Apuntes de Fundamentos de Programación.

FRANCISCO RÍOS ACOSTA
Instituto Tecnológico de la Laguna
Blvd. Revolución y calzada Cuauhtémoc s/n
Colonia centro
Torreón, Coah; México
Contacto : friosam@prodigy.net.mx

4 Introducción a la programación.

Dentro de este tema veremos algunos conceptos y definiciones que nos ayudarán a situarnos en el entorno de aprendizaje de la escritura de programas de computadora. La clasificación del software en 2 tipos : software de sistemas y software de aplicaciones, nos ayudarán en cuanto a la dirección de nuestros esfuerzos al construir programas. Si deseamos dedicarnos a resolver problemas empresariales de tipo administrativo, control de procesos donde intervengan personas, materiales, dinero, reportes estadísticos, manejo de bases de datos, sistemas en el web, entre otros, claramente habremos seleccionado el dedicarnos a escribir programas de aplicaciones. Si nos vamos por el camino de escribir programas para procesos industriales, manejo y codificación-decodificación de video, programación de autómatas industriales, robótica, editores, compiladores, sistemas operativos, hojas de cálculo, procesadores de texto, entre otros, habremos tomado el sendero de la programación de sistemas.

Los conceptos de programa, programación y de lenguaje de programación, son requeridos con el fin de conocer de manera formal los objetos –programa y lenguaje de programación- y las metodologías –programación-, involucrados en el contexto de la escritura de programas de computadora. En otras palabras, cuando escribimos programas que serán ejecutados en una computadora, el conocimiento formal de programa, programación y lenguaje de programación nos ayudarán a posicionarnos de manera firme sobre una plataforma que nos permita orientar nuestros esfuerzos de manera clara, hacia la construcción del *programa* -¿qué vamos a efectuar?-, qué herramientas formales- *programación*- usaremos para el desarrollo del programa –análisis y diseño-, y qué *lenguaje de programación* seleccionaremos para efectuar la implementación del programa. El concepto de programación es aplicado en todas las etapas del desarrollo de un programa, incluyendo la implementación –codificación-, pruebas y mantenimiento.

El estudio de la definición de datos es de importancia fundamental, debido a que ellos representan la entrada y salida de un programa de computadora. ¿Cómo construímos programas sin este concepto?, sería una auténtica "burresada" –sic-. Hemos visto en la unidad I que en la orientación a objetos los datos de entrada a una computadora y los datos de salida, son objetos, de forma que al hablar de datos estaríamos hablando también de objetos. En realidad, los datos pueden tener diferentes "formas de vestir", en algunos casos serán enteros, cadenas, caracteres, números reales, en otras ocasiones serán objetos –datos cuya estructura es mas compleja-. También al estudiar el concepto de datos, veremos que se le asocia a un dato, su almacenamiento, su direccionamiento y representación en memoria, y su representación cuando es númerico, en diferentes notaciones como son la binaria y la hexadecimal.

Los programas manipulan los datos de entrada -usando instrucciones- con el fin de producir datos de salida. Este proceso de manejo de datos en algunas ocasiones requiere de ciertas operaciones aritméticas, relacionales y lógicas, que involucran a operandos, operadores y a expresiones. De aquí que veremos los diferentes operadores aritméticos, relacionales y lógicos, la asociatividad y tipo de estos operadores, la prioridad de ejecución, así como la construcción de expresiones y su evaluación donde involucren a los diferentes tipos de operadores.

Terminaremos explicando la estructura básica de un programa en el lenguaje de programación C# bajo ambiente Windows, y el proceso de creación de un programa ejecutable.

4.1 Clasificación del software.

4.1.1. *Software de sistemas*. Se le denomina también software de base. Son los programas que interactúan con el hardware de la computadora – memoria RAM y de disco, teclado, monitores, periféricos, puertos – como son los programas de sistemas operativos –Windows, Linux-, compiladores, cargadores, ligadores, ensambladores, intérpetes, entre otros. Se comportan también como soporte para otros programas que requieran de interactuar cualquier dispositivo de la computadora –hardware-.

4.1.2. *Software de aplicación*. se refiere a los programas que efectúan una tarea específica ya sea empresarial, científica, educativa, que procesan datos de entrada con el fin de generar información –datos de salida- que serán usados o útiles para un determinado usuario. Ejemplos de este tipo de software son los programas de facturación, de administración de personal, de nómina, software educativo, procesadores de textos, hojas de cálculo, entre otros.

4.2 Conceptos de programación.

4.2.1 ***Programa*****.** es un conjunto de instrucciones para una computadora para efectuar las tareas necesarias para lograr un fin específico. Un procesador que se ubica dentro de la computadora es el encargado de efectuar, interpretar, o ejecutar, dichas instrucciones.

Un programa puede encontrarse en 2 fases principales : (1) como programa fuente y (2) como programa ejecutable. Un programa fuente es un texto que contiene instrucciones escritas en un lenguaje de alto nivel, es decir, un lenguaje que es comprensible por los humanos –programadores-.

Cuando el programa fuente sufre una transformación llamada compilación, cambia a la fase de programa ejecutable. En realidad la compilación consiste de una traducción del programa fuente a un programa objeto. Este programa objeto generalmente es un programa cuyas instrucciones están codificadas en lenguaje ensamblador, propio del procesador de la computadora. Este programa en ensamblador es posteriormente traducido a lenguaje máquina absoluto, es decir, a un lenguaje binario –instrucciones formadas por únicamente 1's y 0's-.

Cuando se pide que el programa sea ejecutado, éste deberá estar en formato binario –lenguaje binario-, sólo entonces el procesador ejecuta el programa instrucción por instrucción, hasta que el programa termina.

La definición de programa va de la mano de la definición de algoritmo. Un algoritmo es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema. Un programa implementa a un algoritmo, es decir, lo traduce a un lenguaje de programación. El programa puede estar compuesto de instrucciones escritas en lenguaje natural, en nuestro caso el español, o bien en un lenguaje de alto nivel.

Otra manera en que se ejecuta un programa, es ejecutar las instrucciones conforme son encontradas, una a una. A este proceso se le llama interpretar y a los programas que lo hacen se les conoce como intérpretes.

4.2.2 ***Programación*****.** es un proceso por el cual se escribe en un lenguaje de programación, se prueba, se depura y se mantiene el código fuente de un programa.

4.2.3 ***Lenguaje de programación*****.** es un lenguaje que consiste en un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Un lenguaje de programación permite a uno o más programadores especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados, transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias.

Una característica relevante de los lenguajes de programación es precisamente que más de un programadores puedan tener un conjunto común de instrucciones que puedan ser comprendidas entre ellos para realizar la construcción del programa de forma colaborativa.

El lenguaje de programación que usaremos en nuestro curso para implementar programas de computadora, es el lenguaje C#. El lenguaje C# es un lenguaje que permite escribir programas orientados a objetos.

4.3 Datos.

4.3.1 ***Definición de datos***. es una representación simbólica (numérica, alfabética, booleana, entre otras), atributo o característica de una entidad. El dato no tiene valor semántico (sentido) en sí mismo, pero convenientemente tratado (procesado) se puede utilizar en la realización de cálculos o toma de decisiones. Los datos es la materia prima para los programas, y a partir de ellos se construye la información (datos de salida).

