

	UNIVERSIDAD DON BOSCO FACULTAD DE ESTUDIOS TECNOLÓGICOS COORDINACIÓN DE COMPUTACION
CICLO: 2/ 2015	<p style="text-align: center;"><i>GUIA DE LABORATORIO #1</i></p> <p> Nombre de la Practica: Introducción a C#.Net Lugar de Ejecución: Centro de Computo Tiempo Estimado: 2 horas y 30 minutos MATERIA: Desarrollo de Aplicaciones con Software Propietario </p>

I. OBJETIVOS

Que el estudiante:

- Reconozca el entorno de herramientas y ventanas que forman al IDE de Visual Studio .Net
- Se familiarice con los tipos de proyectos de C# .NET
- Elabore programas respetando la sintaxis de C#.NET
- Aplique los pasos para la construcción de una aplicación de tipo Windows Form.
- Diferencie uso de los Tipos de Datos y variables que existen en C#.NET
- Pueda declarar variables en diferentes ámbitos/alcances

II. INTRODUCCION TEORICA

¿Qué es Visual C#?

C# (pronunciado si sharp en inglés) es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

El nombre C Sharp fue inspirado por la notación musical, donde '#' (sostenido, en inglés sharp) indica que la nota (C es la nota do en inglés) es un semitono más alta, sugiriendo que C# es superior a C/C++. Además, el signo '#' se compone de cuatro signos '+' pegados.

Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux.

Historia y evolución

Durante el desarrollo de la plataforma .NET, las bibliotecas de clases fueron escritas originalmente usando un sistema de código gestionado llamado Simple Managed C (SMC). En enero de 1999, Anders Hejlsberg formó un equipo con la misión de desarrollar un nuevo lenguaje de programación llamado Cool (Lenguaje C orientado a objetos). Este nombre tuvo que ser cambiado debido a problemas de marca, pasando a llamarse C#.3 La biblioteca de clases de la plataforma .NET fue migrada entonces al nuevo lenguaje.

Supone una evolución de Microsoft C y Microsoft C++; es sencillo, moderno, proporciona seguridad de tipos y está orientado a objetos. El código creado mediante C# se compila como código administrado, lo cual significa

que se beneficia de los servicios de Common Language Runtime. Estos servicios incluyen interoperabilidad entre lenguajes, recolección de elementos no utilizados, mejora de la seguridad y mayor compatibilidad entre versiones.

C# se presenta como Visual C# en el conjunto de programas Visual Studio .NET. Visual C# utiliza plantillas de proyecto, diseñadores, páginas de propiedades, asistentes de código, un modelo de objetos y otras características del entorno de desarrollo. La biblioteca para programar en Visual C# es .NET Framework.

Utilización de C#

C# se puede utilizar para:

- ✱ Programas de escritorio en Windows XP y versiones posteriores.
- ✱ Sistemas o sitios web basados en ASPX utilizando C# como lenguaje de programación.
- ✱ Videojuegos con XNA para pc y XBOX
- ✱ Programas en Linux utilizando mono (implementación de CLR multiplataforma)
- ✱ Sistemas desktop/web que se necesiten conectar a fuentes de datos.

Requerimientos para desarrollar aplicaciones con C#

Para poder realizar programas utilizando C# es necesario tener instalado:

- ✱ El entorno de desarrollo (IDE) más común, que para nuestro caso sería Visual Studio en cualquiera de sus versiones, aunque se recomienda la más actualizada del mercado.

Software de base de datos, principalmente SQL Server y su respectivo sistema gestor de base de datos. Aunque este requisito no es necesario para compilar aplicaciones de C#, si es necesario para el desarrollo de aplicaciones que se conecten a base de datos.

Entorno de Desarrollo

Visual Studio es un conjunto completo de herramientas de desarrollo para la generación de aplicaciones Web ASP.NET, Servicios Web XML, aplicaciones de escritorio y aplicaciones móviles. Visual Basic, Visual C++, Visual C# y Visual J# utilizan el mismo entorno de desarrollo integrado (IDE), que les permite compartir herramientas y facilita la creación de soluciones en varios lenguajes. Asimismo, dichos lenguajes aprovechan las funciones de .NET Framework, que ofrece acceso a tecnologías clave para simplificar el desarrollo de aplicaciones Web ASP y Servicios Web XML.

