propias interfaces para compartir sus recursos, es una especie de driver similar al de impresoras que se comunican con Windows.

OPC es regulado por la fundación (<http://www.opcfoundation.org/>), su objetivo es facilitar el uso de esta tecnología y proporciona código fuente para que diferentes fabricantes implementen sus servidores y clientes.

Objetivo de OPC

El estándar permite acceder el intercambio de datos entre servidores y aplicaciones.



Ventajas de compatibilidad

El éxito de OPC se fundamenta en que los fabricantes solo deben desarrollar el software cliente para sus productos, aparte no tienen que adaptar controladores ante cambios de hardware.

Organización de los datos

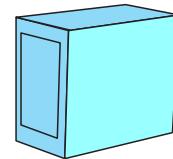
La información se compone de varios elementos:

- El servidor: Actúa como contenedor de todos los datos
- Grupo de objetos: Es el mecanismo para almacenar los elementos (ítems).
- Elementos: Son las conexiones a cada fuente de dato (Tags).

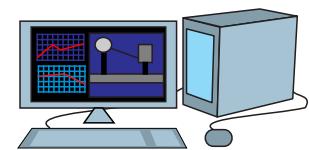
Arquitectura “Cliente-Servidor”

¿Qué es un servidor?

Un servidor es simplemente un programa que recibe peticiones y responde, por lo regular se simboliza con CPU grande, sin embargo es solamente una aplicación, las máquinas pueden correr varios servidores de diversos tipos a la vez.



Un servidor OPC se encarga de leer y actualizar todos los datos de una red de control, su labor es centralizar las operaciones y servir a las máquinas cliente que muestran el proceso en pantalla.



Aplicaciones Cliente con C#

¿Qué tan complejo es desarrollar nuestra aplicación cliente en C#?

Son 3 pasos:

- Descargar e instalar un software servidor OPC (Kepware)
- Dar de alta los equipos con sus tags en el servidor
- Programar su aplicación en C# usando componentes (KepWare ClientAce).

¿En qué consiste configurar el servidor OPC?

El servidor se instala en cualquier sistema operativo Windows, le recomiendo XP por ser más estable y flexible, posteriormente debe conectar sus equipos por red, puerto serie, etc.



En nuestro curso vamos a integrar un servidor OPC comercial, le recomiendo integrar este tipo de programas para reducir el tiempo de implementación.

Práctica 43:

Corrimiento de un bit en el puerto A imitando el efecto del “Auto Increíble”.

Fuente: www.multitecnologia.com/practicas/practica43.rar

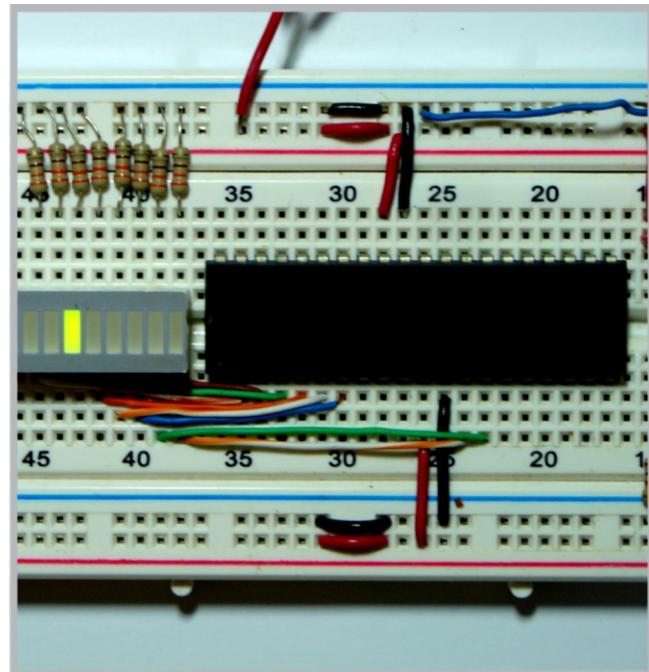
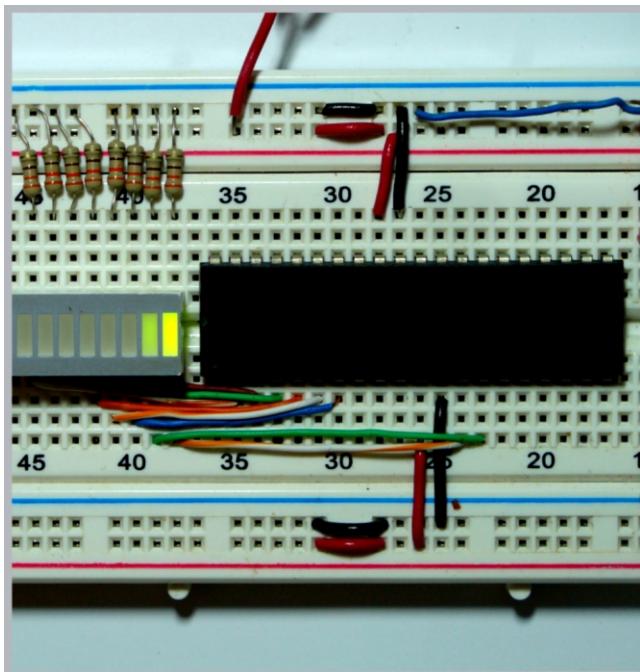
Requerimientos: Se utiliza el circuito de la práctica 1.

Importante:

- #byte puertoA = 0x05 nos facilita el manejo del puerto A
- Observe como es posible multiplicar, dividir y escribir en la salida del PIC directamente:
puertoA=puertoA * 2;
puertoA=puertoA / 2;

```
//Programa que enciende los 8 bits del puerto A con un corrimiento
//estilo "El Auto Increíble"
#include <16F887.h>
#FUSES INTRC_IO           //Internal RC Osc, no CLKOUT
#use delay (clock=8000000) //considera el reloj para el delay a 8Mhz
//para un manejo sencillo del puerto A asignamos su dirección
#byte puertoA = 0x05
void main()
{
    set_tris_a(0x00); //puerto A como salida para Leds
    //desactivamos comparadores del puerto A para que no afecte RA5
    setup_comparator(NC_NC_NC_NC);
    puertoA=1; //encendemos el bit0
    while(1) //se repite por siempre
    {
        while(puertoA!= 128)//ejecuta mientras sea diferente a 128
        {
            //al multiplicar por 2 va generando 2,4,8,16,32,64,128
            puertoA= puertoA *2;
            delay_ms(200); //espera 200 milisegundos
        }
        while(puertoA!= 1)//ejecuta mientras sea diferente a 1
        {
            //al dividir entre 2 va generando 64,32,16,8,4,2,1
            puertoA= puertoA /2 ;
            delay_ms(200); //espera 200 milisegundos
        }
    }
}
```