Los datos son comunicados por varios tipos de símbolos tales como las letras del alfabeto, números, movimientos de labios, puntos y rayas, señales con la mano, dibujos, etc. Estos símbolos se pueden ordenar y reordenar de forma utilizable y se les denomina información.

Los datos son símbolos que describen condiciones, hechos, situaciones o valores. Los datos se caracterizan por no contener ninguna información. Un dato puede significar un número, una letra, un signo ortográfico o cualquier símbolo que represente una cantidad, una medida, una palabra o una descripción.

La importancia de los datos está en su capacidad de asociarse dentro de un contexto para convertirse en información. Por si mismos los datos no tienen capacidad de comunicar un significado y por tanto no pueden afectar el comportamiento de quien los recibe. Para ser útiles, los datos deben convertirse en información para ofrecer un significado, conocimiento, ideas o conclusiones.

Los datos se clasifican en :

- variables
- constantes
- expresiones

Las variables son datos que pueden sufrir cambios en su valor durante la ejecución de un programa –o un alogoritmo-. Por ejemplo, la velocidad de un carro, el atributo nombre de un objeto Alumno, la edad de un objeto Persona, el radio de un círculo.
Las constantes son datos cuyo valor no puede ser cambiado en el transcurso de un programa. Ejemplos que podemos citar son : PI representando al valor 3.1416, MAXALU representando al número máximo de alumnos en un grupo, VELLUZ cuyo valor es 300 000 m/seg representa a la velocidad de la luz. Las constantes de tipo literal son aquellas que se denotan directamente por un valor, por ejemplo : el número entero 100, -30.75 que es un número real, una mensaje de error "ERROR DE SINTAXIS".

Las expresiones están compuestas de operandos ya sea variables y/o constantes, que forman parte de una operación sujeta a determinados operadores. Por ejemplo veamos las 4 expresiones :

```
MAXALU / 2.0

(b – sqrt(b*b – 4*a*c)) / (2.0 * a)

"hola" + "mundo"

PI * pow(radio,2.0)
```

4.3.2 ***Tipos de datos***. establecen los posibles valores que puede tomar un dato. Siempre que manejemos un dato debemos de asociarle además de un identificador, el tipo de dato al que pertenece. Por ejemplo, el atributo `calificación` de un objeto de la clase `Alumno`, es de tipo entero y los rangos de valores que puede tomar son del 0 al 100, es decir, el 0, 1, 2, 3, 4,…,99,100. El `nombre` de un objeto `Persona` es de tipo cadena, y sus valores deberán contener sólo letras y blancos. El `área` de un círculo es de tipo real –númerico con punto flotante, enteros y decimales-. El tipo de dato para un objeto siempre será el nombre de la clase a la que pertenece. Por ejemplo, un alumno será de tipo `Alumno` –su clase-.

Los lenguajes de programación tienen tipos de datos predefinidos, que también reciben el nombre de tipos de datos básicos o primitivos. El lenguaje C# proporciona los tipos de datos básicos :

TIPO DE DATO	DESCRIPCIÓN	RANGO DE VALORES
short	entero corto	-32768 hasta 32767
int	Números enteros	-2^{31} hasta $2^{31}-1$
long	Enteros largos	-2^{63} hasta $2^{63}-1$
float	Números de punto flotante	-3.4×10^{38} hasta 3.4×10^{38}
double	Números de punto flotante de doble precisión	-1.7×10^{308} hasta 1.7×10^{308}
decimal	Valores monetarios	Hasta 28 cifras significativas
string	Secuencia de caracteres	No aplicable
char	Un solo caracter	0 hasta $2^{16}-1$
bool	Valores booleanos	true o false

Veamos algunos ejemplos donde definimos los atributos de objetos en una clase indicando su tipo de dato y su identificador, en el lenguaje C#.

```
class Alumno
{
  string noControl;
  string nombre;
  int calif;

}

class Persona
{
  string nombre;
  int edad;
  char sexo;

}

class Empleado
{
  string nombre;
  string puesto;
  decimal salario;

}

class circulo
{
  float radio;
  float abscisaCentro;
  float ordenadaCentro;

}
```

Notemos que la declaración de un atributo incluye al tipo, al identificador, terminando en el caracter (;).

4.3.3 ***Identificadores***. es un nombre con el que reconocemos algún elemento de nuestro código, ya sea una clase, una variable,un objeto, un método, una constante, un atributo, una propiedad, entre otros. Este identificador es usado para referenciar al elemento mismo sea variable, objeto, método, clase, etc..

Las reglas para formar identificadores en C# son :

- Deben comenzar con letra o con guión bajo.
- Pueden continuar con letras, dígitos y/o guiones bajos.
- No deben ser iguales a una palabra reservada del lenguaje.

Existen diferentes notaciones para la construcción de identificadores, nosotros utilizaremos 2 :

- Notación *Camello*.
- Notación *Pascal*.

La notación Camello nos dice que la palabra de inicio del identificador debemos empezarla con minúscula. Si existen mas palabras en el identificador, deberán iniciarse con mayúscula. Esta notación la usaremos para identificadores de atributos, variables –objetos-. Algunos ejemplos son :

```
noControl
razonSocial
noSeguroSocial
radio
noLlantasDelanteras
```

La notación Pascal es lo mismo que la Camello, con la diferencia de que todas las palabras son iniciadas con mayúsculas. esta notación es usada para nombrar clases, métodos, propiedades. Enseguida tenemos algunos ejemplos :

```
class Alumno
{

}
```

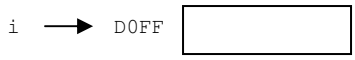
Digamos que algunos métodos de la clase `Alumno` son : `AsignarCalificacion(), AsignarNombre(), VisuaNoControl()`.

4.3.4 ***Almacenamiento, direccionamiento y representación en memoria***. El almacenamiento de datos se refiere al lugar donde ellos residen. Existen 2 formas de almacenar datos : en memoria RAM y en memoria secundaria –disco, usb-. Los datos almacenados en memoria RAM se pierden –son volátiles- cuando la computadora es apagada o cuando termina su ejecución el programa que los ha creado y almacenado. La memoria secundaria permite que los datos permanezcan almacenados de manera permanente en un sistema de archivos.

El direccionamiento es un proceso asociado al manejo de la memoria de una computadora. La memoria está compuesta de unidades denominadas bytes. Un byte a su vez se compone de 8 bits. Un bit mantiene un valor binario : 1 o 0. Los datos se almacenan en memoria usando uno o mas bytes según el tipo del dato. Cada byte en memoria tiene una dirección única, de manera que sabiendo esta dirección, podemos acceder al valor de un dato. Los valores de dirección de memoria se representan usando la notación hexadecimal. El identificador de un dato es una referencia a la dirección de memoria donde se almacena el dato. Por ejemplo, digamos que tenemos una variable entera cuyo identificador es **`i`**, su declaración es :

```
int i;
```

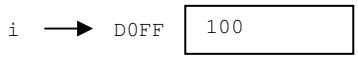
La variable i tendrá su lugar en memoria y según el lenguaje utilizado será el número de bytes que se le asignen para representarlo en memoria.