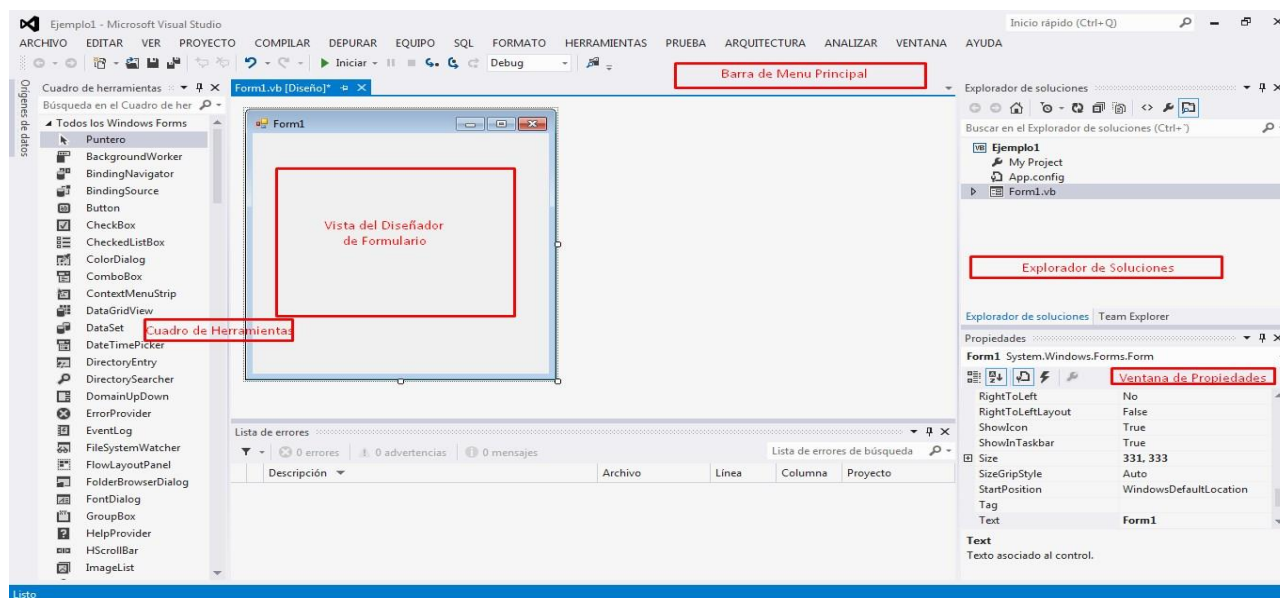


Imagen 1.1: Vista inicial de un proyecto Windows Forms

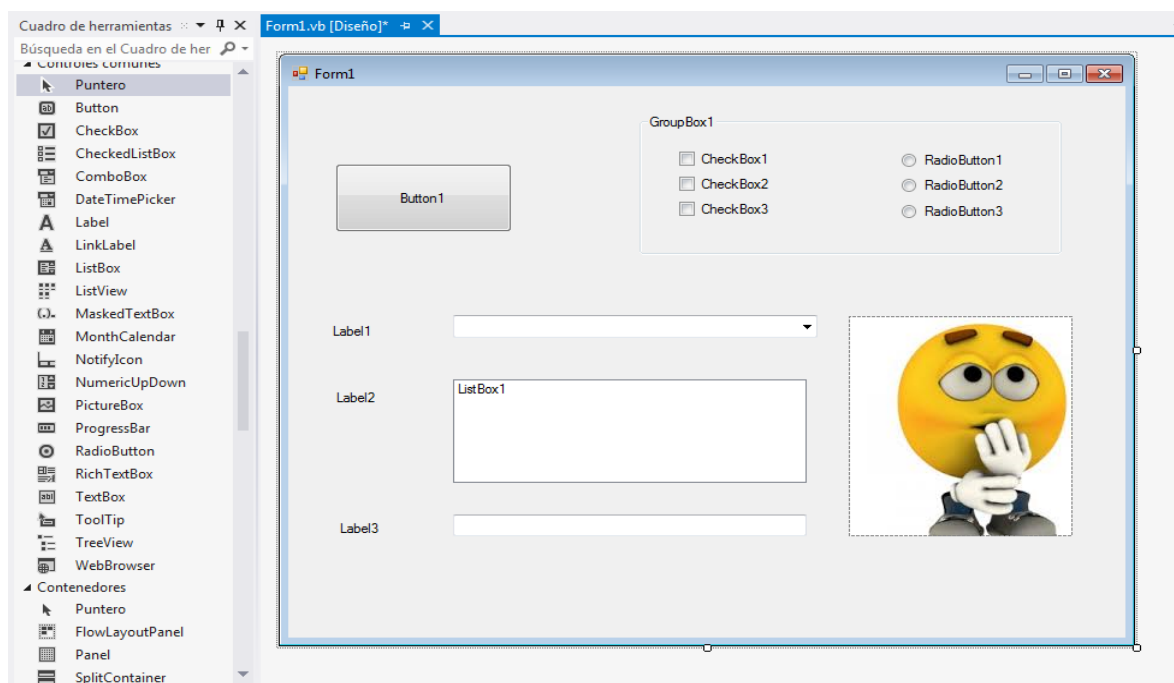


Imagen 1.2: Vista de los controles más comúnmente utilizados en un formulario

Visual C# 2012 incluye entre muchas capacidades, la continuación de bucles, la disposición garantizada de recursos, las propiedades de acceso combinado, los tipos de datos sin signo y que aceptan valores NULL, la sobrecarga de operadores, los tipos parciales y genéricos, los eventos personalizados y la comprobación de la compatibilidad con Common Language Specification (CLS).



Ambiente de desarrollo para Aplicaciones Windows

En C# hay diversas de tipos de proyectos disponibles, de los cuales se trataran a los proyectos de tipo Windows. Estos proyectos presentan un formulario en blanco con las ventanas de apoyo mostradas en la imagen 1.1.

Cuadro de Herramientas

Dentro de un formulario, C# dispone al programador de una serie de controles, los cuales son las maneras de cómo la aplicación puede interactuar con los usuarios a los cuales ira dirigida la aplicación a construir. Con el .NET Framework 4.5 consta de una diversidad de controles y objetos para usar en las aplicaciones, clasificados en categorías. En la imagen 1.2 se muestran un listado de los controles más comunes utilizados dentro de un form.

Una descripción básica de algunos de estos se muestra en la Tabla 1. Observe que cada control tiene su icono, nombre, prefijo y su descripción.

Icono	Nombre	Prefijo	Descripción
Controles comunes			
	Button	btn	Se utiliza para iniciar, detener o interrumpir un proceso
	CheckBox	chk	Muestra una casilla de verificación y una etiqueta para texto. Se utiliza en general para establecer opciones.














Icono	Nombre	Prefijo	Descripción
	ComboBox	cbo	Muestra una lista desplegable de elementos.
	DateTimePicker	dtp	Muestra un calendario gráfico desplegable, que permite que los usuarios seleccionen una fecha o una hora.
	Label	lbl	Muestra texto que los usuarios no pueden modificar directamente.
	ListBox	lst	Muestra una lista de textos, también llamados elementos.
	MaskedTextBox	msk	Captura texto escrito por el usuario, ya sea teniendo un formato de entrada.
	PictureBox	pic	Muestra archivos de imágenes, tales como mapas de bits e iconos, gifs, etc.
	RadioButton	rdb	Muestra un botón que puede activarse o desactivarse.
	TextBox	txt	Muestra texto escrito en tiempo de diseño que puede ser editado por los usuarios en tiempo de ejecución.
Components			
	ImageList	img	Es un contenedor de imágenes, para que después se use con otros controles.
	Timer	tmr	Sirve para realizar un conteo de tiempo, en tiempo de ejecución.
Containers			
	FlowLayoutPanel	flp	Es un contenedor de controles que no permite la modificación de la posición del control.
	GroupBox	gpb	Agrupar un conjunto de controles (tales como botones de opción) en un marco con etiqueta.
	Panel	pnl	Agrupar un conjunto de controles en un marco sin etiqueta que permite el desplazamiento.

Tabla 1: Descripción de algunos controles más utilizados de la “barra de controles”

El **prefijo** es una combinación de letras que se le sugiere al programador colocar al inicio del identificador (nombre) del control agregado, de acuerdo al tipo del mismo. Con este prefijo, se pretende que al momento de dar lectura a un código con muchas referencias a objetos utilizados, sea más fácil de entender y comprender.

Estructura de un programa en C#

C# es bastante sencillo en cuanto a su sintaxis. Alguien con experiencia en programación con lenguaje C/C++, Java o Perl, no debería tener ningún problema de adaptación. Sin embargo, posee cierto tipo de dificultades que se superan en la medida se profundiza en la programación del mismo. A continuación mostraremos los tópicos relacionados a la sintaxis de C#.

Clase estándar

Un módulo agrupa código fuente de una aplicación y se almacena en un archivo .vb. Para definir un módulo se usa la sintaxis siguiente:

```
namespace Nombre_Aplicacion
{
    public partial class Nombre_Formulario : Form
    {
        public Form1()
        {
```

```

        [SENTENCIAS]
    }
}

```

Procedimientos

Un procedimiento es un sub-programa o rutina al que se le asigna un cierto nombre y que realiza unas determinadas operaciones. La ventaja que tiene utilizar procedimientos es que su código se escribe una sola vez pero pueden ser ejecutados desde tantos puntos del script como se desee.