Al ejecutar observe como el bit se desplaza de un lado a otro:



Lo interesante de este ejemplo es la manera como se realiza la operación, no estamos escribiendo valores fijos paso por paso, es una operación aritmética que va duplicando el valor cuando va en un sentido y dividiéndolo entre 2 cuando viene de regreso, observe el código:

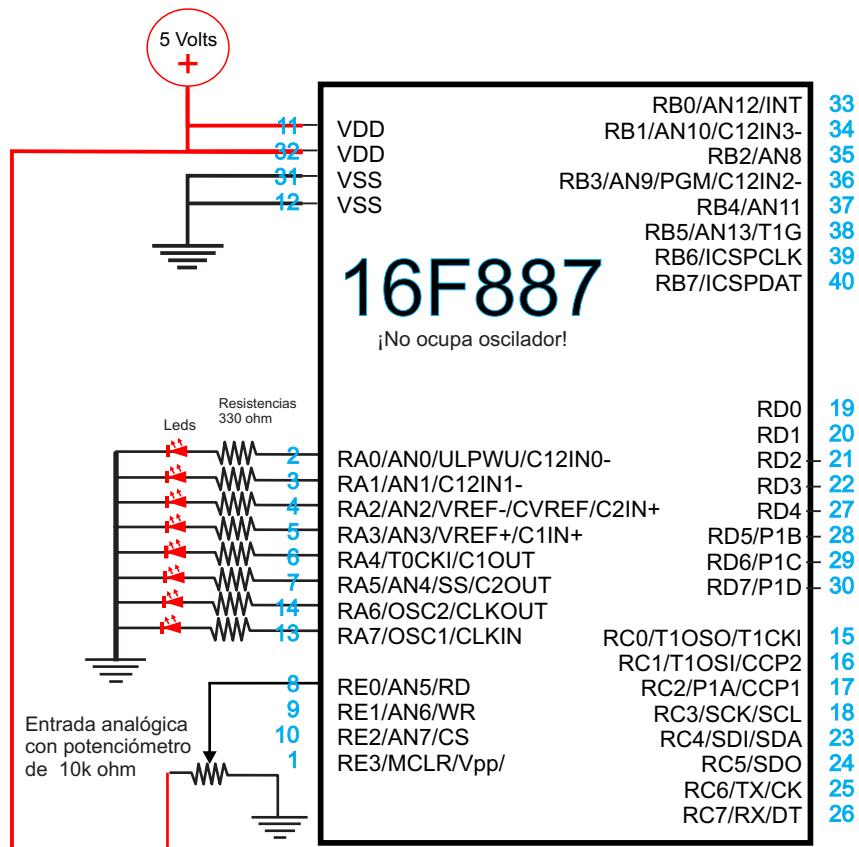
```

while(1) //se repite por siempre
{
    while(puertoA!= 128)//ejecuta mientras sea diferente a 128
    { //al multiplicar por 2 va generando 2,4,8,16,32,64,128
        puertoA= puertoA *2;
        delay_ms(200); //espera 200 milisegundos
    }
    while(puertoA!= 1)//ejecuta mientras sea diferente a 1
    { //al dividir entre 2 va generando 64,32,16,8,4,2,1
        puertoA= puertoA /2 ;
        delay_ms(200); //espera 200 milisegundos
    }
}

```

Práctica 44: Lectura de puerto analógico RE0 y despliegue en puerto A.

En este ejemplo vamos a leer un dato analógico (0-5 volts) del puerto Re0, el byte será reflejado en el puerto A por medio de leds. Fuente: www.multitecnologia.com/practicas/practica44.rar

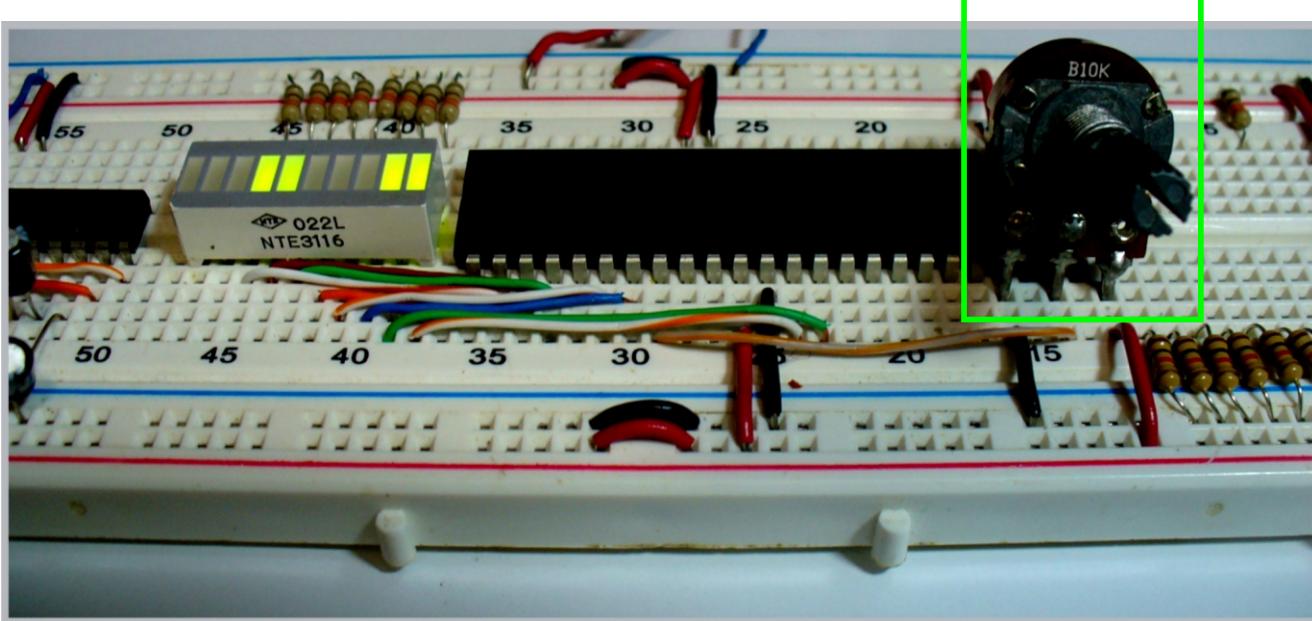


```
//Programa que lee el puerto analógico AN5 y lo despliega en el puerto A
```

```
#include <16F887.h>
```

```
#use delay (clock=8000000) //considera el reloj para el delay a 8Mhz
int valorAnalogo;           //aqui almacenamos el valor del puerto RE0
void main()
{
    set_tris_e(0xFF); //puerto E como entrada
    set_tris_a(0x00); //puerto A como salida para Leds
    SETUP_ADC_PORTS(sAN5); // determina que el puerto RE0/AN5 será analógico
    SET_ADC_CHANNEL(5);
    setup_adc(ADC_CLOCK_DIV_32);
    //desactivamos comparadores del puerto A para que no afecte RA5
    setup_comparator(NC_NC_NC_NC);
    while(1) //se repite por siempre
    {
        valorAnalogo=read_adc(); //leemos el puerto RE0/AN5
        output_A(valorAnalogo); //asignamos el valor analógico al puertoA
        delay_ms(100); //espera 100 milisegundos
    }
}
```