La caja representa los bytes en memoria que se le han asignado a la variable `i`. El número hexadecimal `D0FF` es la dirección de memoria que le ha correspondido a los bytes donde se almacenará el valor de la variable `i`. Nosotros como programadores no sabemos el valor de la dirección de memoria que se le ha reservado a nuestra variable i. Lo que si conocemos es el identificador con el cual vamos a manejar al dato entero que representa la `i`. De manera que sólo con saber como hacemos una sentencia de asignación en el lenguaje que usemos –en nuestro caso el lenguaje C#-, podemos acceder al dato y depositar en los bytes que le corresponden un valor determinado. Veamos la sentencia de asignación a la variable `i` que termina en caracter (;).

```
i = 100;
```

Después de ser ejecutada la sentencia de asignación a la variable i, debemos abstraer que los bytes que corresponden al dato representado por la variable `i`, tendrán o almacenan el valor del número entero 100.



En lenguaje C# los diferentes tipos de datos básicos o predefinidos se representan usando cierta cantidad de bytes, según sea el rango de valores que manejen. La tabla a continuación muestra dicha información.

TIPO DE DATO	DESCRIPCIÓN	No. de bytes
short	entero corto	2
int	Números enteros	4
long	Enteros largos	8
float	Números de punto flotante	4
double	Números de punto flotante de doble precisión	8
decimal	Valores monetarios	16
string	Secuencia de caracteres	2 bytes por caracter
char	Un solo caracter	2
bool	Valores booleanos	1

4.3.5 ***Sistema de numeración binaria y hexadecimal***. Los datos númericos tienen diferentes maneras de ser representados. La representación por default es la decimal, pero existen otros sistema igualmente útiles para ciertas ocasiones. Otras 2 representaciones muy usadas son la binaria, la octal y la hexadecimal.

El sistema de numeración binaria tiene como alfabeto sólo 2 símbolos : `{ 0, 1 }`. Para formar un número binario sólo podemos echar mano de estos 2 símbolos. Un número en cualquier sistema númerico se forma de dígitos que tienen un cierto valor significativo, que está dado por una potencia del número base. Por ejemplo, un número en sistema decimal digamos el 51, se concibe su valor de acuerdo a :

$10^1=10$	$10^0=1$	valor
5	1	
5 x 10=50	1 x 1=1	50+1=51

Ahora veamos al número binario 111101 que valor decimal representa.

$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$	valor
1	1	0	0	1	1	
1 x 32=32	1 x 16=16	0 x 8=0	0 x 4=0	1 x 2=2	1 x 1=1	32+16+0+0+2+1=51

El `51` decimal es representado en binario como `110011`.

Podemos convertir un valor decimal a un número binario haciendo divisiones sucesivas del número decimal entre la base a la cual se requiere cambiar. En cada división se cambia el dividendo por el cociente de la anterior división. Las divisiones se efectúan hasta que el cociente sea 0. Hagámoslo con el número `51` :

	cociente	Residuo	
51 / 2 =	25	1	Bit menos significativo
25 / 2 =	12	1	
12 / 2 =	6	0	
6 / 2 =	3	0	
3 / 2 =	1	1	
1 / 2 =	0	1	Bit mas significativo

Un número mas pequeño digamos el 12 :

	cociente	Residuo	
12 / 2 =	6	0	Bit menos significativo
6 / 2 =	3	0	
3 / 2 =	1	1	
1 / 2 =	0	1	Bit mas significativo

Ejercicios propuestos :

- Convierte el número binario 1010101010 a decimal.
- Convierte el número decimal 112 a binario.
- Convierte el número binario 11111000 a decimal.

El sistema numérico hexadecimal tiene como alfabeto a 16 símbolos `{ 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F }`. A partir de la combinación de estos símbolos se construyen los números hexadecimales. Ejemplos de números hexadecimales son `21`, `AA`, `F4AC`, `1234`, `CBFE`. Su correspondiente número decimal para cada número hexadecimal es el mostrado en la tabla siguiente.

16^3=4096	16^2=256	16^1=16	16^0=1	Valor decimal
		2	**1**	
		2 x 16=32	1 x 1=1	32+1=33
		A	**A**	
		10 x 16=160	10 x 1=10	160+10=170
F	**4**	**A**	**C**	
15 X 4096=61440	4 X 256=1024	10 X 16=160	12 X 1=12	61440+1024+160+12=62636
1	**2**	**3**	**4**	
1 X 4096=4096	2 X 256=512	3 X 16=48	4 X 1=4	4096+512+48+4=4660
C	**B**	**F**	**E**	

Házlo de ejercicio

La conversión de un decimal a un hexadecimal se efectúa de manera similar a como lo hicimos con el binario, sólo debemos cambiar el divisor a la base 16. Veamos algunos ejemplos.

El número decimal `33` en hexadecimal es el `21`.

	cociente	Residuo	
33 / 16 =	2	1	Bit menos significativo
2 / 16 =	0	2	Bit mas significativo

El número decimal `62716` en hexadecimal es el `F4FC`.

	cociente	Residuo	
62716 / 16 =	3919	12=C	Bit menos significativo
3919 / 16 =	244	15=F	
244 / 16 =	15	4	
15 / 16 =	0	15=F	Bit mas significativo

En ocasiones se requiere de una conversión de binario a hexadecimal o al contrario, de hexadecimal a binario. La regla que se sigue es que para cada símbolo hexadecimal, le corresponden 4 símbolos binarios. En la tabla siguiente se han listado algunos ejemplos de conversión de binario a hexadecimal si lo vemos de izquierda a derecha, y de hexadecimal a binario si lo vemos de derecha a izquierda. Los números binarios se han separado en grupos de 4 símbolos de derecha a izquierda.

binario	hexadecimal
101 1100	5A
1100 1111	AF
1001	9
01 0001	11

4.4 Operadores, operandos y expresiones.

Operadores. Los operadores nos permiten manipular datos, sean variables, constantes, otras expresiones, objetos, atributos de objetos, entre otros, de manera que podamos (1) transformarlos, (2) usarlos en decisiones para controlar el flujo de ejecución de un programa, (3) formar valores para asignarlos a otros datos. El tipo de datos involucrado en una expresión se relaciona muy de cerca con los operadores utilizados.
En nuestro curso veremos 7 tipos de operadores :

- aritméticos.
- relacionales
- lógicos
- asignación
- incremento
- decremento
- concatenación

A continuación listaremos en tablas, a cada clasificación de operadores utilizando el lenguaje C#.

Operadores aritméticos. sus operandos son numéricos int, float, double.

operador	descripción	uso	Tipo según operandos	conversión implícita
+	suma	a + b +a	binario monario	int + float = float int + double = double float + double = double
-	resta	a - b -a	binario monario	idem operador +
*	multiplicación	a * b	binario	idem operador +
/	división	a / b	binario	idem operador +
%	residuo	a % b	binario	no aplicable

Ejercicios propuestos :

- Indica la conversión implícita para `short + int`.