Un procedimiento puede recibir una serie de parámetros que variarán su funcionamiento. Los parámetros a recibir (de qué tipo es cada uno y para qué se utiliza) se determinan al escribir el procedimiento.

Dentro del código que determina el funcionamiento de un procedimiento se pueden a su vez ejecutar otros procedimientos, incluso sí mismo de forma recursiva.

Funciones:

Una función es un procedimiento que, adicionalmente, devuelve un valor como resultado de su ejecución. La ventaja de las funciones es que pueden utilizarse en asignaciones, expresiones matemáticas, etc.

Sintaxis:

Declaración y programación (escritura) de un procedimiento:

```

public int Maximo (int x, int y) {

    int mayor;

    if (x==y)

        return -1;

    else

        if (x>y)

            return x;

        else

            return y;

}

```

En la primera parte de la declaración

public int Maximo (int x, int y)

public, es el nivel de acceso, indica que puede ser accedida desde cualquier scrip, dentro o fuera del archivo donde se declaró.

int, es el valor que retornará la función, en este caso como es un numero el que tiene que retornar, pero no siempre es así, algunas veces, hay funciones que piden de argumento un string y retornan un bool u otro tipo de datos.

Maximo, es el nombre de la función

(**int x**, **int y**) son los argumentos de la función, los valores que necesita para operar, en este caso son dos, siempre se separan con una coma y se debe de definir el nombre y el tipo de datos.

Y dentro de esta función utiliza los argumentos que pide y realiza la comparación, lo de adentro es un simple if...

Un evento es un mensaje que envía un objeto cuando ocurre una acción. La acción podría ser causada por la interacción del usuario, como un clic del botón, o podría ser iniciado por otra lógica de programa, como cambiar un valor de propiedad. El objeto que provoca el evento se conoce como *emisor del evento*. El emisor del evento no sabe qué objeto o método recibirá (controlará) los eventos que genera. El evento normalmente es un miembro del emisor del evento; por ejemplo, el evento Click es un miembro de la clase Button, y el evento PropertyChanged es un miembro de la clase que implementa la interfaz INotifyPropertyChanged.

Para definir un evento, se utiliza la palabra clave **event** (en C#) o **Event** (en Visual Basic) en la signature de la clase de eventos y se especifica el tipo de delegado para el evento. Los delegados se describen en la sección siguiente.

Normalmente, para generar un evento, se agrega un método marcado como **protected** y **virtual** (en C#) o **Protected** y **Overridable** (en Visual Basic). Asigne a este método el nombre **OnEventName**; por ejemplo, **OnDataReceived**. El método debe tomar un parámetro que especifica un objeto de datos de evento. Este método se proporciona para permitir que las clases derivadas reemplacen la lógica para generar el evento. Una clase derivada siempre debería llamar al método **OnEventName** de la clase base para asegurarse de que los delegados registrados reciben el evento.

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("HOLA MUNDO");
}
```

Manejo de Variables en C#.Net

Puntuadores de sentencias.

Los puntuadores son caracteres especiales que delimitan bloques de código, y el fin de una sentencia.

```
{ línea_de_codigo1; línea_de_codigo2, línea_de_codigon; ... }
```

Las llaves agrupan múltiples sentencias dentro de un bloque de sentencias. Las sentencias pueden ocupar una o varias líneas. Esto último resulta útil para asuntos de legibilidad del código fuente. El terminador o delimitador de sentencias en C# es el punto y coma (;), el mismo que se utiliza en C/C++. Por lo tanto, cuando desee terminar una sentencia para iniciar otra debe utilizar el punto y coma.

Comentarios.

C# utiliza los estilos de comentarios del lenguaje C/C++ y del *shell* de la interfaz de comandos de Unix y Linux. Estos son el comentario de una sola línea `//` y el comentario de bloque `/* ... */`, también utilizados en el lenguaje C.

Veamos algunos ejemplos:

1. Comentarios de línea simple: Por ejemplo

```
int x=3;//Asigna 3 a x, es útil para documentar una línea simple de código.
```
2. Comentarios de múltiples líneas:

```
int x = 12 * 30; /* Este comentario
se expande en dos líneas */
```

Tipos de datos disponibles

C# contiene dos categorías generales de tipos de datos integrados: tipos de valor y tipos de referencia. El término tipo de valor indica que esos tipos contienen directamente sus valores.

Tipo de datos de enteros				
Tipo	Equivalente BCL	Tamaño	Rango	Significado
byte	<code>System.Byte</code>	8-bit (1-byte)	0 a 255	Entero sin signo
sbyte	<code>System.SByte</code>	8-bit (1-byte)	-128 a 127	Entero con signo
short	<code>System.Int16</code>	16-bit (2-byte)	-32.768 a 32.767	Entero corto con signo
ushort	<code>System.UInt16</code>	16-bit (2-byte)	0 a 65.535	Entero corto sin signo
int	<code>System.Int32</code>	32-bit (4-byte)	-2.147.483.648 a 2.147.483.647	Entero medio con signo
uint	<code>System.UInt32</code>	32-bit (4-byte)	0 a 4.294.967.295	Entero medio sin signo
long	<code>System.Int64</code>	64-bit (8-byte)	-9.223.372.036.854.775.808 9.223.372.036.854.775.807	a Entero largo con signo
ulong	<code>System.UInt64</code>	64-bit (8-byte)	0 a 18.446.744.073.709.551.615	Entero largo sin signo

Los tipos de coma flotante pueden representar números con componentes fraccionales. Existen dos clases de tipos de coma flotante: `float` y `double`. El tipo `double` es el más utilizado porque muchas funciones matemáticas de la biblioteca de clases de C# usan valores `double`. Quizá, el tipo de coma flotante más interesante de C# es `decimal`, dirigido al uso de cálculos monetarios. La aritmética de coma flotante normal está sujeta a una variedad de errores de redondeo cuando se aplica a valores decimales. El tipo `decimal` elimina estos errores y puede representar hasta 28 lugares decimales.

Tipo de datos de coma flotante				
Tipo	Equivalente BCL	Tamaño	Rango	Significado
float	<code>System.Single</code>	32-bit (4-byte)	±1.401298E-45 a ±3.402823E+38	Coma flotante corto
double	<code>System.Double</code>	64-bit (8-byte)	±4.94065645841246E-324 ±1.79769313486232E+308	a Coma flotante largo
decimal	<code>System.Decimal</code>	128-bit (16-byte)	(16-a -7.9228162514264337593543950335	Coma flotante monetario

Los caracteres en C# no tienen un tamaño de 8 bits como en otros muchos lenguajes de programación, sino que usa un tamaño de 16 bits llamado Unicode al cual se le llama *char*. No existen conversiones automáticas de tipo entero a *char*.

Tipo de datos de caracteres				
Tipo	Equivalente BCL	Tamaño	Rango	Significado
char	<code>System.Char</code>	16-bit (2-byte)	'\u0000' a '\uFFFF'	Carácter unicode

Para los tipos de datos lógicos no existen conversiones automáticas de tipo entero a *bool*.

Tipo de datos lógicos				
Tipo	Equivalente BCL	Tamaño	Rango	Significado
bool	<code>System.Boolean</code>	8-bit (1-byte)	true o false	Verdadero o falso

Declaración de Variables

Las variables son *identificadores* asociados a valores. Se declaran indicando el tipo de dato que almacenará y su identificador. *Una variable representa un valor numérico o de cadena o un objeto de una clase*. El valor que la variable almacena puede cambiar, pero el nombre sigue siendo el mismo. Una variable es un tipo de campo. El código siguiente es un ejemplo sencillo de cómo declarar una variable de entero, asignarle un valor y, a continuación, asignarle un nuevo valor.

