Manual de Usuario

**Introducción**

Este es un programa se ha sido diseñado para su uso como una aplicación de ecuaciones matemáticas. Emplea una programación en la plataforma de Visual Studio y se utilizó un lenguaje de programación en C#. Por lo tanto, se pude ejecutar en cualquier sistema operativo que tenga instala Visual Studio.

Este programa se ha utilizado la estructura de ciclo, colas y pilas para poder realizar operaciones complejas, con varios operadores a la vez, y por eso es necesario utilizar estas funciones. Se explica cuáles son los tipos de operadores y cómo funcionan, los bubles y los diferentes tipos que hay, y su funcionamiento, los operadores de condición (if, else), como se utiliza los paréntesis y el de quitar espacios.

También se puede ejecutar ese programa si no se tiene la plataforma de Visual Studio, por medio de este link: <https://drive.google.com/file/d/0B0DM_L6uhYu3MEVSQzUwMG9DSDg/view?usp=sharing>

Para que funcione hay que instalarlo y luego aparecerá un icono de un pájaro negro al cual es el .exe. luego aparecerá una ver la ventana negra donde se pide que escriba una expresión por ejemplo: ((10-2)\*2)

luego presionar dos veces enter y mostrara el resultado.

**Requisitos del programa**

Para el correcto funcionamiento del programa es necesario tener cubiertos una serie de requisitos, tanto Hardware como Software.

Los requisitos previos de Software instalados para poder ejecutar el programa serian:

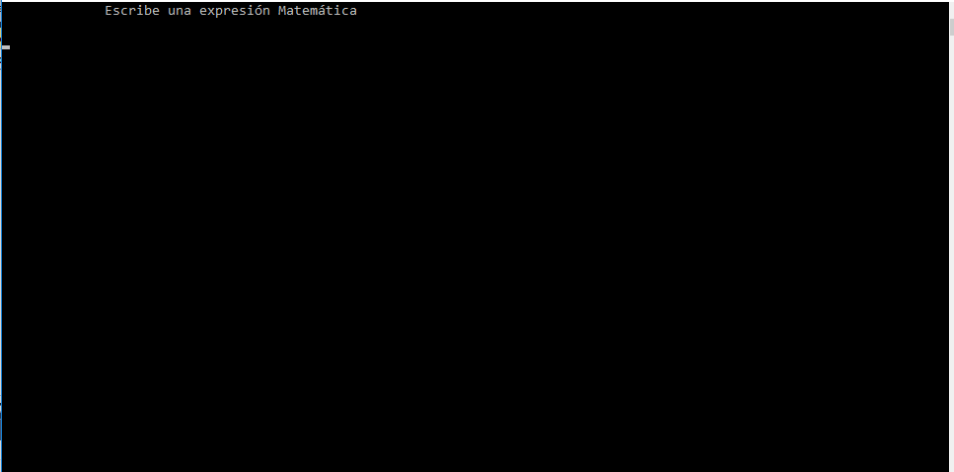
Programa de Visual Studio

Lenguaje de programación C#

O instalar el ejecutable

**Menú de opciones**

El interfaz del programa ofrece al usuario la opción de ingresar la ecuación numérica a ejecutar, como se nuestra en la Figura.



Las diferentes opciones disponibles que se pueden realizar en el programa son las que se detallan en la siguiente tabla.

|  |  |  |
| --- | --- | --- |
| Opción | Signo | Funcionalidad |
| Suma | + | Permite realizar la suma de dos o más números |
| Resta | - | Realiza la resta de dos números |
| Multiplicación | \* | Multiplica dos o más números |
| División | / | Divide dos números |
|  |  |  |
|  |  |  |
|  |  |  |

Las opciones que aparecen en la tabla son las que pude realizar el programa.

El programa utiliza varios elementos que son:

**operadores de C#**

Los operadores son los elementos que se utilizan para programar que se pueden aplicar a una o varias expresiones o instrucciones, los cuales pueden ser de incremento ( ++ ) de un operador unario, los que toman varios operadores, como las operaciones aritméticas ( +, -, \*, / ) y estos que identifica también como operadores Binarios, y los operadores de condición ( ?, : ) estos son los que toman tres valores y se conocen como el único operador ternario.