Al variar el potenciómetro de 10 k ohm puede comprobar como se refleja el valor digital en el puerto A, la resolución es de 8 bits.



IMPORTANTE

Los siguientes ejercicios utilizan librerías protegidas por derechos de autor del fabricante (<http://www.ccsinfo.com/>), para avanzar requiere comprar la licencia del software con las librerías:

LCD.C, STDIO.H, STRING.H, INPUT.C.

Práctica 45: Manejo de un Display LCD

Fuente: www.multitecnologia.com/practicas/practica45.rar

Vamos a generar un proyecto como en el primer ejemplo de este manual, en nuestro archivo fuente principal lo nombramos: pruebasLcd.c

Con este programa se puede controlar un display LCD común con la librería LCD.C, primero imprime 2 mensajes y después entra en un ciclo donde empieza a contar imprimiendo el resultado en el display.

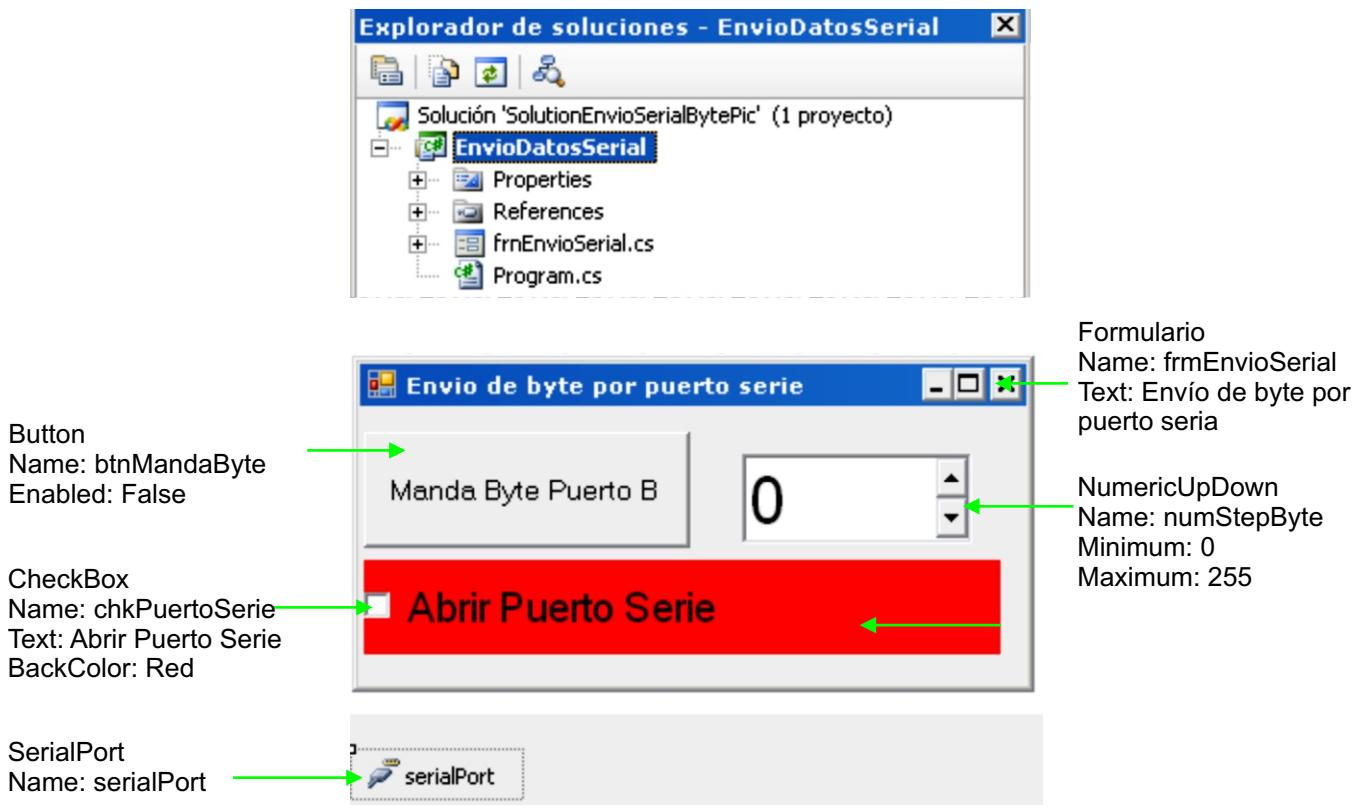
LCD.C debe estar en la misma carpeta donde esta el proyecto y nuestro archivo fuente, también es el lugar donde se genera el hexadecimal cuando compilamos.

Tipos	Nombre
Archivo C	LCD
Archivo C	Icdbasico887
Archivo PJT	Icdbasico887.pjt

Práctica 62: Envío de datos por puerto serie al PIC

Este ejercicio esta vinculado a la práctica 10 del manual 7, el objetivo es mandar un byte por puerto serial al PIC.