Operadores relacionales. sus operandos son numéricos int, float, double.

operador	descripción	uso	Tipo según operandos
<	menor que	a < b	binario
<=	menor o igual que	a <= b	binario
>	mayor que	a > b	binario
>=	mayor o igual que	a >= b	binario
==	igual que	a == b	binario
!=	diferente que	a != b	binario

Operadores lógicos. sus operandos son expresiones relacionales.

operador	descripción	uso	Tipo según operandos
&&	y	a && b	binario
||	o	a || b	binario
!	no	! a	monario

Operadores de asignación. sus operandos son numéricos int, float, double.

operador	descripción	uso	Tipo según operandos
=	asignación	a = b	binario
+=	suma	a += b, a = a + b	binario
-=	resta	a -= b, a = a - b	binario
*=	multiplicación	a *= b, a = a * b	binario
/=	división	a /= b, a = a / b	binario
%=	residuo	a %= b, a = a % b	binario

Operadores de incremento y decremento. sus operandos son numéricos int, float, double.

operador	descripción	uso	Tipo según operandos
++	incremento	a++, postfijo, a=a+1	monario
		++a, prefijo, a=a+1	monario
--	decremento	a--, postfijo, a=a-1	monario
		--a, prefijo, a=a-1	monario

Operador de concatenación. se utilza para concatenar cadenas, sus operandos son de tipo string.

operador	descripción	uso	Tipo según operandos
+	concatenación	s = s1 + s2	binario
+=	asignación y concatenación	s += s1, concatena los caracteres de s1 a s.	binario

Operandos. Son los datos a los que se les asocia un operador para efectuar una determinada operación con sus valores. Recordemos que los datos pueden ser variables o constantes. En cuanto a los datos variables ya sabemos que se les asocia un identificador y un tipo de dato, además que su identificador es una referencia a la dirección de memoria donde se ha almacenado el valor del dato. Por ejemplo, el operador `+` es usado para la suma de las variables `x` y `y` , su resultado es asignado al dato `x` mediante el uso del operador de asignación `=`.

```
x = x + y;
```

Los datos constantes involucrados como operandos pueden ser :

- literales, que son especificados por medio de un valor escrito en el código.
- simbólicas, especificadas por medio de un identificador asociado con un literal. Son mas recomendadas que el uso de literales. ¿Por qué? discútelo con tu profesor.

Ejemplo de constantes literales :

```
int ancho = 80;
string mensaje="ERROR";
```

`80` y `"ERROR"` son constantes literales. Observa que las literales también tienen tipo. Notemos que el `80` no se ha escrito con un punto al final. Las constantes literales de tipo cadena –`string`- se deben ecerrar entre comillas. Una constante literal de tipo `float` debe terminar en `f` ó `F`. Las `doubles` no deben tener esta terminación. Enseguida se muestran los ejemplos correspondientes a constantes `float -> 200.56F`, y `double -> 3.1416`.

```
float x = 200.56F;
double pi=3.1416;
```

Las constantes literales de tipo caracter –char- deberán ser escritas encerradas entre apóstrofes. Deberán contener un solo caracter , por ejemplo `'M'` asignado al dato variable `sexo` :

```
char sexo='M';
```

Ejemplo de constantes simbólicas :

```
const float PI=3.1416f;
```

Luego podemos usar la constante simbólica `PI` en una expresión, mediante su identificador :

```
float area, radio=1.5f;
area = PI * radio * radio;
```

Notemos que una constante simbólica usa un identificador asociado a un literal, y que su definición requiere de la palabra reservada del lenguaje `const`.

Expresiones. involucran operandos y operadores, siempre el resultado de la evaluación de una expresión es de un cierto tipo. La evaluación de una expresión la veremos en la siguiente sección. Por ahora diremos que dependiendo de los operadores involucrados, la expresión se denominará de asignación, aritmética, relacional, lógica, de incremeno, de decremento, de concatenación. Existen expresiones compuestas, pero el resulado será de un solo tipo. Veamos algunos ejemplos donde explicaremos algunas cuestiones acerca de las expresiones.

expresión	comentarios	estado
`int num = 100 + 40.3;`	No podemos mezclar constantes literales enteras y dobles para asignarlas a un dato entero.	ERROR
`int noVueltas = 10 * 3;`	Las constantes literales son todas enteras. La expresión es valida.	OK
`float x = 10.0 + 75.2;`	Error. Las constantes literales cuando trabajamos con float, deben terminar con f.	ERROR
`float x = 10.0F + 75.2F;`	Se ha corregido el error.	OK
`string s = ”HOLA” + “ MUNDO”;`	Tenemos 2 constantes literales de tipo string –cadena-. La expresión esta bien. El resultado depositado en la variable s, es la concatenación de las 2 constantes literales cadena, s tiene le valor de “HOLA MUNDO”.	OK
`char sexo = ”M”;`	ERROR, una constante literal string no puede asignarse a un dato tipo char. Los datos tipo char sólo pueden contener un solo caracter, además de que una constante literal char debe ser encerrada entre apóstrofes.	ERROR
`char sexo = ‘M’;`	La expresión de asignación ahora es correcta, ya que la constante literal es de tipo char, es decir, se ha encerrado entre apóstrofes.	OK

4.5 Prioridad de operadores, evaluación de expresiones.

Prioridad de operadores. Los programas de computadora evalúan expresiones de tipo aritméticas, relacionales, lógicas, de incremento, de decremento, de asignación, entre otras, tomando en cuenta las prioridades predefinidas por el lenguaje para los diferentes operadores. Esto lo apuntamos, debido a que las expresiones pueden involucrar a diferente tipo de operadores, por ejemplo :

```
! x < 0 && y >= x * 12.0 - 5.5
```

Digamos que x tiene el valor de 10, y el valor de 200.4, entonces ¿cuál sería el valor que retorna la expresión?. Para calcularlo deberemos conocer la prioridad de ejecución de los diferentes tipos de operadores en el lenguaje de programación que en nuestro caso es el C#. La prioridad de ejecución puede ser alterada por el programador por medio del uso de los paréntesis.
Otra cuestión importante en la evaluación de una expresión, es que ésta es ejecutada de izquierda a derecha por la computadora.
Entonces para saber cómo es evaluada una expresión, debemos conocer 3 cuestiones : (1) la prioridad de ejecución de los operadortes involucrados, (2) la evaluación es realizada de izquierda a derecha, (3) el uso de paréntesis permite la alteración de la prioridad de ejecución.

En la tabla mostrada a continuación podemos ver la prioridad de ejecución de ciertos operadores en C#.

NIVEL DE PRIORIDAD	OPERADORES
1	()
2	! - + ++ -- (Nota : el menos y el mas cuando son usados como operadores monarios)
3	* / %
4	+ - (Nota : usados como operadores binarios)
5	< <= > >=
6	== !=
7	&&
8	||
9	?: (Nota : operador ternario)
10	= += -= *= /= %=

Ejemplos de evaluación de expresiones :

```
x = 10 + 4 / 2                      resultado :  12
```

primero es evaluada la división 4/2 y luego la suma.

```
y = 1 / 2  +  1 / 2                 resultado :  0
```

las divisiones son evaluadas y al final la suma. ¿por qué 0?, la división de un entero entre otro entero da como resultado un entero. Entonces la división 1/2 es = 0.5 de manera que la computadora toma el 0 –cifra entera del resultado. Luego la suma de 0 + 0 es 0.

Podemos modificar la expresión para que el resultado sea el esperado, usando un molde. Un molde es un tipo de dato que se aplica a un operando para convertir su valor al tipo de dato que representa dicho molde. Este operando es el inmediato a su derecha. El molde –identificador del tipo de dato- debe encerrarse entre paréntesis.

```
y = (double) 1 / 2  +  1 / (double) 2               resultado :  1.0
```

El molde double actúa en el primer caso sobre el operando literal 1. El molde double aplicado en la 2da. división, actúa sobre el operando literal 2.

