

C# Y .NET 8

Parte 7. Evaluador de expresiones algebraicas

2024-07

Rafael Alberto Moreno Parra
ramsoftware@gmail.com

Contenido

Tabla de ilustraciones.....	3
Acerca del autor.....	4
Licencia de este libro	4
Licencia del software	4
Marcas registradas	5
Dedicatoria	6
El problema.....	7
El algoritmo	7
Paso 1: Convertirla a minúsculas y retirar cualquier caracter que no sea de una expresión algebraica	7
Paso 2: Verificando la sintaxis de la expresión algebraica	7
Paso 3: Transformando la expresión	8
Paso 4: Dividiendo la cadena en partes	9
Paso 5: Generando las Piezas desde las Partes	12
Paso 6: Evaluando a partir de las Piezas.....	12
Como usar el evaluador de expresiones	27
Con números reales	29
Con paréntesis	30
Con funciones	31
Con variables	32
Haciendo múltiples cálculos	33
Validando la sintaxis.....	35
Métricas: Desempeño con el evaluador interno de C#	39

Tabla de ilustraciones

Ilustración 1: Uso del evaluador de expresiones	27
Ilustración 2: Resultado obtenido con el evaluador	28
Ilustración 3: Operando números reales.....	29
Ilustración 4: Operando con paréntesis.....	30
Ilustración 5: Operando con funciones.....	31
Ilustración 6: Operando con uso de variables.....	32
Ilustración 7: Generar múltiples valores de una ecuación al cambiar los valores.	34
Ilustración 8: Evaluando la sintaxis	36
Ilustración 9: Evaluando la sintaxis	37
Ilustración 10: Evaluando la sintaxis	38
Ilustración 11: Métrica compara ambos evaluadores	42
Ilustración 12: Métrica compara ambos evaluadores	43
Ilustración 13: Métrica compara ambos evaluadores	43

Acerca del autor

Rafael Alberto Moreno Parra

ramsoftware@gmail.com o enginelif@hotmai.com

Sitio Web: <http://darwin.50webs.com> (dedicado a la investigación de algoritmos evolutivos y vida artificial).

Github: <https://github.com/ramsoftware>

Youtube: <https://www.youtube.com/@RafaelMorenoP>

Licencia de este libro



Licencia del software

Todo el software desarrollado aquí tiene licencia LGPL “Lesser General Public License” [1]



Marcas registradas

En este libro se hace uso de las siguientes tecnologías registradas:

Microsoft ® Windows ® Enlace: <http://windows.microsoft.com/en-US/windows/home>

Microsoft ® Visual Studio 2022 ® Enlace: <https://visualstudio.microsoft.com/es/vs/>

Dedicatoria

A mis padres, a mi hermana....

Y a mi tropa gatuna: Sally, Suini, Grisú, Capuchina, Milú,
Arián, Frac y mis recordados Tinita, Tammy, Vikingo y
Michu.

El problema

Dada una expresión algebraica almacenada en una cadena "string", esta debe ser interpretada para devolver un valor real. Ejemplo:

Cadena: "3*4+1" → Resultado: 13

Hay que considerar que las expresiones algebraicas pueden tener:

1. Números reales
2. Variables (de la a .. z)
3. Operadores (suma, resta, multiplicación, división, potencia)
4. Uso de paréntesis
5. Uso de funciones (seno, coseno, tangente, valor absoluto, arcoseno, arcocoseno, arcotangente, logaritmo natural, valor techo, exponencial, raíz cuadrada).

El algoritmo

Esta es la versión 4.0 del evaluador de expresiones.

La expresión algebraica está almacenada en una variable de tipo cadena (string). Por ejemplo:

String Cadena = "0.004 - (1.78 / 3.45 + h) * sen(k ^ x)"

La expresión algebraica puede tener números reales, operadores, paréntesis, variables y funciones. Luego hay que interpretarla y evaluarla siguiendo las estrictas reglas matemáticas.