C# proporciona muchos operadores, que son símbolos que especifican las operaciones (matemáticas, indización, llamada de función, etc.) que se realizan en una expresión. Las operaciones en tipos enteros (como **==**, **!=**, **<**, **>**, **&**, **|**) se suelen permitir en tipos de enumeración (**enum**).

[**Operadores principales**](javascript:void(0))

[**x.y**](https://msdn.microsoft.com/es-gt/library/6zhxzbds.aspx)**:** acceso a miembros.

[**x?.y**](https://msdn.microsoft.com/es-gt/library/dn986595.aspx)**:** acceso a miembros condicional null.  Devuelve null si el operando izquierdo es null.

[**f(x)**](https://msdn.microsoft.com/es-gt/library/0z4503sa.aspx)**:** invocación de función.

[**a[x]**](https://msdn.microsoft.com/es-gt/library/a3hd7ste.aspx)**:** indización de objeto agregado.

[**a?[x]**](https://msdn.microsoft.com/es-gt/library/dn986595.aspx)**:** indización condicional null.  Devuelve null si el operando izquierdo es null.

[**x++**](https://msdn.microsoft.com/es-gt/library/36x43w8w.aspx)**:** incremento de postfijo.  Devuelve el valor de x y, a continuación, actualiza la ubicación de almacenamiento con el valor de x que es uno mayor (normalmente agrega el entero 1).

[**x--**](https://msdn.microsoft.com/es-gt/library/wc3z3k8c.aspx)**:** decremento de postfijo.  Devuelve el valor de x; a continuación, actualiza la ubicación de almacenamiento con el valor de x que es uno menos (normalmente resta el entero 1).

[**New**](https://msdn.microsoft.com/es-gt/library/fa0ab757.aspx)**:** creación de instancias de tipo.

[**Typeof**](https://msdn.microsoft.com/es-gt/library/58918ffs.aspx)**:** devuelve el objeto System.Type que representa el operando.

[**Checked**](https://msdn.microsoft.com/es-gt/library/74b4xzyw.aspx)**:** habilita la comprobación de desbordamiento para operaciones con enteros.

[**Unchecked**](https://msdn.microsoft.com/es-gt/library/a569z7k8.aspx)**:** deshabilita la comprobación de desbordamiento para operaciones con enteros.  Este es el comportamiento predeterminado del compilador.

[**default(T)**](https://msdn.microsoft.com/es-gt/library/xwth0h0d.aspx)**:** devuelve el valor inicializado predeterminado de tipo T, null para tipos de referencia, cero para tipos numéricos y cero o null en miembros para tipos de struct.

[**Delegate**](https://msdn.microsoft.com/es-gt/library/0yw3tz5k.aspx)**:** declara y devuelve una instancia de delegado.

[**Sizeof**](https://msdn.microsoft.com/es-gt/library/eahchzkf.aspx)**:** devuelve el tamaño en bytes del operando de tipo.

[**->**](https://msdn.microsoft.com/es-gt/library/s8bz4d5h.aspx)**:** desreferenciación del puntero combinada con acceso a miembros.

[**Operadores unarios**](javascript:void(0))

[**+x**](https://msdn.microsoft.com/es-gt/library/k1a63xkz.aspx)**:** devuelve el valor de x.

[**-x**](https://msdn.microsoft.com/es-gt/library/wch5w409.aspx)**:** negación numérica.

[**!x**](https://msdn.microsoft.com/es-gt/library/f2kd6eb2.aspx)**:** negación lógica.

[**~x**](https://msdn.microsoft.com/es-gt/library/d2bd4x66.aspx)**:** complemento bit a bit.

[**++x**](https://msdn.microsoft.com/es-gt/library/36x43w8w.aspx)**:** incremento de prefijo.  Devuelve el valor de x después de actualizar la ubicación de almacenamiento con el valor de x que es uno mayor (normalmente agrega el entero 1).

[**--x**](https://msdn.microsoft.com/es-gt/library/wc3z3k8c.aspx)**:** decremento de prefijo.  Devuelve el valor de x después de actualizar la ubicación de almacenamiento con el valor de x que es uno menos (normalmente resta el entero 1).