Creamos un nuevo proyecto de Windows Forms y agregamos los siguientes controles:



```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace EnvioDatosSerial
{
    public partial class frnEnvioSerial : Form
    {
        public frnEnvioSerial()
        {
            InitializeComponent();
        }
    }
}

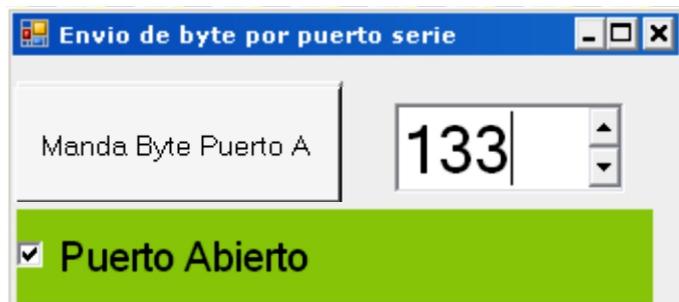
```

```

private void chkPuertoSerie_CheckedChanged(object sender, EventArgs e)
{
    if (chkPuertoSerie.Checked)
    {
        try
        {
            if (!serialPort.IsOpen) //si el puerto esta cerrado
            {
                serialPort.Open();
            }
            btnMandaByte.Enabled = true;
            chkPuertoSerie.Text = "Puerto Abierto";
            chkPuertoSerie.BackColor = Color.YellowGreen;
        }
        catch
        {
            MessageBox.Show("Algo anda mal con el puerto COM");
            chkPuertoSerie.Checked = false;
        }
    }
    else
    {
        try
        {
            if (serialPort.IsOpen)
            {
                serialPort.Close();
                chkPuertoSerie.BackColor = Color.Red;
                chkPuertoSerie.Text = "Puerto Cerrado";
                btnMandaByte.Enabled = false;
            }
        }
        catch
        {
            MessageBox.Show("Algo anda mal no pude cerrar el COM");
        }
    }
}
private void btnMandaByte_Click(object sender, EventArgs e)
{
    // ahora envia el numero entre 0 y 255 del numStepByte acompañado de un Enter
    serialPort.Write(numStepByte.Value.ToString() + (char)13);
}
}
}

```

Al ejecutar ambos programas puede mandar los datos desde el numericStepper al PIC, el dato se refleja en el display LCD.





En los ejemplos anteriores establecimos envío y recepción serial por separado, en el siguiente ejercicio creamos la comunicación bidireccional por medio de mensajes entre C# y el PIC.

Práctica 63: Comunicación serial bidireccional con PIC 16f887

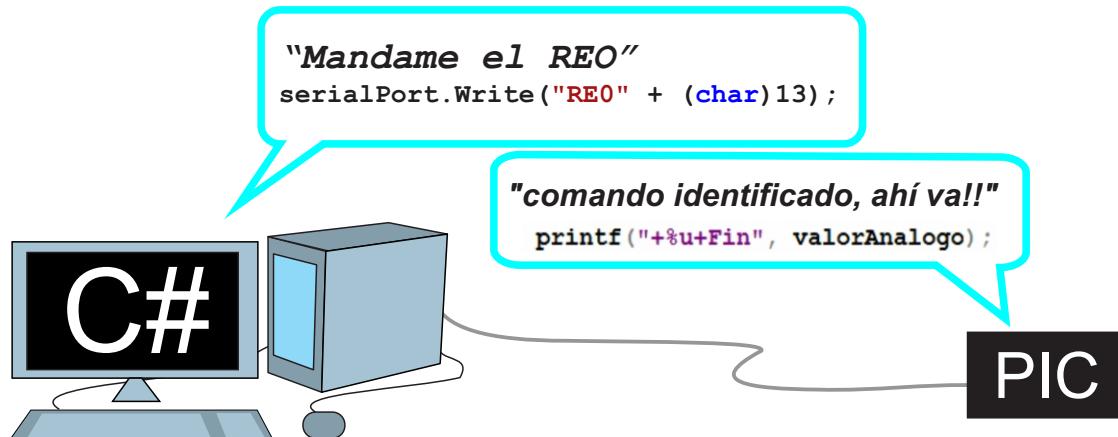
En este ejercicio la cantidad de comandos de intercambio es mayor, por tal motivo tenemos 3 mensajes para que C# y el PIC se comuniquen:

C# manda los siguientes mensajes al PIC:

"escribe": Cuando el PIC recibe esta cadena se queda esperando un número entre 0-255 que imprime en el puerto A.

"RE0": Con este mensaje el PIC nos responde con la conversión a decimal del puerto analogico RE0 (0-5 volts), la resolución es de 8 bits.

"byteEnt" : Esta cadena le dice al PIC que nos responda con los 8 bits de entradas digitales del microswitch.



El PIC está siempre esperando cualquiera de los tres mensajes para responder.



Cada mensaje que manda el PIC tiene un identificadorl con la cadena "Fin", esto permite hacer mas eficiente la lectura en C# y conocer el final de la trama de datos.