Paso 1: Convertirla a minúsculas y retirar cualquier caracter que no sea de una expresión algebraica

Como la expresión algebraica ha sido digitada por un usuario final entonces se quita cualquier caracter(char) que no sea parte de una expresión algebraica. Los únicos caracteres permitidos son: "abcdefghijklmnopqrstuvwxyz0123456789.+*/^()"

Paso 2: Verificando la sintaxis de la expresión algebraica

Luego se verifica si la expresión cumple con las estrictas reglas sintácticas del algebra. Se hacen 26 validaciones que son:

1. Un número seguido de una letra. Ejemplo: 2q>(*3)
2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
3. Doble punto seguido. Ejemplo: 3..1
4. Punto seguido de operador. Ejemplo: 3.*1

5. Un punto y sigue una letra. Ejemplo: $3+5.w-8$
6. Punto seguido de paréntesis que abre. Ejemplo: $2-5.(4+1)*3$
7. Punto seguido de paréntesis que cierra. Ejemplo: $2-(5.)*3$
8. Un operador seguido de un punto. Ejemplo: $2-(4+.1)-7$
9. Dos operadores estén seguidos. Ejemplo: $2++4, 5-*3$
10. Un operador seguido de un paréntesis que cierra. Ejemplo: $2-(4+)-7$
11. Una letra seguida de número. Ejemplo: $7-2a-6$
12. Una letra seguida de punto. Ejemplo: $7-a.-6$
13. Un paréntesis que abre seguido de punto. Ejemplo: $7-(.4-6)$
14. Un paréntesis que abre seguido de un operador. Ejemplo: $2-(*3)$
15. Un paréntesis que abre seguido de un paréntesis que cierra. Ejemplo: $7-()-6$
16. Un paréntesis que cierra y sigue un número. Ejemplo: $(3-5)7$
17. Un paréntesis que cierra y sigue un punto. Ejemplo: $(3-5).$
18. Un paréntesis que cierra y sigue una letra. Ejemplo: $(3-5)t$
19. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: $(3-5)(4*5)$
20. Hay dos o más letras seguidas (obviando las funciones)
21. Los paréntesis están desbalanceados. Ejemplo: $3-(2*4))$
22. Doble punto en un número de tipo real. Ejemplo: $7-6.46.1+2$
23. Paréntesis que abre no corresponde con el que cierra. Ejemplo: $2+3)-2*(4 ,$
24. Inicia con operador. Ejemplo: $+3*5$
25. Finaliza con operador. Ejemplo: $3*5*$
26. Letra seguida de paréntesis que abre (obviando las funciones). Ejemplo: $4*a(6-2)$

Paso 3: Transformando la expresión

Se agrega un paréntesis que abre al inicio y luego un paréntesis al final.

De ser: $"0.004-(1.78/3.45+h)*\text{sen}(k^x)"$

Pasa a: $"(0.004-(1.78/3.45+h)*\text{sen}(k^x))"$

Es necesario ese paso porque el evaluador busca los paréntesis para extraer la expresión interna.

Luego convierte las funciones (seno, coseno, tangente) detectadas en alguna letra mayúscula predeterminada.

De ser: $"(0.004-(1.78/3.45+h)*\text{sen}(k^x))"$

Pasa a: $"(0.004-(1.78/3.45+h)*A(k^x))"$

Esta es la tabla de conversión:

Función	Descripción	Letra con que se reemplaza
---------	-------------	----------------------------

Sen	Seno	A
Cos	Coseno	B
Tan	Tangente	C
Abs	Valor absoluto	D
Asn	Arcoseno	E
Acs	Arcocoseno	F
Atn	Arcotangente	G
Log	Logaritmo Natural	H
Cei	Valor techo	I
Exp	Exponencial	J
Sqr	Raíz cuadrada	K

Paso 4: Dividiendo la cadena en partes

Se toma la cadena y se divide en partes diferenciadas: número real, operador, variable, paréntesis que abre, paréntesis que cierra y función. Por ejemplo:

"(0.004-(1.78/3.45+h)*A(k^x))"

Queda en:

(0.004	-	(1.78	/	3.45	+	h)	*	A	(k	^	x))
---	-------	---	---	------	---	------	---	---	---	---	---	---	---	---	---	---	---

Las variables y números constantes quedan en una lista dinámica llamada Valores de tipo double (las variables guardan sus respectivos valores allí). De esa forma cuando el evaluador necesite un valor para consultarlo o modificarlo, lo busca directamente en esa lista. Este cambio algorítmico da como resultado, que esta versión del evaluador (la 4.0), sea más rápida que la versión anterior del evaluador de expresiones.

Lista **Valores**

Posición 0 a 25	26	27	28
Variables de la expresión	0.004	1.78	3.45

Por lo que la lista de partes queda ahora así:

([26] - ([27] / [28] + [7]) * A ([10] ^ [23]))

Lo que está entre [] es la posición del valor en la lista de Valores.