```
<tipo_de_dato> nombre_variable [= valor];
int x = 1; // x almacena el valor de 1
x = 2;    // ahora, x almacena el valor de 2
```

En C#, las variables se declaran con un tipo de datos y una etiqueta concretos como se ha visto en el ejemplo anterior. Si hasta ahora sólo ha utilizado lenguajes con tipos definidos de forma imprecisa como JavaScript, estará acostumbrado a emplear el mismo tipo "var" para todas las variables, pero en C# se tiene que especificar si la variable es de tipo int, float, byte, short u otro cualquiera entre más de 20 tipos de datos diferentes. El tipo especifica, entre otras cosas, *la cantidad de memoria exacta* que se debe asignar para almacenar el valor cuando la aplicación se ejecuta.

Un identificador puede:

- ✧ Empezar por "_".
- ✧ Contener caracteres Unicode en mayúsculas y minúsculas (sensible a mayúsculas y minúsculas).

Un identificador no puede:

- ✧ Empezar por un número.
- ✧ Empezar por un símbolo, ni aunque sea una palabra clave.
- ✧ Contener más de 511 caracteres.

Constantes.

Las constantes son valores inmutables, y por tanto no se pueden cambiar. Cuando se declara una constante con la palabra clave *const*, también se debe asignar el valor. Tras esto, la constante queda bloqueada y no se puede cambiar. Son implícitamente estáticas (*static*).

```
const double PI = 3.1415;
```


Otra forma alternativa para la manipulación de constantes dentro de C# es haciendo uso de miembros de solo lectura, que a diferencia de *const*, no requiere que se asigne el valor al mismo tiempo que se declara. Pueden ser miembros de la instancia o miembros estáticos de la clase (static).

```
readonly double E;
E = 2.71828;
```

Tratamiento de cadenas

- ✧ El tipo de dato para las cadenas de caracteres es string.
- ✧ Realmente la palabra clave *string* es un alias de la clase System.String de la plataforma .NET.
- ✧ En C# las cadenas son objetos y no una matriz de caracteres; aun así, se puede obtener un carácter arbitrario de una cadena por medio de su índice (pero no modificarlo).
- ✧ Las cadenas son inmutables, una vez creadas no se pueden modificar, solo se pueden copiar total o parcialmente.
- ✧ El operador == determina si dos referencias hacen referencia al mismo objeto, pero al usar dicho operador con dos variables tipo string se prueba la igualdad del contenido de las cadenas y no su referencia. Sin embargo, con el resto de los operadores relacionales, como < y >=, sí se comparan las referencias.
- ✧ Se pueden concatenar (unir) dos cadenas mediante el operador +.
- ✧ Las cadenas se pueden usar en las instrucciones switch.

1. Declarar una cadena de caracteres (como si fuera una variable de un tipo de dato como int o double):

```
string texto = "Cadena de caracteres";
string texto = new System.String("Cadena de caracteres"); // Equivalente al anterior
```

2. Longitud de una cadena:

```
string texto = "Cadena de caracteres";
int i = texto.Length; // Retornará '20'
```

3. Comparar dos cadenas:

```
bool b = "texto" == "texto"; // Retornará 'true', ya que ambas cadenas contienen "texto"
```

4. Concatenar cadenas:

```
string texto = "Cadena de" + " caracteres"; // Nótese el espacio antes de "caracteres", si no se pusiera, aparecería junto: "decaracteres"
```

Códigos o secuencias de escape

En C#, al igual que en otros lenguajes existen ciertos caracteres que generan problemas cuando se encuentran dentro de cadenas, debido a que dentro del lenguaje se interpretan de alguna forma especial. Para poder visualizar correctamente dichos caracteres en una operación de salida se utilizan secuencias de escape que involucran dichos caracteres especiales. La siguiente tabla muestra varios de estos caracteres especiales con su correspondiente secuencia de escape:

Secuencia de escape	Significado
\b	Espacio hacia atrás (<i>backspace</i>)
\a	Alerta (timbre)
\f	Cambio de página (<i>form feed</i>)

\n	Cambio de línea (<i>line feed</i>)
\r	Retorno de carro (<i>carriage return</i>)
\t	Tabulación horizontal
\v	Tabulación vertical
\\	Barra inversa (<i>backslash</i>)
\'	Comilla simple
\"	Comilla doble
\0	Valor nulo

Expresiones en C#: Prioridad en los Operadores

Los operadores son símbolos especiales que se utilizan en los lenguajes de programación para poder realizar operaciones con las expresiones. Como lo que se obtiene en dichas operaciones es un valor, el resultado también será una expresión.

Los operadores se pueden clasificar como: unarios (que operan sobre un único valor o expresión), binarios (que operan sobre dos valores o expresiones) y ternarios (que consta de tres valores o expresiones)

C# proporciona un amplio conjunto de operadores, que son símbolos que especifican las operaciones que se deben realizar en una expresión. Operaciones con tipos enteros como ==, !=, <, >, <=, >=, binary +, binary -, ^, &, |, ~, ++, -- y sizeof() son generalmente permitidas en enumeraciones. Además, el usuario puede sobrecargar muchos de los operadores, es decir, cambiar su significado al aplicarlos a un tipo definido por el usuario.

En la tabla siguiente se muestran los operadores de C# agrupados por orden de prioridad. Los operadores dentro de cada de grupo tienen la misma prioridad.

Categorías	Operadores	Uso
Primario	<u>x.y</u>	Acceso a miembros, sea propiedad o método
	<u>f(x)</u>	Orden de operaciones en una expresión, especificar conversión de tipos
	<u>a[x]</u>	Utilizados en matrices, indizadores y atributos
	<u>x++</u>	Incremento postfijo en 1, devuelve el valor de x
	<u>x--</u>	Decremento postfijo en 1, devuelve el valor de x
	<u>new</u>	Creación de nuevos objetos e invocación de constructores
	<u>typeof</u>	Obtiene el tipo de dato asociado
	<u>checked</u>	Habilitar explícitamente la comprobación de desbordamiento para operaciones aritméticas y conversiones de tipo integral.
	<u>unchecked</u>	Suprimir la comprobación de desbordamiento para operaciones aritméticas y conversiones de tipo integral.
	<u>sizeof</u>	Se usa para obtener el tamaño en bytes de un tipo no administrado
Unario	<u>-></u>	El operador -> combina la des referenciación de un puntero y el acceso a un miembro.
	<u>+x</u>	Suma de operandos y en el caso de cadenas su concatenación
	<u>-x</u>	Resta de operandos
	<u>!x</u>	Negación del operando utilizado con valores de tipo bool
	<u>++x</u>	Incremento prefijo en 1, devuelve el valor de x+1
	<u>--x</u>	Decremento prefijo en 1, devuelve el valor de x-1
	<u>&x</u>	Devuelve la dirección de memoria de su operando

	<u>*x</u>	Operador de desferenciación que permite leer y escribir en un puntero
Multiplicativo	<u>x * y</u>	Multiplicación de operandos
	<u>x / y</u>	División de operandos
	<u>x % y</u>	Residuo de dividir x entre y
Sumatorio	<u>x + y</u>	Suma de operandos
	<u>x - y</u>	Resta de operandos
Desplazamiento	<u>x << y</u>	Desplaza x a la izquierda y cantidad de bits
	<u>x >> y</u>	Desplaza x a la derecha y cantidad de bits
Comprobación de tipos y relacionales	<u>x < y</u>	Devuelve TRUE si x es menor que y, FALSE caso contrario
	<u>x > y</u>	Devuelve TRUE si x es mayor que y, FALSE caso contrario
	<u>x <= y</u>	Devuelve TRUE si x es menor o igual que y, FALSE caso contrario
	<u>x >= y</u>	Devuelve TRUE si x es mayor o igual que y, FALSE caso contrario
Igualdad	<u>is</u>	Devuelve TRUE si el objeto es compatible con cierto tipo
	<u>x == y</u>	Devuelve TRUE si x es igual que y, FALSE caso contrario
	<u>x != y</u>	Devuelve TRUE si x es diferente que y, FALSE caso contrario
AND condicional	<u>x && y</u>	Devuelve TRUE si x es TRUE y y es TRUE, FALSE caso contrario
OR condicional	<u>x y</u>	Devuelve TRUE si x es TRUE o y es TRUE, FALSE caso contrario
Condicional	<u>?:</u>	Devuelve uno de los valores dependiendo del resultado de la expresión booleana: condition ? first_expression_true : second_expression_false;
Expresión de asignación y lambda	<u>x = y</u>	Asigna a x el valor de y
	<u>x += y</u>	Actualiza el valor de x con la suma de x con y
	<u>x -= y</u>	Actualiza el valor de x con la resta de x con y
	<u>x *= y</u>	Actualiza el valor de x con el producto de x por y
	<u>x /= y</u>	Actualiza el valor de x con la división de x por y
	<u>x %= y</u>	Actualiza el valor de x con el residuo o módulo de x entre y

Para mayor referencia sobre otros operadores y operadores adicionales visite: <http://msdn.microsoft.com/es-sv/library/6a71f45d.aspx>

FUNCIONES DE CONVERSION DE TIPOS.

Supongamos que dejamos que un usuario introduzca un valor numérico por pantalla y queremos realizar cálculos con ese valor. Por lo tanto, el valor introducido por el usuario será del tipo *System.String* para poder realizar conversiones en visual C# podemos convertirlo de dos formas, por ejemplo si queremos pasarlo a *System.Int32*.

```
//sintaxis: Convert.Tipo_dato_a_convertir(variable_a_convertir);
int valorA = Convert.ToInt32(valorUsuario);
```