[**(T)x**](https://msdn.microsoft.com/es-gt/library/0z4503sa.aspx)**:** conversión de tipos.

[**Await**](https://msdn.microsoft.com/es-gt/library/hh156528.aspx): espera una Task.

[**&x**](https://msdn.microsoft.com/es-gt/library/sbf85k1c.aspx)**:** dirección de.

[**\*x**](https://msdn.microsoft.com/es-gt/library/z19tbbca.aspx)**:** desreferenciación.

## [**Operadores multiplicativos**](javascript:void(0))

[**x \* y**](https://msdn.microsoft.com/es-gt/library/z19tbbca.aspx)**:** multiplicación.

[**x / y**](https://msdn.microsoft.com/es-gt/library/3b1ff23f.aspx)**:** división.  Si los operandos son enteros, el resultado es un entero que se trunca hacia cero (por ejemplo, -7 / 2 is -3).

[**x % y**](https://msdn.microsoft.com/es-gt/library/0w4e0fzs.aspx)**:** módulo.  Si los operandos son enteros, devuelve el resto de dividir x entre y.  Si q = x / y y r = x % y, entonces x = q \* y + r.

## [**Operadores aditivos**](javascript:void(0))

.

[**x + y**](https://msdn.microsoft.com/es-gt/library/k1a63xkz.aspx)**:** suma.

[**x – y**](https://msdn.microsoft.com/es-gt/library/wch5w409.aspx)**:** resta.

## [**Operadores de desplazamiento**](javascript:void(0))

[**x << y**](https://msdn.microsoft.com/es-gt/library/a1sway8w.aspx)**:** desplaza los bits a la izquierda y rellena con cero a la derecha.

[**x >> y**](https://msdn.microsoft.com/es-gt/library/xt18et0d.aspx)**:** desplaza los bits a la derecha.  Si el operando izquierdo es **int** o **long**, los bits de la izquierda se rellenan con el bit de signo.  Si el operando izquierdo es **uint** o **ulong**, los bits de la izquierda se rellenan con cero.

## [**Operadores de comprobación de tipos y relacionales**](javascript:void(0))

[**x < y**](https://msdn.microsoft.com/es-gt/library/z5wecxwa.aspx)**:** menor que (true si x es menor que y).

[**x > y**](https://msdn.microsoft.com/es-gt/library/yxk8751b.aspx)**:** mayor que (true si x es mayor que y).

[**x <= y**](https://msdn.microsoft.com/es-gt/library/hx063734.aspx)**:** menor o igual que.

[**x >= y**](https://msdn.microsoft.com/es-gt/library/a59bsyk4.aspx)**:** menor o igual que.

[**Is**](https://msdn.microsoft.com/es-gt/library/scekt9xw.aspx)**:** compatibilidad de tipos.  Devuelve true si el operando izquierdo evaluado se puede convertir al tipo especificado en el operando derecho (un tipo estático).

[**As**](https://msdn.microsoft.com/es-gt/library/cscsdfbt.aspx)**:** conversión de tipos.  Devuelve el operando izquierdo convertido al tipo especificado por el operando derecho (un tipo estático), pero **as**devuelve **null** donde (T)x produciría una excepción.

## [**Operadores de igualdad**](javascript:void(0))

[**x == y**](https://msdn.microsoft.com/es-gt/library/53k8ybth.aspx)**:** igualdad.  De forma predeterminada, para los tipos de referencia distintos de **string**, devuelve igualdad de referencia (prueba de identidad).  Sin embargo, los tipos pueden sobrecargar **==**, por lo que si su intención es probar la identidad, es mejor usar el métodoReferenceEquals en **object**.

[**x != y**](https://msdn.microsoft.com/es-gt/library/3tz250sf.aspx)**:** distinto de.  Vea el comentario de **==**.  Si un tipo sobrecarga **==**, debe sobrecargar **!=**.

## [**Operador lógico AND**](javascript:void(0))

[**x & y**](https://msdn.microsoft.com/es-gt/library/sbf85k1c.aspx)**:** AND lógico o bit a bit.  El uso con tipos enteros y tipos **enum** suele estar permitido.