Observemos que si aplicamos el molde sobre toda la división el resultado seguirá siendo 0.

```
y = (double) (1 / 2)  +  (double) (1 / 2)           resultado :  0
```

Otras expresiones son :

```
area = PI * radio * radio          donde radio = 2.0 y tenemos la declaración const double PI = 3.1416
```

Primero es evaluado el producto 3.1416 x 2.0 = 6.2832 y su resultado multiplicado x 2.0.

```
x = 4.0 - 3.0 * y / ( 2.0 + a )      donde y = 1.0   y a = 3.5
```

Primero se evalúa 3.0 * y = 3.0, luego el paréntesis altera la prioridad del operado / de manera que primero es efectuada la suma 2.0 + a = 5.5. Una vez efectuada la suma la evaluación sigue con la división 3.0 / 5.5 = 0.54545. Por último se efectúa la resta de 4.0 - 0.54545 = 3.4546

4.6 Estructura básica de un programa.

Las aplicaciones Windows que vamos a construir en nuestro curso están compuestas de varios elementos que contienen información acerca de la aplicación misma. Estos elementos generalmente son archivos de un cierto tipo : de recursos, de interfase, de código, de ensamble, entre otros.

El ambiente que presenta el C# para construcción de aplicaciones Windows es el mostrado en la figura #4.6.1. Notemos que en la ventana principal visualiza una ventana cuyo texto en su encabezado tiene la leyenda "`Form1`".

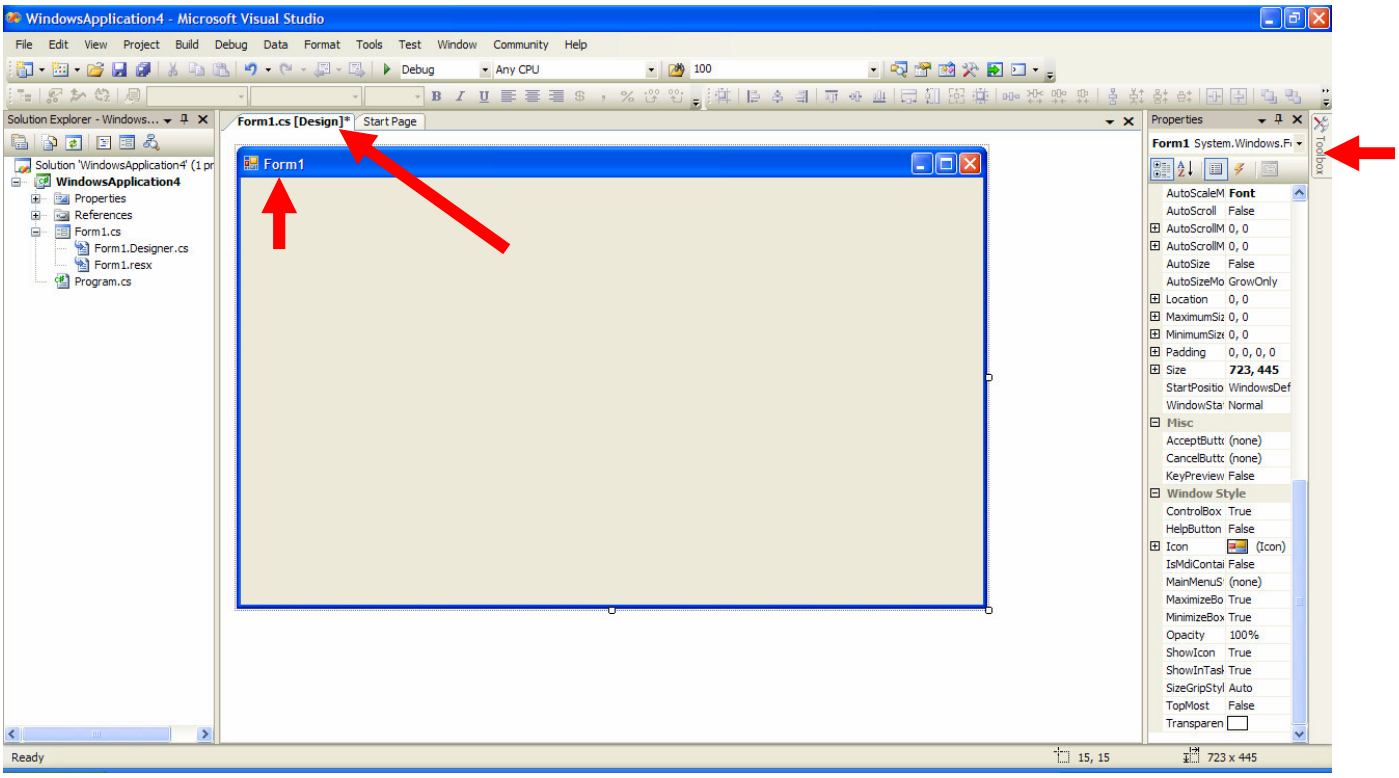


Fig. No. 4.6.1 Ambiente de desarrollo del lenguaje C#.

La ventana cuyo nombre es `Form1`, representa la interfase Windows que verá el usuario cuando sea ejecutada la aplicación. En esta forma `Form1` es donde depositaremos en tiempos de diseño los componentes que utilizaremos para "darle vida" a la aplicación.

Haremos uso de 3 tipos de componentes en nuestros programas –aplicaciones- :

- **Label** .- Sirve para desplegar texto –cadenas- como mensajes al usuario, despliegue de uno o mas resultados.
- **TextBox** .- Los utilizamos cuando requerimos leer datos. El usuario del programa podrá ingresar cualquier dato que se le solicite usando estos componentes.
- **Button** .- Permiten realizar acciones cuando el usuario hace click sobre ellos. Este componente es básico para poder escribir código que sea ejecutado cuando el usuario los oprima.

La manera en que son agregados a la forma `Form1` en tiempos de diseño, es mediante el uso de la pestaña en la parte derecha superior con leyenda *Toolbox*. La pestaña *Toolbox* despliega una ventana donde se nuestra un catálogo de todos los componentes que están disponibles para añadirlos a nuestra aplicación. La manera en que estos componentes residentes en este *Toolbox* se agregan a la aplicación, es haciéndo doble click sobre ellos o bien arrastrándolos a la forma `Form1`.

La forma `Form1` que se accede haciéndo click sobre la pestaña `Form1.cs [Design]` constituye la parte fundamental de la estructura de un programa en C#. Sin ella no existiría la ventana que ve el usuario cuando es ejecutada la aplicación. `Form1` representa la interfase de usuario de la aplicación.

En la figura #4.6.2 se muestra a la forma `Form1.cs[Design]` con un componente `Label`, un `TextBox` y un componente `Button` agregados con sus propiedades `Text` modificadas.