Práctica 67: Lectura y escritura básica de puerto paralelo

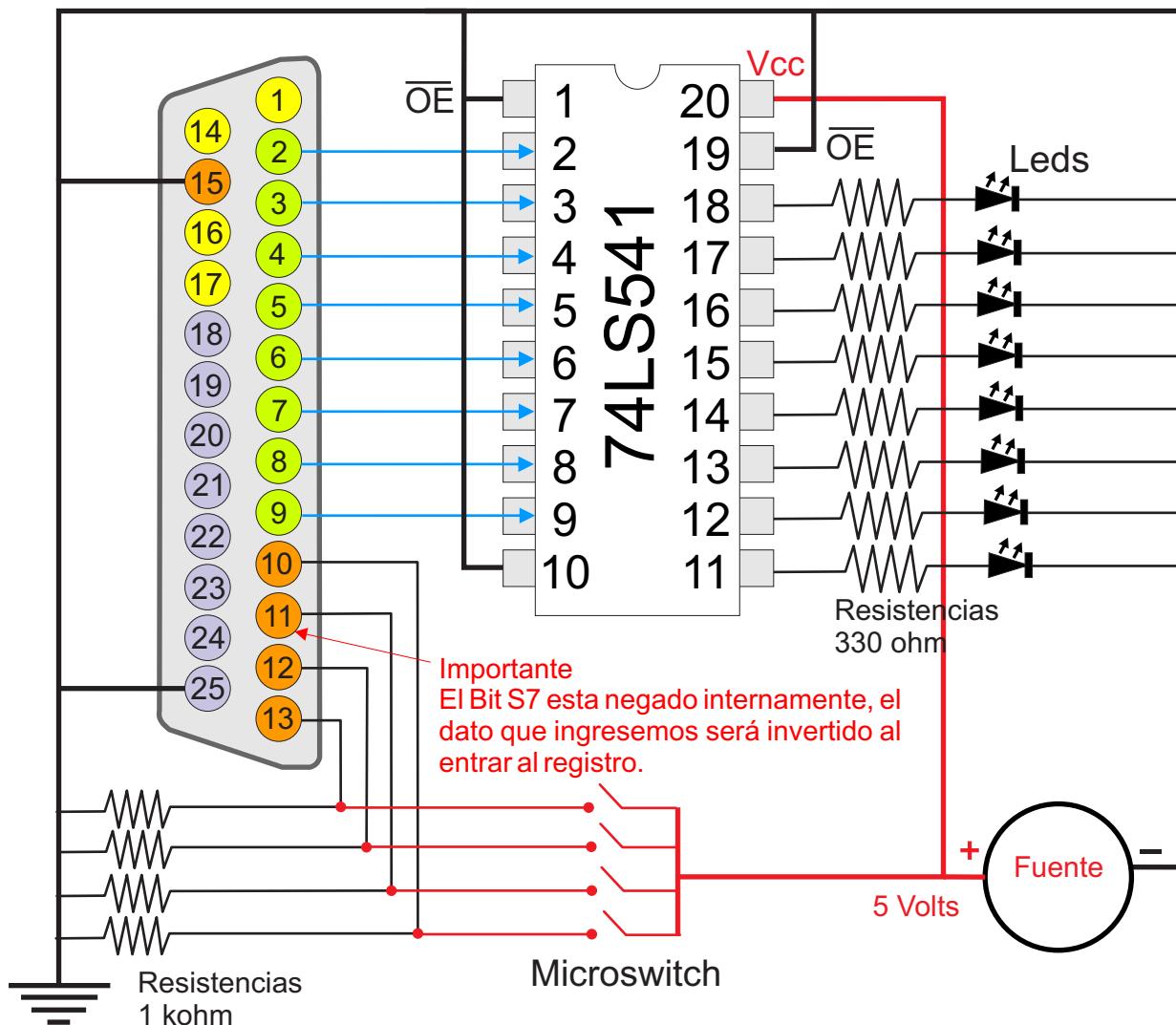
En este ejercicio vamos construir la interfaz electrónica para controlar 8 bits de salida y leer 4 entrada por medio del puerto DB25.

Vamos a implementar el siguiente circuito con el Buffer 74L541 para alimentar los leds y reducir la carga de corriente en el puerto, el material necesario es:

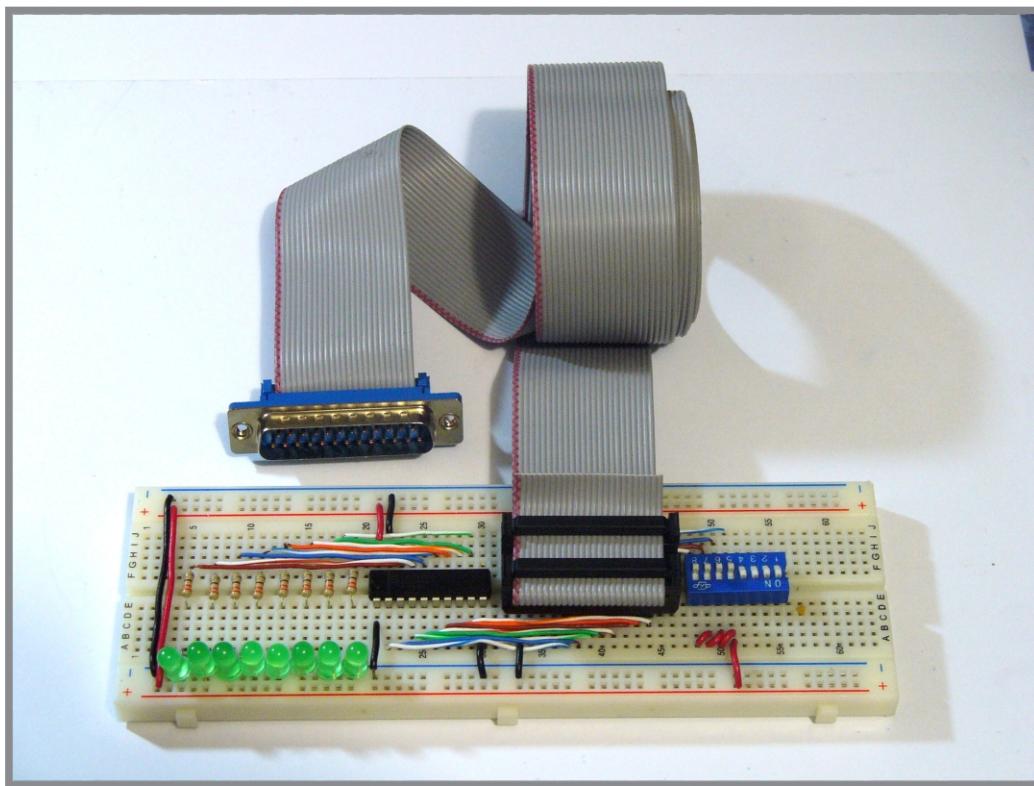
1 Protoboard
1 Buffer 74LS541
1 Microswitch
4 resistencias de 1 kohm
8 resistencias de 330 ohm
8 leds comunes de 5mm
Cableado

Equipo necesario:
1 fuente de voltaje de 5 volts DC
Pinzas, Multímetro.

Diagrama del circuito

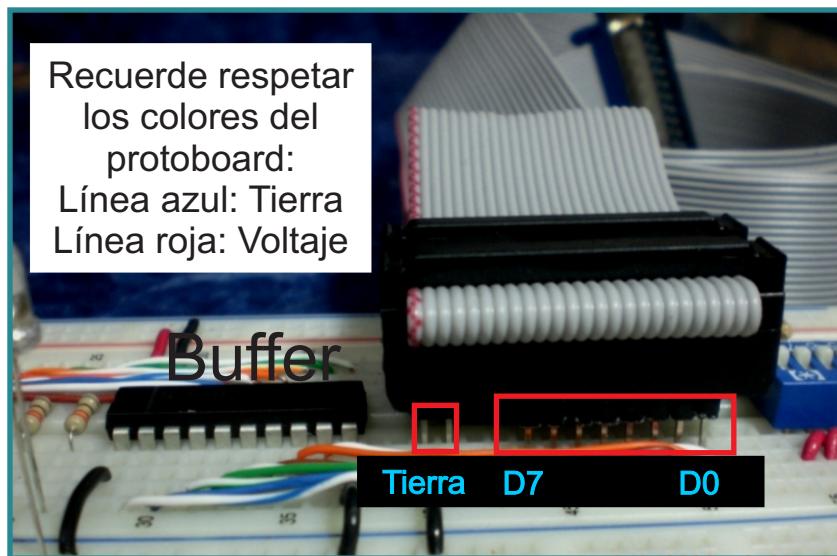


Todo el circuito puede implementarlo en un protoboard.