## [**Operador lógico XOR**](javascript:void(0))

[**x ^ y**](https://msdn.microsoft.com/es-gt/library/zkacc7k1.aspx)**:** XOR lógico o bit a bit.  Por lo general puede usarlo con tipos enteros y tipos **enum**.

## [**Operador lógico OR**](javascript:void(0))

[**x | y**](https://msdn.microsoft.com/es-gt/library/kxszd0kx.aspx)**:** OR lógico o bit a bit.  El uso con tipos enteros y tipos **enum** suele estar permitido.

## [**Operador condicional AND**](javascript:void(0))

[**x && y**](https://msdn.microsoft.com/es-gt/library/2a723cdk.aspx)**:** AND lógico.  Si el primer operando es false, C# no evalúa el segundo operando.

## [**Operador condicional OR**](javascript:void(0))

[**x || y**](https://msdn.microsoft.com/es-gt/library/6373h346.aspx)**:** OR lógico.  Si el primer operando es true, C# no evalúa el segundo operando.

## [**Operador de uso combinado de null**](javascript:void(0))

[**x ?? y**](https://msdn.microsoft.com/es-gt/library/ms173224.aspx)**:** devuelve x si no es **null**; de lo contrario, devuelve y.

## [**Operador condicional**](javascript:void(0))

[**t ? x : y**](https://msdn.microsoft.com/es-gt/library/ty67wk28.aspx)**:** si la prueba t es true, evalúa y devuelve x; en caso contrario, evalúa y devuelve y.

## [**Operadores de asignación y Lambda**](javascript:void(0))

[**x = y**](https://msdn.microsoft.com/es-gt/library/sbkb459w.aspx)**:** asignación.

[**x += y**](https://msdn.microsoft.com/es-gt/library/sa7629ew.aspx)**:** incremento.  Agregue el valor de y al valor de x, almacene el resultado en x y devuelva el nuevo valor.  Si x designa un **event**, y debe ser una función adecuada que C# agregue como un controlador de eventos.

[**x -= y**](https://msdn.microsoft.com/es-gt/library/2y9zhhx1.aspx)**:** decremento.  Reste el valor de y del valor de x, almacene el resultado en x y devuelva el nuevo valor.  Si x designa un **event**, y debe ser una función adecuada que C# quite como un controlador de eventos.

[**x \*= y**](https://msdn.microsoft.com/es-gt/library/s2bkaksf.aspx)**:** asignación y multiplicación.  Multiplique el valor de y por el valor de x, almacene el resultado en x y devuelva el nuevo valor.

[**x /= y**](https://msdn.microsoft.com/es-gt/library/d31sybc9.aspx)**:** asignación y división.  Divida el valor de x por el valor de y, almacene el resultado en x y devuelva el nuevo valor.

[**x %= y**](https://msdn.microsoft.com/es-gt/library/ydwa9zh0.aspx)**:** asignación y módulo.  Divida el valor de x por el valor de y, almacene el resto en x y devuelva el nuevo valor.

[**x &= y**](https://msdn.microsoft.com/es-gt/library/e669ax02.aspx)**:** asignación y AND.  AND el valor de y con el valor de x, almacene el resultado en x y devuelva el nuevo valor.

[**x |= y**](https://msdn.microsoft.com/es-gt/library/h5f1zzaw.aspx)**:** asignación y OR.  OR el valor de y con el valor de x, almacene el resultado en x y devuelva el nuevo valor.

[**x ^= y**](https://msdn.microsoft.com/es-gt/library/0zbsw2z6.aspx): asignación y XOR.  XOR el valor de y con el valor de x, almacene el resultado en x y devuelva el nuevo valor.

[**x <<= y**](https://msdn.microsoft.com/es-gt/library/ayt2kcfb.aspx)**:** asignación y desplazamiento a la izquierda.  Desplace el valor de x a la izquierda y lugares, almacene el resultado en x y devuelva el nuevo valor.

[**x >>= y**](https://msdn.microsoft.com/es-gt/library/23as4533.aspx)**:** asignación y desplazamiento a la derecha.  Desplace el valor de x a la derecha y posiciones, almacene el resultado en x y devuelva el nuevo valor.