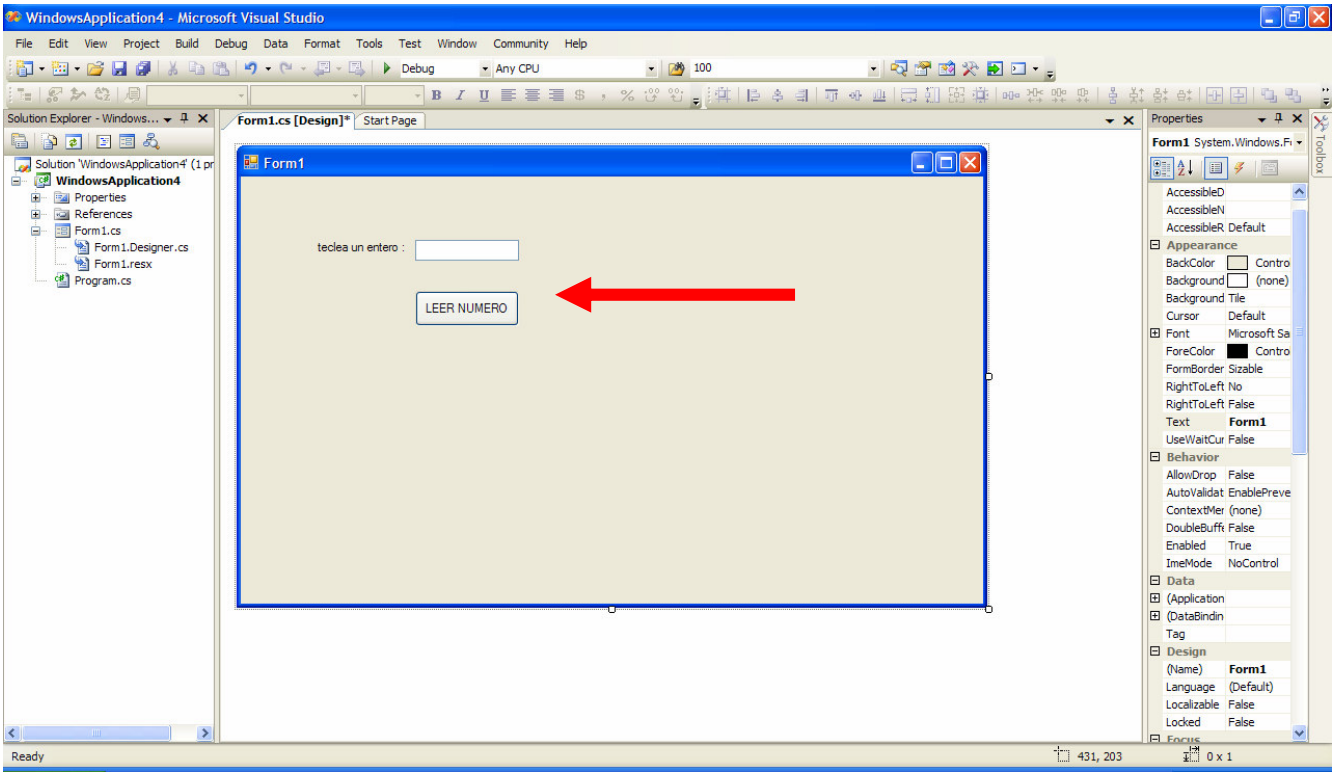


Fig. No. 4.6.2 `Form1` con 3 componentes incluídos.

Otra parte fundamental en la estructura de un programa en Windows C# es el archivo `Form1.cs` (relacionado al `Form1.cs[Design]`).
El archivo `Form1.cs` consiste de 3 partes fundamentales :

- El encabezado donde son incluídos las sentencias `using` que agregan clases predefinidas por el C# a nuestra aplicación, de manera que podamos hacer uso de sus métodos.
- El encabezado del espacio de nombres al cual pertenece nuestra aplicación. Por ahora no profundizaremos en este concepto.
- El cuerpo de la clase `Form1` que hereda –se deriva- de la clase predefinida por C# `Form`.

La importancia del archivo `Form1.cs` consiste en que es el repositorio de todo nuestro código. El programador es el encargado de darle utilidad a la aplicación. El ambiente C# sólo nos proporciona muchas facilidades tales como la interfase gráfica `Form1`, las bibliotecas de clases predefinidas que se incluyen por medio de las sentencias `using`, un conjunto de componentes que podemos agregar a la interfase `Form1` proporcionadas por la herramienta *Toolbox*, pero nosotros como programadores seremos los encargados de añadir el código que resuelva un cierto problema. TODO el código que nosotros escribimos tenemos que insertarlo en este archivo `Form1.cs`.

También escribiremos código en otros archivos que contienen una clase cada uno. Por ejemplo, si manejamos una clse `Alumno`, deberemos agregar a nuestra aplicación un archivo `Alumno.cs` que contendrá a la clase `Alumno` que encapsula a los atributos y métodos definidos para todo objeto alumno declarado en el programa residente en el archivo `Form1.cs`.

A toda forma `Form1.cs[Design]` le corresponderá siempre un solo archivo `Form1.cs`, de manera que si tenemos una aplicación con 2 formas digamos `Form1.cs[Design]` y `Form2.cs[Design]`, deberemos tener sus correspondientes archivos de código `Form1.cs` y `Form2.cs`.

Existen diferentes formas de acceder al archivo `Form1.cs`. Una de ellas es teniendo en vista la interfase `Form1.cs[Design]`, hacemos click en el botón derecho del mouse y en la opción *View Code* del menú emergente que se muestra, hacemos click con el izquierdo del ratón. Aparecerá enseguida la pestaña y el código del archivo `Form1.cs` según lo ilustra la figura #4.6.3.

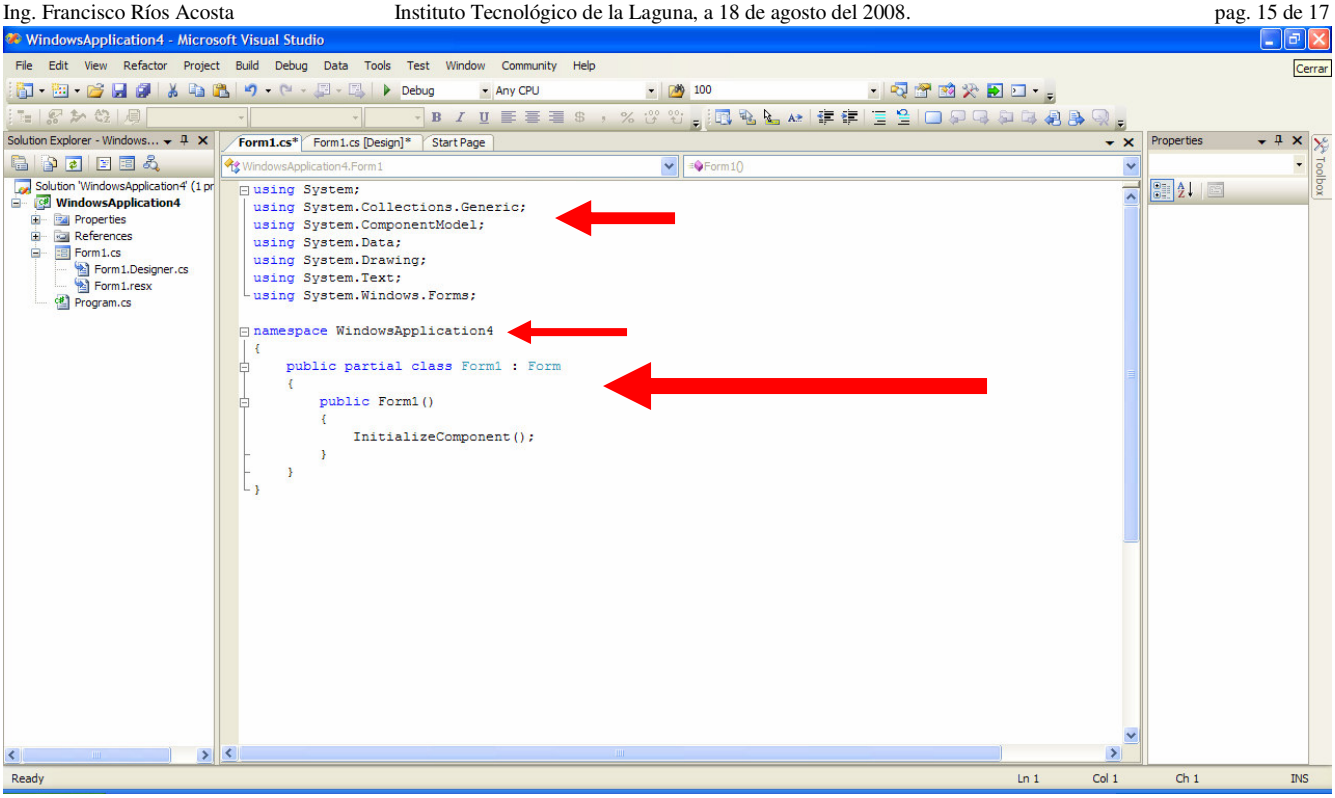


Fig. No. 4.6.3 Archivo `Form1.cs` para inserción de código por el programador.