IMPORTANTE

Trabajar con fuentes externas de voltaje y periféricos de la PC presenta un riesgo de quemar componentes de la tarjeta madre, se debe poner en común la tierra de la PC con la fuente externa y tener mucho cuidado en que no entre voltaje por ese punto a la computadora, esto provocaría un daño irreparable.



El puerto paralelo requiere una librería para funcionar: inpout32.dll, si no la tiene en la carpeta C:\Windows\System32\inpout32.dll debe descargarla desde internet.

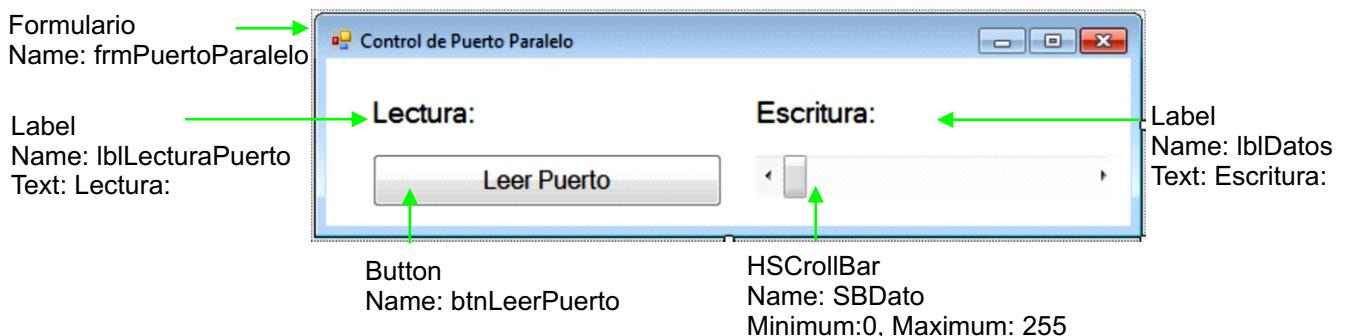
Possiblemente ocupe registrarla, escriba lo siguiente en la ventana de ejecutar:

REGSVR32 c:\windows\system32\inpout32.dll

Con nuestra interfaz electrónica lista nos vamos a VisualStudio y creamos un nuevo proyecto de WindowsForms.



Ahora arrastramos los siguientes controles:



Escribimos el código para los eventos de cada control mostrados:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices; //esta directiva es necesaria
namespace frmPuertoParalelo
{
    public partial class frmPuertoParalelo : Form
    {
        //Creamos esta clase para importar la DLL y tener acceso a sus comandos
        public class PortAccess
        {
            [DllImport("inpout32.dll", EntryPoint = "Out32")]
            public static extern void Output(int adress, int value);
            [DllImport("inpout32.dll", EntryPoint = "Inp32")]
            public static extern int Input(int address);
        }
        public frmPuertoParalelo()
        {
            InitializeComponent();
        }
    }
}
```