[**=>**](https://msdn.microsoft.com/es-gt/library/bb311046.aspx)**:** declaración lambda.

## [**Desbordamiento aritmético**](javascript:void(0))

Los operadores aritméticos ([+](https://msdn.microsoft.com/es-gt/library/k1a63xkz.aspx), [-](https://msdn.microsoft.com/es-gt/library/wch5w409.aspx), [\*](https://msdn.microsoft.com/es-gt/library/z19tbbca.aspx), [/](https://msdn.microsoft.com/es-gt/library/3b1ff23f.aspx)) pueden producir resultados fuera del rango de valores posibles para el tipo numérico implicado.

El desbordamiento aritmético de enteros produce una [OverflowException](https://msdn.microsoft.com/es-gt/library/system.overflowexception.aspx) o descarta los bits más significativos del resultado.  La división de enteros por cero siempre produce una **DivideByZeroException**.

El desbordamiento aritmético de punto flotante o la división por cero nunca producen una excepción, porque los tipos de punto flotante se basan en el estándar IEEE 754 y por tanto tienen disposiciones para representar el infinito y NaN (No es un número).

El desbordamiento aritmético [decimal](https://msdn.microsoft.com/es-gt/library/364x0z75.aspx) siempre produce una [OverflowException](https://msdn.microsoft.com/es-gt/library/system.overflowexception.aspx).  La división decimal por cero siempre produce una[DivideByZeroException](https://msdn.microsoft.com/es-gt/library/system.dividebyzeroexception.aspx).

Cuando se produce el desbordamiento de enteros, lo que suceda dependerá del contexto de ejecución, que puede ser [comprobado o no comprobado](https://msdn.microsoft.com/es-gt/library/khy08726.aspx).  En un contexto comprobado, se produce una [OverflowException](https://msdn.microsoft.com/es-gt/library/system.overflowexception.aspx).  En un contexto no comprobado, se descartan los bits más significativos del resultado y la ejecución continúa.

**Bucles:**

Un bucle es una instrucción o conjunto de instrucciones que se repite un número especificado de veces o hasta que se cumpla alguna condición. El tipo de bucle que deba utilizar dependerá de la tarea de programación y de sus preferencias a la hora de escribir código.

Existen diferentes tipos de Bucles, entre ellos están:

**Foreach:** repite un grupo de instrucciones incrustadas para cada elemente de una matriz o colección de objetos, se utiliza para recorrer la colección en iteración y obtener la información deseada, pero no se puede utilizar para agregar o quitar elementos de la colección de origen.

**For:** Ejecuta una instrucción o un bloque de instrucciones repetidamente hasta que una determinada expresión se evalúa como falsa. Es utilizado para recorrer en iteración matrices y para procesar secuencialmente.

**While:** Ejecuta una instrucción o un bloque de instrucciones repetidamente hasta que una expresión específicamente se evalúa como falsa.

**do-while:** Ejecuta una instrucción o un bloque de instrucciones entre {} repetidamente hasta que una expresión específicamente como falsa.

**Operadores de Condición:**

Puede expresar cálculos que, de lo contrario, podrían requerir una construcción **if-else** más concisa mediante el operador condicional. Por ejemplo, el código siguiente usa primero una instrucción **if** y después un operador condicional para clasificar un entero como positivo o negativo.

**Paréntesis:**

Son usados comúnmente para definir la sintaxis de la estructura de algunas expresiones, y son usados comúnmente en programación para forzar y dar una prioridad a una expresión matemática o de otra índole.

Además de usarlos para especificar el orden de las operaciones en una expresión, los paréntesis se utilizan para realizar las siguientes tareas:

1. Especificar conversiones o conversiones de tipo.

double x = 1234.7;

int a;

a = (int)x; // Cast double to int

1. Invocar métodos o delegados.

TestMethod();

El operador **()** no se puede sobrecargar

Una expresión de conversión podría llevar a una sintaxis ambigua.  Por ejemplo, la expresión (x)–y puede interpretarse como una expresión de conversión (una conversión del tipo –y al tipo x) o como una expresión de suma combinada con una expresión entre paréntesis, que calcula el valor de x – y.