En C# esta sería la lista de Valores

```
private List<double> Valores = new List<double>();
```

Las partes se guardan aquí:

```
/* Listado de partes en que se divide la expresión
Toma una expresión, por ejemplo:
1.68 + sen( 3 / x ) * ( 2.92 - 9 )
y la divide en partes así:
[1.68] [+] [sen() [3] [/] [x] []] [*] [(] [2.92] [-] [9] []]
Cada parte puede tener un número, un operador, una función,
un paréntesis que abre o un paréntesis que cierra.
En esta versión 4.0, las constantes, piezas y variables guardan
sus valores en la lista Valores.
En partes, se almacena la posición en Valores */
private List<ParteEvl4> Partes = new List<ParteEvl4>();
```

Y la clase de las partes es así:

```
internal class ParteEvl4 {
    /* Constantes de los diferentes tipos
    * de datos que tendrán las piezas */
    private const int ESFUNCION = 1;
    private const int ESPARABRE = 2;
    private const int ESOPERADOR = 4;
    private const int ESNUMERO = 5;
    private const int ESVARIABLE = 6;

    /* Acumulador, función, paréntesis que abre,
    * paréntesis que cierra, operador,
    * número, variable */
```

```

public int Tipo;

/* Código de la función 0:seno, 1:coseno, 2:tangente,
 * 3: valor absoluto, 4: arcoseno, 5: arcocoseno,
 * 6: arcotangente, 7: logaritmo natural, 8: valor tope,
 * 9: exponencial, 10: raíz cuadrada */
public int Funcion;

/* + suma - resta * multiplicación / división ^ potencia */
public int Operador;

/* Posición en lista de valores del número literal
 * por ejemplo: 3.141592 */
public int posNumero;

/* Posición en lista de valores del
 * valor de la variable algebraica */
public int posVariable;

/* Posición en lista de valores del valor de la pieza.
 * Por ejemplo:
   3 + 2 / 5 se convierte así:
   |3| |+| |2| | / | |5|
   |3| |+| |A| A es un identificador de acumulador */
public int posAcumula;

public ParteEv14(int TipoParte, int Valor) {
    this.Tipo = TipoParte;
    switch (TipoParte) {
        case ESFUNCION: this.Funcion = Valor; break;
        case ESNUMERO: this.posNumero = Valor; break;
        case ESVARIABLE: this.posVariable = Valor; break;
        case ESPARABRE: this.Funcion = -1; break;
    }
}

public ParteEv14(char Operador) {
    this.Tipo = ESOPERADOR;
    switch (Operador) {
        case '+': this.Operador = 0; break;
        case '-': this.Operador = 1; break;
        case '*': this.Operador = 2; break;
        case '/': this.Operador = 3; break;
        case '^': this.Operador = 4; break;
    }
}
}

```

Paso 5: Generando las Piezas desde las Partes

Las Piezas tienen esta estructura: [Pieza] Función Parte1 Operador Parte2

Esta sería la clase que representa las piezas:

```
internal class PiezaEvl4 {  
    /* Posición donde se almacena el valor que genera  
    * la pieza al evaluarse */  
    public int PosResultado;  
  
    /* Código de la función 0:seno, 1:coseno, 2:tangente,  
    * 3: valor absoluto, 4: arcoseno, 5: arcocoseno,  
    * 6: arcotangente, 7: logaritmo natural, 8: valor tope,  
    * 9: exponencial, 10: raíz cuadrada, 11: raíz cúbica */  
    public int Funcion;  
  
    /* Posición donde se almacena la primera parte de la pieza */  
    public int pA;  
  
    /* 0 suma 1 resta 2 multiplicación 3 división 4 potencia */  
    public int Operador;  
  
    /* Posición donde se almacena la segunda parte de la pieza */  
    public int pB;  
}
```

Esas Piezas se construyen desde las Partes. Se sigue el orden de evaluación de los paréntesis y operadores. De:

([26]	-	([27]	/	[28]	+	[7])	*	A	([10]	^	[23]))
---	------	---	---	------	---	------	---	-----	---	---	---	---	------	---	------	---	---

Queda así:

Pieza	Función	Parte1	Operador	Parte2
[29]	A	[10]	^	[23]
[30]		[27]	/	[28]
[31]		[30]	+	[7]
[32]		[31]	*	[29]
[33]		[26]	-	[32]