```

private void SBData_Scroll(object sender, ScrollEventArgs e)
{
    //escribimos un dato de 8 bits (0-255) en el registro de datos
    //observe que el valor se manda en decimal (int)
    //el valor 888 es la dirección del registro de datos
    PortAccess.Output(888, SBData.Value);
    lblDatos.Text = "Escritura: " + SBData.Value.ToString();
}
private void btnLeerPuerto_Click(object sender, EventArgs e)
{
    //leemos el registro de status con los bits S4, S5, S6, S7 del diagrama,
    //el bit S3 se encuentra forzado a tierra sin el microswitch.
    //los bits S0,S1,S2 no estan disponibles y por lo regular su valor es 0
    //el valor 889 es la dirección del registro de Status
    lblLecturaPuerto.Text = "Lectura: " + PortAccess.Input(889).ToString();
}
}
}
}

```

Antes de conectar el protoboard a la PC probamos las entradas del buffer, observe que por defecto sus salidas están activadas, por lo tanto esta esperando una tierra en su entrada para apagar el led, tomamos cualquier cable largo y lo conectamos a tierra, el otro extremo lo posicionamos en cada entrada del buffer para comprobar como se apaga cada led.

Una vez probado el protoboard verifique que no existan cables sueltos que puedan provocar cortos, use su multímetro para probar continuidad entre el cable DB25 y las entradas del buffer.

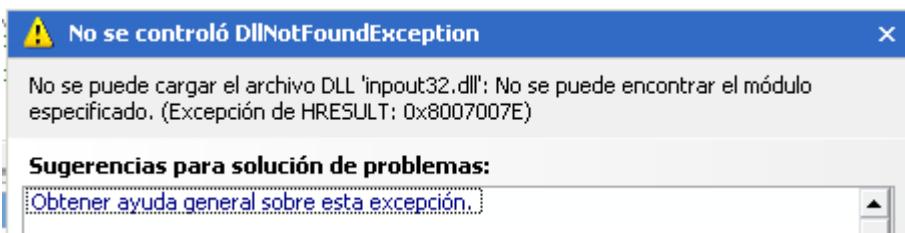


Antes de prender la fuente que alimenta el protoboard observe el monitor de la PC, si nota un parpadeo al encenderla apague rápidamente la fuente o desconecte el cable, esto puede salvar su tarjeta madre, si no hay cambio todo esta bien conectado, en caso de apagarse el monitor su tarjeta acaba de pasar “al otro mundo”.



Cuando conectamos mal el puerto y la tarjeta se daña se debe a un corto circuito interno, en estos casos se queman integrados básicos que pueden ser mas caros de reparar que comprar una tarjeta nueva, tenga

Ejecutamos y al mover el scrollbar los leds deben cambiar su estado:



¿Aparece este mensaje?

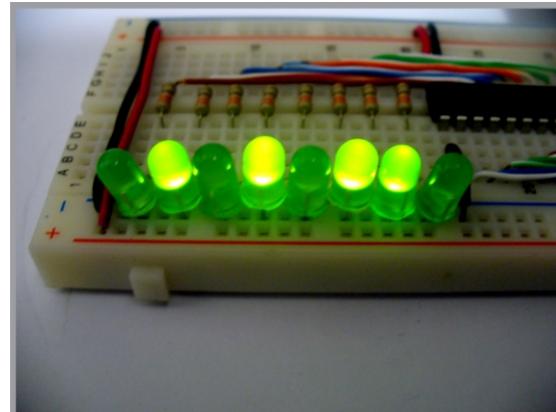
El error se debe a la falta inpout32.dll en la carpeta Windows\system32 o en los ejecutables de su proyecto.



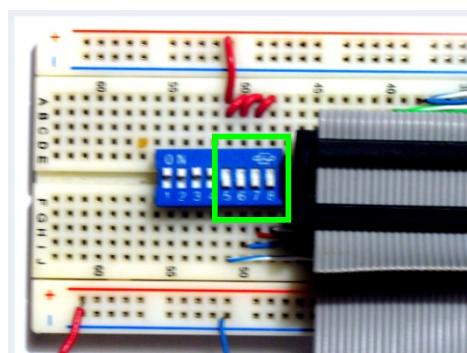
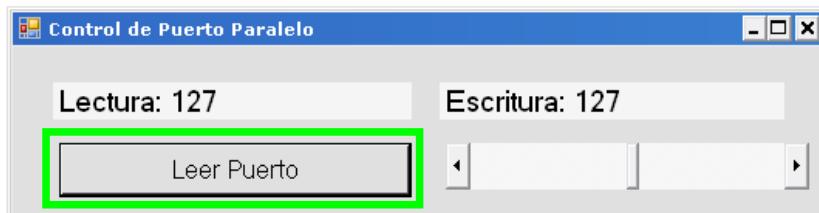
En caso de no existir error usted puede ver el valor del scrollbar representado por los leds.



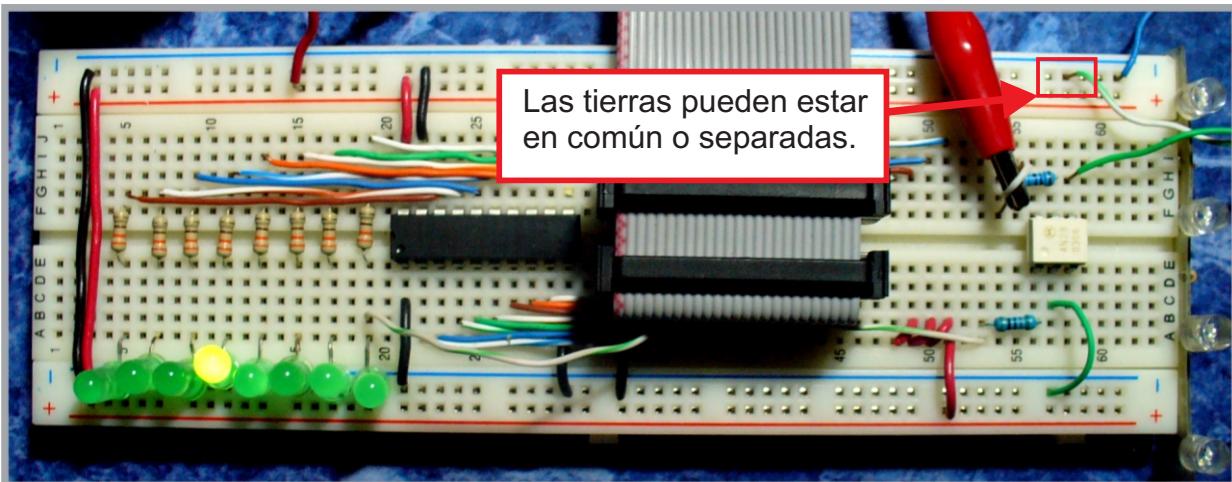
Observe como el dato se actualiza de forma instantánea en los leds, la razón es por utilizar el evento scroll.



Para leer diferentes datos movemos los interruptores del microswitch y damos click en el btnLeerPuerto para actualizar.

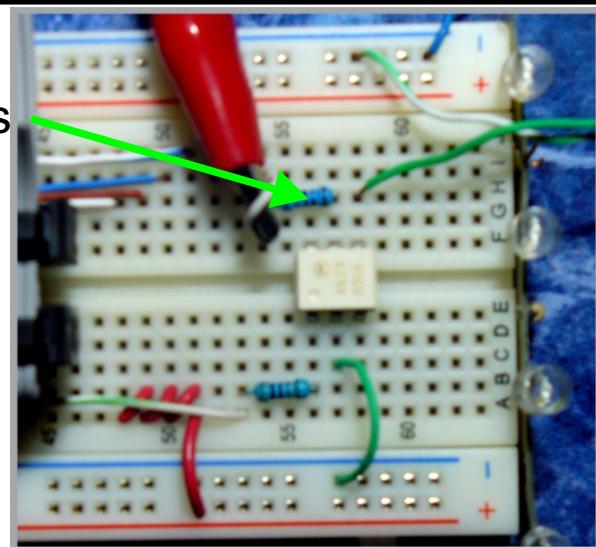


En la imagen podemos ver el circuito con 1 de los optoacopladores, para armarlo completo debe agregar 4 extra.

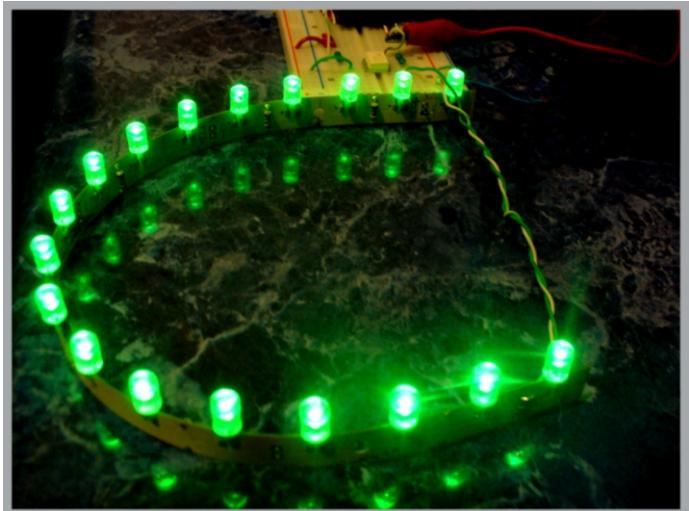
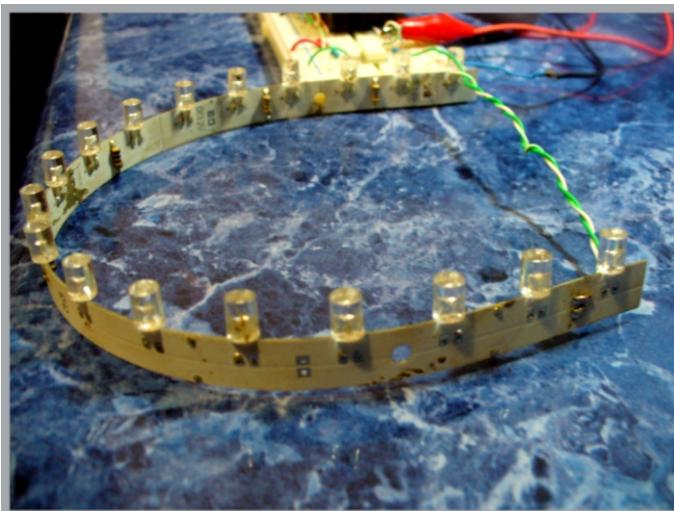


12 Volts