Los archivos donde son depositadas las definiciones de las clases usadas en una aplicación, constituyen otra parte fundamental de la estructura de una aplicación Windows C#. La manera en que son agregados estos archivos de clases a la aplicación en muy simple, sólo debemos seleccionar la trayectoria de menú *Project* | *Add Class* y en la caja de diálogo que nos muestra C# teclear el nombre de la clase. La figura #4.6.4 muestra dicho diálogo para una clase Alumno.

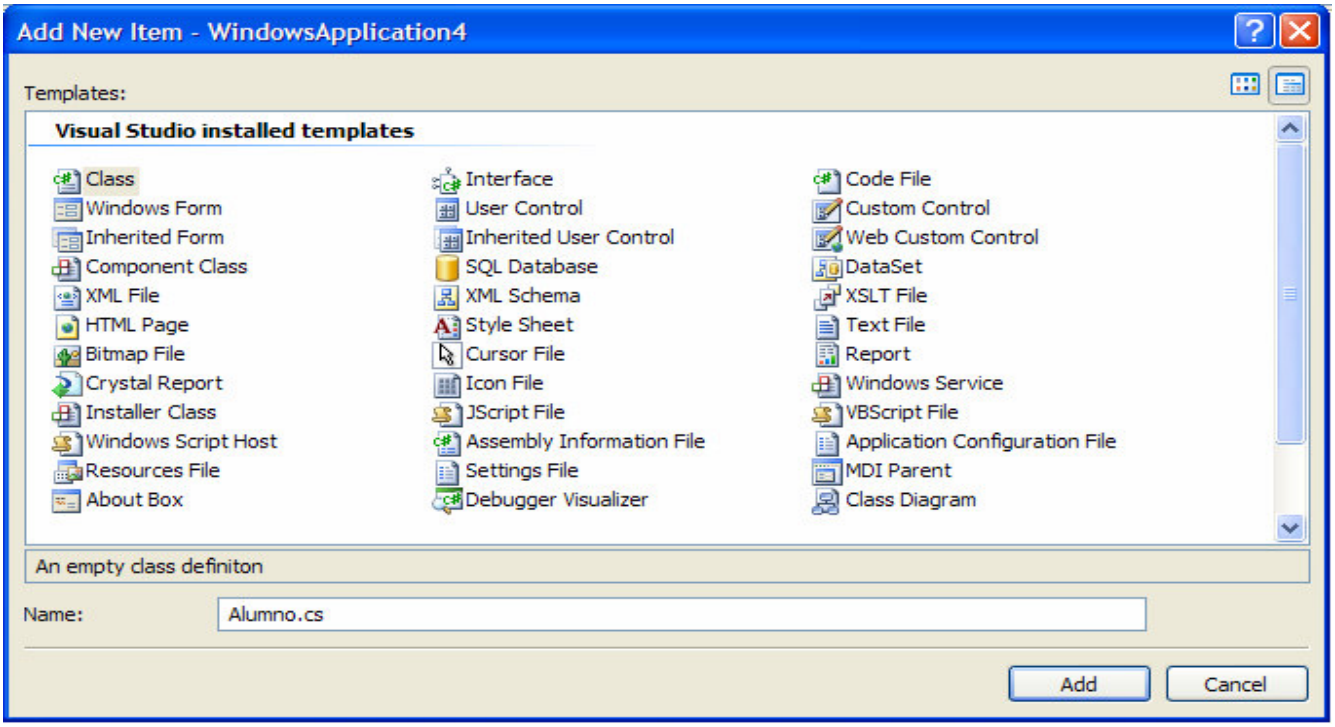


Fig. No. 4.6.4 Adición de la clase `Alumno` archivo `Alumno.cs`, al proyecto.

Una vez que hacemos click al botón con leyenda *Add*, la aplicación muestra en una pestaña nueva, los contenidos del archivo `Alumno.cs`. La figura #4.6.5 muestra la clase `Alumno` con sus atributos.

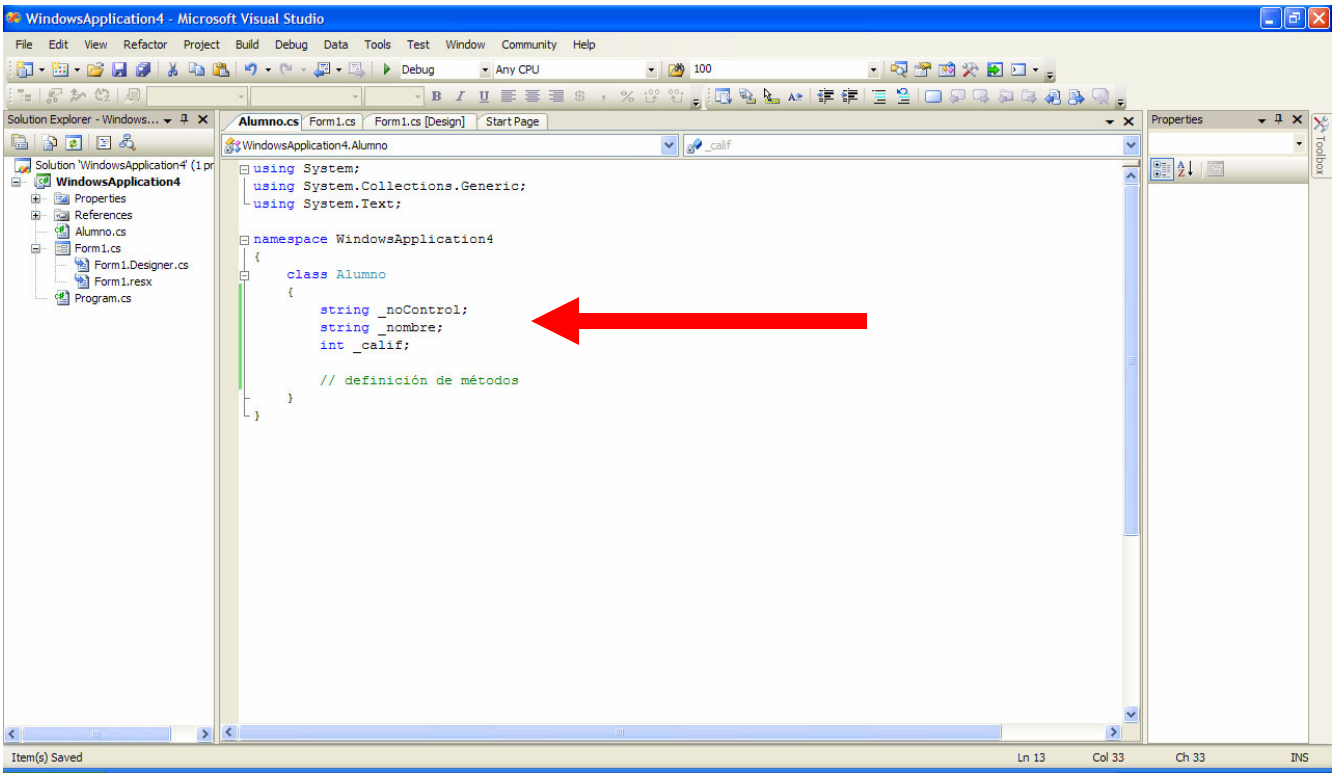


Fig. No. 4.6.5 Clase `Alumno` añadida a la aplicación.