**Quitar espacios:**

Quita todos los caracteres de espacio en blanco del principio y el final del objeto [String](https://msdn.microsoft.com/es-es/library/system.string(v=vs.110).aspx) actual.

#### Valor devuelto (Type: [System.String](https://msdn.microsoft.com/es-es/library/system.string(v=vs.110).aspx) )

Cadena que queda una vez quitados todos los caracteres de espacio en blanco del principio y el final de la cadena actual. Si no se puede quitar ningún carácter de la instancia actual, el método devuelve la instancia actual sin cambios.

El Trim método quita de la cadena actual todos los caracteres de espacio en blanco iniciales y finales. Cada operación de recorte inicial y final se detiene cuando se encuentra un carácter no sea un espacio en blanco. Por ejemplo, si la cadena actual es "abc xyz", la Trim método devuelve "abc xyz". Para quitar los caracteres de espacio en blanco entre palabras en una cadena.

**Repeticiones:**

Esta función permite la repetición de una lista o una secuencia de instrucciones, el número de veces que las instrucciones se ejecutan, estas se pueden determinar de manera explicitas o a través de una condición que explique cuando se puede ejecutar o no.

**Tipos de estructuras de repeticiones:**

Una estructura repetitiva permite ejecutar una instrucción o un conjunto de instrucciones varias veces.

Una ejecución repetitiva de sentencias se caracteriza por:

* La o las sentencias que se repiten.
* La prueba de condición antes de cada repetición, que motivará que se repitan o no las sentencias.

En C#, todos los ciclos repiten por verdadero y cortan por falso.

Los ciclos de repetición son:

* while
* for
* do while

La sintaxis del while es:

        x=1;   Inicialización de la variable contador

        while(x<10)  Evaluación de la condición

            {

                   sentencia o sentencias a ejecutar

                   x++;   Incremento de la variable contador

             }

En este ejemplo la estructura requiere una variable entera que cumple la función de un CONTADOR de vueltas.

En la sección indicada como "inicialización contador", se suele colocar el nombre de la variable que hará de contador, asignándole a dicha variable un valor inicial. En la sección de "condición" se coloca la condición que deberá ser verdadera para que el ciclo continúe (en caso de un falso, el ciclo se detendrá). Y finalmente, en la sección de "incremento contador" se coloca una instrucción que permite modificar el valor de la variable que hace de contador (para permitir que alguna vez la condición sea falsa).

Sintaxis del for:

        for(x=0; x<10;x++)

          {

               sentencia o sentencias a ejecutar

           }

Cuando el ciclo comienza, antes de dar la primera vuelta, la variable del for toma el valor indicado en la sección de de "inicialización contador". Inmediatamente se verifica, en forma automática, si la condición es verdadera. En caso de serlo se ejecuta el bloque de operaciones del ciclo, y al finalizar el mismo se ejecuta la instrucción que se haya colocado en la tercer sección.  
Seguidamente, se vuelve a controlar el valor de la condición, y así prosigue hasta que dicha condición entregue un falso.

En general, la estructura for se usa en aquellas situaciones en las cuales CONOCEMOS la cantidad de veces que queremos que se ejecute el bloque de instrucciones. Veremos, sin embargo, que en el lenguaje C# la estructura for puede usarse en cualquier situación repetitiva, porque en última instancia no es otra cosa que una estructura while generalizada.

Sintaxis del    do  while:

        x=0;

        do

        {

             sentencia o sentencias a ejecutar

             x++; Incremento de la variable contador

         }

         while (x<10);

La estructura do while es otra estructura repetitiva, la cual ejecuta al menos una vez su bloque repetitivo, a diferencia del while o del for que podían no ejecutar el bloque.  
Esta estructura repetitiva se utiliza cuando conocemos de antemano que por lo menos una vez se ejecutará el bloque repetitivo.   
La condición de la estructura está abajo del bloque a repetir, a diferencia del while o del for que está en la parte superior.

El bloque de operaciones se repite MIENTRAS que la condición sea Verdadera.  
Si la condición retorna Falso el ciclo se detiene. 