El optoacoplador es una solución sencilla para elevar el voltaje de nuestras luces, también puede conectar motores cuidando que las tierras se encuentren separadas.



La barra de leds esta diseñada para operar con 12 volts.



Vamos a simular el ejemplo anterior en nuestro programa:

Tenemos un proceso que se divide en 5 personas y 5 semáforos, entre todos se reparten el 100% el tiempo que dura en crearse el producto (120 segundos).

Gráficamente las etapas del proceso lucen de esta forma:

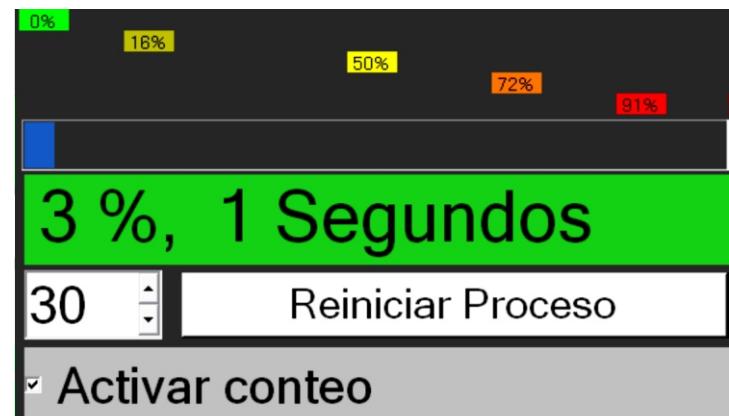


En este bloque de tiempos ajustamos el programa para que al 0% encienda el verde:

Al terminar el verde inicia el color oro (16%), los tres siguientes colores es la suma de cada porcentaje, al final la luz roja enciende al 91% para que solo dure encendida el 9% acordado.



Al activar el CheckBox inicia el conteo y se puede ver el avance en segundos y porcentaje.



Cada vez que cambie el color del proceso se activa un bit del puerto paralelo, usted debe conectar un optoacoplador como lo indica el diagrama.

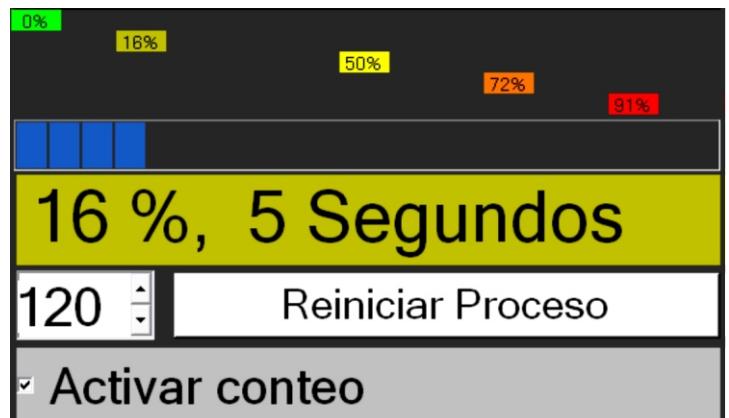
Utilizando barras de leds de 12 volts puede crear semáforos grandes que realmente le sirvan en su proceso.

Registro de datos:00000001



Al pasar de 16 % se activa la segunda etapa y su correspondiente bit en el puerto.

Registro de datos:00000010



En el 50% enciende el amarillo y la barra de progreso debe estar a la mitad.

Registro de datos:00000100



C#

Por años la plataforma Windows ha sido la base para sistemas de monitoreo industrial, en este libro le mostramos como puede crear proyectos de control personalizados con las ventajas de C#.NET.

Si usted cumple con alguno de estos requisitos el curso le resultará útil.

- Ha desarrollado en VisualBasic (98), Turbo C, Pascal, etc. y desea actualizarse.
- Le interesa aprender las bases de la programación orientada a objetos con un enfoque industrial.
- Se dedica a la integración y desarrollo de prototipos y proyectos de automatización.
- Desarrolla sistemas electrónicos y necesita comunicarse con la PC.
- Maneja programación de alto nivel y ocupa aprender a integrar microcontroladores PIC.
- Quiere dar un valor agregado a sus proyectos integrando multimedia y electrónica.

Temario por capítulos:

- 1: Introducción a VisualStudio.
- 2: Programación orientada a eventos.
- 3: Programación orientada a objetos.
- 4: Introducción a Adobe Flash.
- 5: Aplicaciones móviles y control telefónico.
- 6: Comunicación con PLC's y configuración de servidor OPC.
- 7: Lenguaje C para microcontroladores PIC con el entorno CCS.
- 8: Manejo de periféricos (TCP/IP, RS232, Puerto Paralelo, USB).

ISBN: 978-607-00-5217-0



9 786070 052170

Visual Studio, C# y .NET Framework son marcas registradas
propiedad de Microsoft Corporation. Inc.

Descargas de ejemplos y utilerías: www.multitecnologia.com