Para concluir diremos que la estructura de un programa Windows C# está compuesta por :

- Un archivo `Form1.cs[Design]` que contiene la interfase gráfica de usuario.
- Un archivo `Form1.cs` que contiene el código escrito por el programador de la aplicación.
- Un archivo por cada clase que utilicemos en la aplicación. Ejemplo, si manejamos las clases `Alumno`, `Grupo` y `Profesor` en nuestro proggrama, deberemos tener en nuestro proyecto de la aplicación 3 archivos : `Alumno.cs`, `Grupo.cs` y `Profesor.cs`.

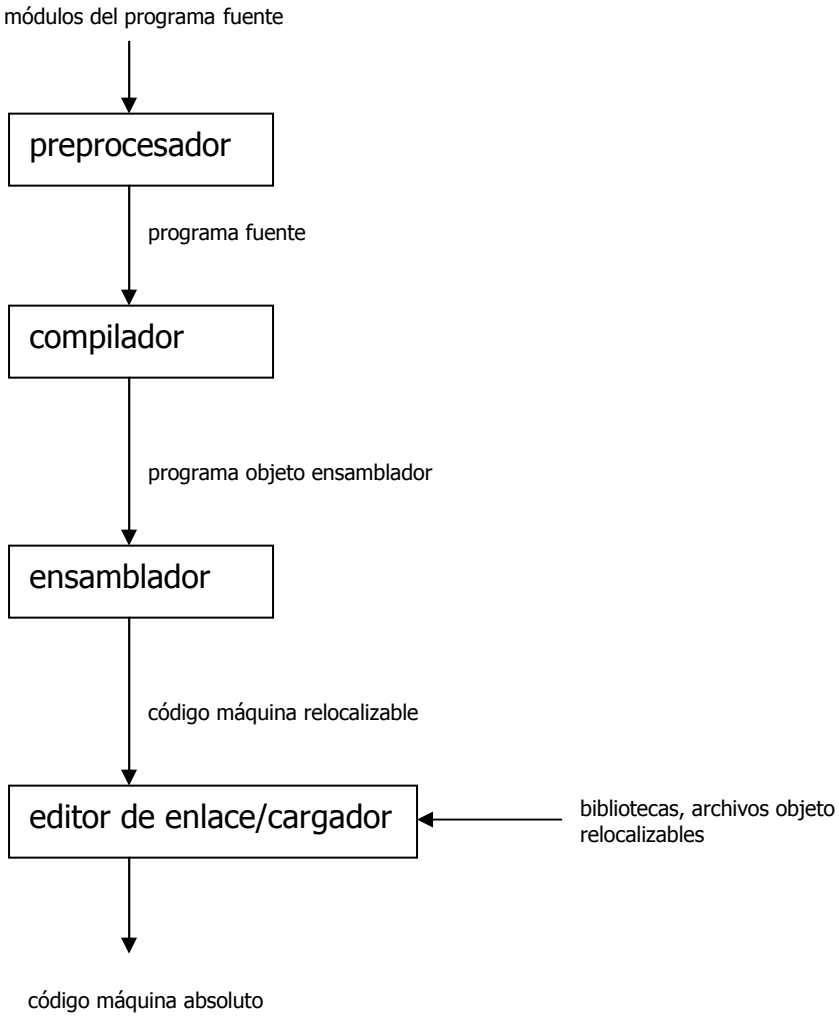
4.7 Proceso de creación de un ejecutable.

Además de un compilador, son necesarios otros programas para lograr crear un programa ejecutable. Un programa fuente puede ser dividido en varios módulos almacenados en archivos por separado. La tarea de unir los diferentes módulos que constituyen el programa fuente es asignada a un programa llamado preprocesador. Luego el programa fuente es alimentado al compilador, el cual es un programa que traduce el programa fuente en un programa objeto. este programa objeto generalmente está escrito en lenguaje ensamblador, propio del microprocesador de la computadora (Intel, AMD, etc). Este programa objeto es traducido por otro programa llamado ensamblador, que crea un programa máquina al cual se le agregan bibliotecas por medio de otro programa denominado Editor de enlaces. En este mismo programa se contiene a otro programa llamado cargador, el cual se encarga de cargar en memoria RAM de la computadora al código de máquina absoluto producido por el programa editor de enlaces.

Entonces, el proceso genérico de producir un lenguaje ejecutable consiste del uso de los programas :

- Preprocesador
- Compilador
- Ensamblador
- Editor de enlaces
- Cargador

A continuación tenemos el diagrama a bloques del proceso de creación de un programa ejecutable.



Un programa ejecutable tiene el formato binario, lo que significa que sus instrucciones y datos están compuestos de solamente 1's y 0's. El programa ejecutable es cargado en RAM por un programa llamado cargador, según lo estipulado por Von Newman en 1945 "todo programa debe ser cargado en memoria RAM para poder ser ejecutado por la computadora", cuestión que perdura hasta nuestros días.

En el ambiente de desarrollo del Visual Studio C# que estaremos usando en nuestro curso, podemos crear el programa ejecutable realizando las etapas de preprocesamiento, compilación, ensamblado, edición de enlace y carga del programa en RAM, con sólo teclear la combinación **CTRL-F5**. La aplicación en ejecución es mostrada en la figura #4.6.6.

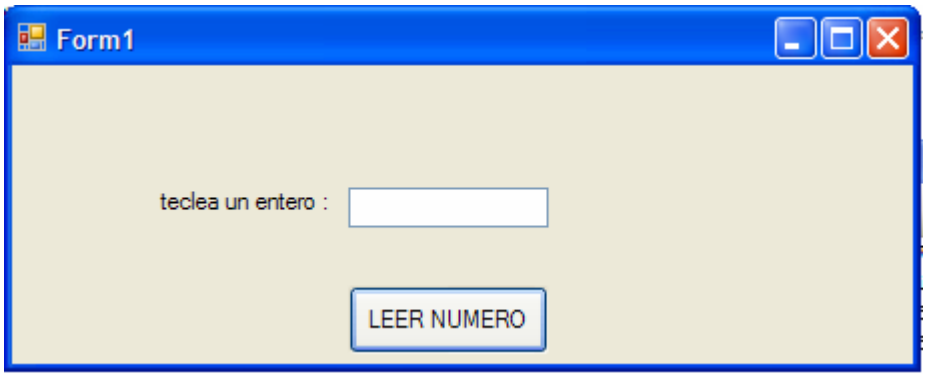


Fig. No. 4.6.6 Aplicación Windows C# en ejecución.