```
//sintaxis: Tipo_dato_a_convertir.Parse(Valor_a_convertir);
int valorB = Int32.Parse(valorUsuario);
```

Cuando utilizamos la segunda opción, el método *Parse()*, si el usuario no ha introducido ningún valor (*null*) recibiremos una excepción del tipo *System.FormatException*. Lo que indica que el formato del argumento no cumple las especificaciones del parámetro del método invocado.

En cambio, cuando utilizamos la primera opción, los métodos de la clase *System.Convert*, si el valor pasado al método es *null*, el método *Convert.ToInt32()* devolverá un valor numérico, devolverá cero en todos los casos que el valor de entrada sea *null*.

SENTENCIAS DE CONTROL CONDICIONAL

En C# se dispone de las siguientes estructuras de tomas de decisiones:

If .. [Else]

Switch

Ambas estructuras son equivalentes en su ejecución a las estructuras if-else y switch de lenguaje C, respectivamente. Observe la sintaxis de uso en Tabla 4.

Estructura If-Then-[Else]	Estructura Switch
<p>✪ if-else</p> <pre> if (i == 2) { // ... } else if (i == 3) { // ... } else { // ... } </pre>	<pre> switch (i) { case 1: ... break; case 2: case 3: ... break; default: ... break; } </pre>

Tabla 4: sintaxis generales de estructuras repetitivas de C#

III. MATERIALES Y EQUIPO

Para la realización de la guía de práctica se requerirá lo siguiente:

No.	Requerimiento	Cantidad
1	Guía de Laboratorio #1	1
	PC con Microsoft Visual Studio 2012 .NET instalado	1
2	Memoria USB	1
3	Computadora con acceso a Internet	1

IV. PROCEDIMIENTO

Para esta práctica deberá crear una carpeta con el nombre de "Práctica1LP1", en la cual va a guardar todos los archivos de esta práctica y de análisis de resultados.

PARTE 1: Creando una Aplicación Windows

- Pasos para la creación de aplicaciones en Visual C#

1. En el menú Inicio de Windows, localice a Microsoft Visual C# 2012 dentro del listado de accesos de programas instalados
2. Aparecerá la pantalla de bienvenida que es la interfaz para Visual Studio, también conocida como Entorno de Desarrollo Integrado o IDE.
3. Haga clic en el menú Archivo y luego sobre opción Nuevo Proyecto...
4. Aparece el cuadro de diálogo Nuevo proyecto (Ver Figura 1.1).
5. De las plantillas recientes (ver columna a la izquierda), seleccione Visual C# y luego su plantilla "Windows". En la parte central, de clic sobre opción "Aplicación de Windows Forms"

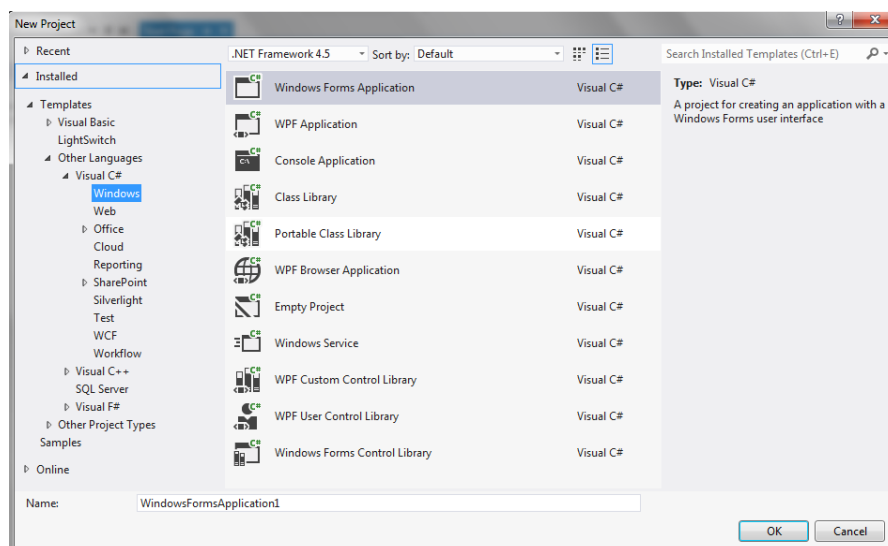


Figura 1.1: Selección de tipos de Proyectos en Visual C# .NET

6. En la parte inferior, en la opción Nombre, modifíquelo con el nombre **Ejemplo1**, el cual será el nombre de nuestro proyecto. Haga clic en Aceptar.
 7. Ahora proceda a incluir en el area de Diseño del Form, a c/u de los controles mostrados en la Figura 1.2 Apóyese en la imagen 1.1 (de la introducción teórica), localice el panel Cuadro de herramientas y determine el tipo control a utilizar.
 8. Hay 2 maneras de agregar los controles deseados al formulario:
 - a) Dar clic sobre control de la barra herramientas y dar un clic en el area del form que usara ese control.
 - b) Dar clic sobre el control en barra herramientas y luego, colocar cursor del ratón sobre area del form, manteniendo presionado botón principal.
- Establecer las propiedades de los Controles
 9. Las propiedades de un Control, cambian la apariencia y/o el funcionamiento del mismo ante el usuario que usa el form. Una propiedad en Visual C# representa un atributo de un objeto, en este caso, un Control. Por ejemplo:
 - ✓ Uno de los atributos de un control Button es el texto que este muestra. En este caso, se puede modificar al asignar la propiedad **Text**. La mayoría de éstas modificaciones también se pueden establecer o modificar dentro del código del programa
 - ✓ Las propiedades pueden tomar muchos tipos diferentes de valores además del texto.
 - ✓ Si se cambia el tamaño de un control o se reubica, también se actualizan las propiedades **Size** y **Location** que determinan el tamaño y la ubicación de un control en el formulario.

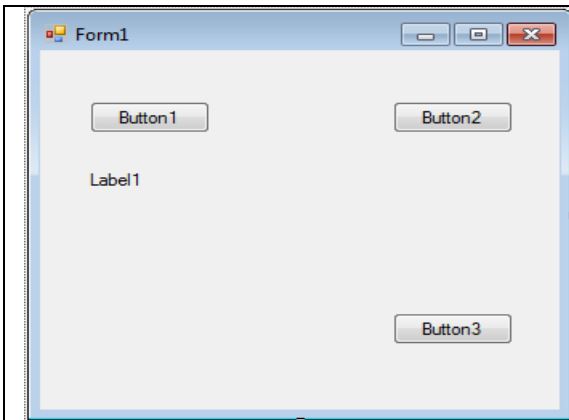


Figura 1.2: Vista Formulario del Ejemplo 1 y los controles requeridos.

CONTROL	PROPIEDAD	Nuevo VALOR
Form1	Text	Pruebas de ámbito de variables
	BackColor	Un color Personalizado a su gusto
Button1	Name	btnContar
	Text	Contar
Button2	Name	btnReinicio
	Text	Reiniciar conteo
Button3	Name	btnFin
	Text	Salir programa

10. En la tabla de la Figura 1.2 se detalla a los diferentes controles colocados en el Form1 y las propiedades (y su nuevo valor) que deberá modificar antes de continuar. Observe uso de prefijos en el identificador de c/control (que es definido en propiedad Name de c/control).

- Agregar el código y la funcionalidad al Programa

11. Para agregar código a un determinado control se debe de hacer doble clic sobre el. De doble clic sobre un area del control Form1. Se abrirá una nueva ventana denominada Editor de código, como se muestra en la Figura 1.3.

```
namespace Ejemplo1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

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

Figura 1.3: Editor de código de Visual Studio .NET

12. En la Figura 1.3, el Editor de código muestra primero el de nuestra clase Form1, el cual contendrá a su vez todo el código utilizado para programar a los objetos insertados en el mismo.

Luego se muestra el encabezado-final del procedimiento de evento Load del objeto formulario Form1, asi:


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

13. Este código es un *Controlador de evento*, también denominado *procedimiento manejado por evento*.

Cualquier código dentro de este procedimiento se ejecuta cada vez que suceda el evento programado para ese control.

14. Ejecute el programa creado hasta aquí. Lo puede hacer de tres formas diferentes:

- Presionando la tecla F5
- Eligiendo en el menú Depurar -> Iniciar Depuración

- Haciendo clic en la Barra Estándar en el 1er icono del siguiente trio  de la barra de botones Estándar.

15. Finalice la ejecución del programa en diseño, seleccionando el 3er boton del trio mencionado en paso anterior.

PARTE 2: Declaración de variables y Ambito/Alcances de las mismas

16. Retorne a la ventana de código del form1. Desplace el cursor en cada línea de código y observe que el listados superiores de ubicación de código cambian de acuerdo a donde se localice el cursor.
17. Ahora redacte cada bloque de código a continuación. Para ello, seleccione c/objeto y el Evento/Nivel a programar (de los cuadros de selección objeto-eventos superiores) indicados.

(*) Por cada selección cuide la ubicación del cursor, porque C# le ayuda a colocarlo al inicio del bloque donde usted ha ubicado a las opciones de objetos y eventos para programar!!

Objeto	y	Evento/Nivel
Form1	(Declaraciones variables globales)	
<ul style="list-style-type: none"> • Declarar variable: mensa, tipo string de alcance privado. • Declarar variable: conta, tipo int de alcance privado y poner como comentario esta línea. 		
(Form1 eventos)	Load	
<ul style="list-style-type: none"> • Agregar el siguiente comentario: Evento se ejecuta al iniciar ejecución de Form1 • Asignarle a mensa la siguiente cadena: "Aun no ha presionado botón Contar" • Hacer que label1 muestre el contenido de mensa 		
btnContar	Click	
<ul style="list-style-type: none"> • Digitar el siguiente código: <pre>int conta = 0; //Acumulador, Total veces que presiona botón conta = conta + 1; //conta+=1; //En forma de operador abreviado</pre> • Asignarle a mensa la siguiente cadena: "Presiono botón Contar, un total de " + Convert.ToString(conta)+" veces"; //convert me pase mi variable conta de int a string • Hacer que label1 muestre el contenido de mensa 		
btnReinicio	Click	
<ul style="list-style-type: none"> • Digitar el siguiente código: <pre>//Restaura conteo clic realizado a button1 //conta =0; //mensa = "Presiono boton Contar, un total de" + Convert.ToString(conta) + " veces";</pre> • Hacer que label1 muestre el contenido de mensa 		
BtnFin	Click	
<ul style="list-style-type: none"> • Digitar el siguiente código: <pre>Close();//finaliza aplicación</pre> 		

18. Ejecute nuevamente la aplicación, observe el mensaje en la Label1. De clic sobre el botón Contar. Observe que variable conta cuenta el total de clic hechos al control por usuario.
19. De clic varias veces en botón Contar. Vera que no funciona el conteo hecho por la variable **conta**, siempre muestra 1 vez ha hecho clic.
20. De clic en boton Salir programa. El programa finaliza su ejecución y retorna a la etapa de diseño.
21. Localice el código del evento Clic del control **btnContar** y comente la declaración de variable **conta**.
22. Localice las variables globales y des comente la declaración de variable **conta** y agregué que será una variable estática.

23. Finalice ejecución y regrese a la ventana de Diseño del Form1. Ubique el cursor en el código del evento Click del botón btnReinicio y reactive (quitando comilla simple de comentario) a las asignaciones a las variables conta y mensa ejecutadas ahí.
24. Vuelva a ejecutar el programa y en algún momento presione botón btnReinicio. El valor de variable conta será asignado a cero
25. Guarde su proyecto, presionando opción Archivo/Guardar Todo... Observe la ubicación (carpeta de proyectos de Visual Studio) predetermina que ofrece Framework. Guarde su proyecto en carpeta mis documentos del usuario con el cual ha iniciado sesión en su PC.
26. Finalmente localice y presione opción Archivo/Cerrar Proyecto

PARTE 3: Tomas de decisiones (If Then Else)

Ahora procederá a solucionar este problema:

“Una administradora de prestamos requiere ingresar los datos de un prestamo (Monto a prestar, Tasa interes mensual y total de años (a 2 años inicialmente)) que se entregara a una Empresa ingresada. Determine el monto final que se pagara”.

1. En la opción StartPage. Localice el nombre del proyecto (archivo Ejemplo1.csproj) utilizado hasta ahora que se encuentra en la opción del Recent(Recientes) del mismo y dar clic para volver abrir el proyecto.
2. En la ventana Explorador de Solución, ubique el nombre del proyecto actual y de clic secundario sobre el mismo. Del menú emergente seleccione opción **(Add)Agregar**, luego **Windows Forms...**
3. En la ventana “Agregar nuevo elemento- Ejemplo1”, observe que en la parte central ya esta seleccionado “Windows Forms”. Confirme en última opción (Nombre:) que el nombre del archivo sea **(Form2.vb)** y presione Agregar. Observe que se muestra un nuevo form vacio y también, desde el explorador de solución, se agrega un elemento mas (Form2.vb) al proyecto actual.
4. Luego, desde el explorador de soluciones de clic Programs.cs, Ahí sustituya el Form1 por Form2
5. Guarde los cambios del proyecto actual.
6. Luego aplique al Form2 el diseño definido en Figura 1.4.

Figura 1.4: Diseño de controles del Form2

7. Con mucho cuidado, seleccione a cada control mostrado en la tabla a continuación, para luego modificar a c/propiedad solicitada ahí con el valor indicado.

	(propiedades)			
(controles)	Name	Text	Enabled	Checked
Form2		Calculo del Monto a pagar por un Prestamo		
Label1	lblEmpresa	Nombre de Empresa:		
Label2	lblMonto	Monto prestamo (\$)		
Label3	lblTiempo	Tiempo (años)		
Label4	lblTasaInter	Tasa Interes:		
RadioButton1	rdbInteres1	12 %		True
RadioButton2	rdbInteres2	23.5 %		
RadioButton3	rdbInteres3	Otro, indique:		
TextBox1	txtEmpresa	(vacío)		
TextBox2	txtMonto	(vacío)		
TextBox3	txtTiempo	2		
TextBox4	txtTasaInterEX	0	False	
Button1	btnAnalisis	Análisis Financiero		
Button2	btnFin	Salir del programa		
ListBox1	lstResul		False	

* **Extra:** Para el cuadro lstResul, seleccione además la propiedad Items, y presione en Colecciones.. que se indica ahí, para luego agregar valor: (Resultados). Esto agregara este texto como 1er elemento del control lstResul

8. Realice la siguiente asignación de código, de acuerdo al objeto y evento indicados a continuación.

Objeto y Evento/Nivel	
Form2	(Declaraciones globales)
<ul style="list-style-type: none"> Declarar variable: TasaI, tipo double de alcance privado. Digitar el siguiente código: <pre>//valida que el dato recibido es un numero public static Boolean IsNumeric(string valor) { int result; return int.TryParse(valor, out result); }</pre> 	
rdbInteres3	CheckedChanged
<ul style="list-style-type: none"> Digitar el siguiente código: <pre>if (rdbInteres3.Checked == true) { txtTasaInterEX.Enabled = true; txtTasaInterEX.Focus(); } else { txtTasaInterEX.Text = "0"; txtTasaInterEX.Enabled = false; }</pre> 	

}		
	rdbInteres1	Click
<ul style="list-style-type: none"> Asignar a TasaI el valor de 0.12 		
	rdbInteres2	Click
<ul style="list-style-type: none"> Asignar a TasaI el valor de 0.235 		
	btnFin	Click
<ul style="list-style-type: none"> Instrucción para finalizar aplicación. 		
	btnAnalisis	Click
<pre> //Declaracion de variables a utilizar string NomEmpre; double MontoInic=0, MontoFin=0; int Tiempo; NomEmpre = txtEmpresa.Text; NomEmpre = NomEmpre.Trim();//Quita el espacio-blanco al inicio-final if (NomEmpre.Length == 0) { //si no hay caracteres en nombre empresa MessageBox.Show("Debe indicar Nombre de la empresa", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Information); txtMonto.Focus(); //metodo que indica que control txtEmpresa recibira cursor return;//sale del procedimiento btnanalisis } if (!(IsNumeric(txtMonto.Text))) { MessageBox.Show("Valor Monto incorrecto", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error); txtMonto.Focus(); //metodo que indica que control txtEmpresa recibira cursor return; } else { MontoInic = Convert.ToDouble(txtMonto.Text); if (!(MontoInic >0)){ MessageBox.Show("Valor Monto no puede ser negativo", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error); txtMonto.Focus(); //metodo que indica que control txtEmpresa recibira cursor return; } } Tiempo = Convert.ToInt32(txtTiempo.Text); //si selcciono Tasa interes 3, valida que sea correcta txtTasaInterEX.Text = txtTasaInterEX.Text.Trim(); if (rdbInteres3.Checked == true) { if (txtTasaInterEX.Text.Length > 0){ if (!(IsNumeric(txtTasaInterEX.Text) == true)) { MessageBox.Show("Tasa interes incorrecto", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Error); txtTasaInterEX.Text = "0"; txtTasaInterEX.Focus(); return; } } else { </pre>		

```

        TasaI = Convert.ToDouble(txtTasaInterEX.Text) / 100;
    }
}

else
{
    MessageBox.Show("Aun no ha indicado una tasa interes", "ERROR",
    MessageBoxButtons.OK, MessageBoxIcon.Information);
    txtTasaInterEX.Focus();
    return;
}
}

//Hace el cálculo esperado
MontoFin =(1+TasaI);
MontoFin = MontoInic*(Math.Pow(Convert.ToDouble(MontoFin), Tiempo));
TasaI *= 100;
//Muestra la respuesta (Monto a pagar)
lstResul.Items.Clear();
lstResul.Items.Add("Empresa: " + txtEmpresa.Text);
lstResul.Items.Add("Monto: $" + MontoInic + ", Tasa anual: " + TasaI);
lstResul.Items.Add("Monto a pagar: $" + MontoFin);

```

9. Guarde los cambios del proyecto actual y ejecute la aplicación. Para ver las validaciones de entradas programadas, equivóquese a propósito en varios de los datos ingresados y presione boton btnAnalisis

PARTE 4: Tomas de decisiones (Switch)

Problema a solucionar: “Solicite cada apellido y los nombres de un estudiante, así como el valor del CUM obtenido de su último ciclo de estudios. Determine el total de unidades valorativas que un estudiante puede llevar en el próximo ciclo de estudios según el valor del CUM ingresado. Cuando el CUM es mayor a 7.5, puede llevar 32 UV máximo, si tiene entre 7 hasta 7.5, llevaría 24 UV Max, si tiene CUM entre 6 a 6.9, cursaría 20 UV. Y finalmente de 1 a 5.9, serán 16 unidades valorativas. En cualquier otro caso cursara 0 UV”.

1. Abra nuevamente el proyecto creado en la parte 1, y agregue un nuevo formulario.
2. Luego diseñe el siguiente esquema de objetos (mostrado en Figura 1.5) en este nuevo form (Form3)

Figura 1.5: Diseño de controles del Form3

	(Propiedades)		
(controles)	Name	Text	Enabled
Form3		Calculo de UV a cursar	
Label1	lblApe1	1er Apellido:	
Label2	lblApe2	2do Apellido:	
Label3	lblNom	Nombres	
Label4	lblCUM	Valor de CUM	
TextBox1	txtApe1	(vacio)	
TextBox2	txtApe2	(vacio)	
TextBox3	txtNom	(vacio)	

TextBox4	txtCUM	0.0	
TextBox5	txtResul	Resultado de evaluación	False
Button1	btnAnalisis	Calcular UV	
Button2	btnFin	Salir del programa	

3. Proceda a digitar el código detallado a continuación, en cada evento y objeto especificado.

Objeto y Evento/Nivel código a incluir

Form3 (Declaraciones globales)

- Declarar variable: **noms,ape1,ape2** tipo string de alcance privado.
- Declarar variable: **CUM** tipo double de alcance privado.
- Declarar variable: **UV** tipo int de alcance privado.
- Digitar el siguiente código:

```
private string noms, ape1, ape2;
private double CUM;
private int UV;
//valida que el dato recibido es un numero
public static Boolean IsNumeric(string valor)
{
    int result;
    return int.TryParse(valor, out result);
}
//evalua el cum
private void EvaluarCUM() {
    //una vez recibidos los nombres del estudiante, asi como su CUM se detemina las UV'
solicitadas

    string nombrecompleto;
    nombrecompleto = noms + " " + ape1 + " " + ape2;

    nombrecompleto = nombrecompleto.ToUpper() ;

    if (CUM < 0 | CUM > 10)
    {
        MessageBox.Show("Valor de CUM fuera de rango (0.0 - 10.0)", "Error",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
        txtCUM.Focus();
    }
    else {
        //Usa estructura switch

        switch (Convert.ToInt16(CUM))
        {
            case 8 - 10:
                UV = 32;
                break;
            case 7:
                UV = 24;
                break;
```

```

        case 6:
            UV = 20;
            break;
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
            UV = 16;
            break;
        default:
            UV = 0;
            break;
    }

    txtResul.Text = nombrecompleto + "Puede cursar " + UV + "UV";
}
}

```

btnAnalisis	Click
<ul style="list-style-type: none"> Realizar las respectivas validaciones vistas en el ejercicio 2. Llamar a EvaluarCUM, donde usted crea conveniente. 	
btnFin	Click
<ul style="list-style-type: none"> Instrucción para finalizar aplicación. 	

- Guarde los cambios del proyecto actual y ejecute la aplicación. Para ver las validaciones de entradas programadas, equívóquese a propósito, ingrese el CUM, pero olvídense de los nombres, y otros casos.
- En evento Click del botón btnFin redacte un mensaje de despedida y realice el final de ejecución

V. ANALISIS DE RESULTADOS

PROBLEMAS A RESOLVER:

- Solicitar 10 números decimales al usuario, para luego mostrar uno de estos resultados (según lo indique usuario):
 - El número menor y el número Mayor de solamente los negativos del listado y el promedio de los positivos.
 - Mostar el valor de la Media de toda la serie.
- Ayude a un usuario a ingresar una fecha de manera correcta, en el formato: Mes, Día, Año. Debe utilizar controles Combobox, y estructuras select case en su código. Recuerde que existen años bisiestos.
- Permita el ingreso de 10 números decimales cuales quiera. Cada nuevo número ingresado se mostrara en una de 2 listas diferentes (una para valores positivos y en la segunda, los valores negativos).
 - Una vez finalice el ingreso, se mostrara en otro listado la solución a estas incógnitas:
 - ¿Cuál fue el menor de los números negativos?
 - ¿Promedio de los números positivos?

Importante:

+ En todos los ejercicios, documente los bloques y líneas de código de acuerdo a lo crea conveniente, para que instructor comprenda mejor su solución.

+ Finalmente, envíe la carpeta del proyecto resultante, de manera comprimida a donde el instructor(a) indique y en la fecha que lo indique.

VII. BIBLIOGRAFÍA

- Título: Aprenda ya Microsoft Visual C#.NET.

Autores: Sharp, John Jagger, Jon Coautor

Publisher: Madrid, España: McGraw-Hill, 2002.

Clasificación: Libro 005.362 S581 2002

- Título: C # Manual de Programación.

Autores: Luis Joyanes Aguilar y Matilde Fernández Azuela

Publisher: Madrid, España: McGraw-Hill, 2002

Clasificación: Libro 005.362 J88 2002