Paso 6: Evaluando a partir de las Piezas

Yendo de pieza en pieza se evalúa toda la expresión.

```

using System.Globalization;
using System.Text;

/* Evaluador de expresiones versión 4 (enero de 2024)
 * Autor: Rafael Alberto Moreno Parra
 * Correo: ramsoftware@gmail.com ; enginelife@hotmail.com
 * URL: http://darwin.50webs.com
 * GitHub: https://github.com/ramsoftware
 * */

namespace Ejemplo {

    internal class ParteEvl4 {
        /* Constantes de los diferentes tipos
         * de datos que tendrán las piezas */
        private const int ESFUNCION = 1;
        private const int ESPARABRE = 2;
        private const int ESOPERADOR = 4;
        private const int ESNUMERO = 5;
        private const int ESVARIABLE = 6;

        /* Acumulador, función, paréntesis que abre,
         * paréntesis que cierra, operador,
         * número, variable */
        public int Tipo;

        /* Código de la función 0:seno, 1:coseno, 2:tangente,
         * 3: valor absoluto, 4: arcoseno, 5: arcocoseno,
         * 6: arcotangente, 7: logaritmo natural, 8: valor tope,
         * 9: exponencial, 10: raíz cuadrada */
        public int Funcion;

        /* + suma - resta * multiplicación / división ^ potencia */
        public int Operador;

        /* Posición en lista de valores del número literal
         * por ejemplo: 3.141592 */
        public int posNumero;

        /* Posición en lista de valores del
         * valor de la variable algebraica */
        public int posVariable;

        /* Posición en lista de valores del valor de la pieza.
         * Por ejemplo:
         * 3 + 2 / 5 se convierte así:
         * |3| |+| |2| | / | |5|

```

```

    |3| |+| |A|  A es un identificador de acumulador */
public int posAcumula;

public ParteEvl4(int TipoParte, int Valor) {
    this.Tipo = TipoParte;
    switch (TipoParte) {
        case ESFUNCION: this.Funcion = Valor; break;
        case ESNUMERO: this.posNumero = Valor; break;
        case ESVARIABLE: this.posVariable = Valor; break;
        case ESPARABRE: this.Funcion = -1; break;
    }
}

public ParteEvl4(char Operador) {
    this.Tipo = ESOPERADOR;
    switch (Operador) {
        case '+': this.Operador = 0; break;
        case '-': this.Operador = 1; break;
        case '*': this.Operador = 2; break;
        case '/': this.Operador = 3; break;
        case '^': this.Operador = 4; break;
    }
}
}

internal class PiezaEvl4 {
    /* Posición donde se almacena el valor que genera
    * la pieza al evaluarse */
    public int PosResultado;

    /* Código de la función 0:seno, 1:coseno, 2:tangente,
    * 3: valor absoluto, 4: arcoseno, 5: arcocoseno,
    * 6: arcotangente, 7: logaritmo natural, 8: valor tope,
    * 9: exponencial, 10: raíz cuadrada, 11: raíz cúbica */
    public int Funcion;

    /* Posición donde se almacena la primera parte de la pieza */
    public int pA;

    /* 0 suma 1 resta 2 multiplicación 3 división 4 potencia */
    public int Operador;

    /* Posición donde se almacena la segunda parte de la pieza */
    public int pB;
}

```

```

internal class Evaluador4 {
    /* Constantes de los diferentes tipos
     * de datos que tendrán las piezas */
    private const int ESFUNCION = 1;
    private const int ESPARABRE = 2;
    private const int ESPARCIERRA = 3;
    private const int ESOPERADOR = 4;
    private const int ESNUMERO = 5;
    private const int ESVARIABLE = 6;
    private const int ESACUMULA = 7;

    /* Expresión algebraica convertida de la
     * original dada por el usuario */
    private StringBuilder Analizada = new();

    /* Donde guarda los valores de variables, constantes y piezas */
    private List<double> Valores = new List<double>();

    /* Listado de partes en que se divide la expresión
     Toma una expresión, por ejemplo:
     1.68 + sen( 3 / x ) * ( 2.92 - 9 )
     y la divide en partes así:
     [1.68] [+] [sen() [3] [/] [x] []] [*] [(] [2.92] [-] [9] []]
     Cada parte puede tener un número, un operador, una función,
     un paréntesis que abre o un paréntesis que cierra.
     En esta versión 4.0, las constantes, piezas y variables guardan
     sus valores en la lista Valores.
     En partes, se almacena la posición en Valores */
    private List<ParteEvl4> Partes = new List<ParteEvl4>();

    /* Listado de piezas que ejecutan
     Toma las partes y las divide en piezas con
     la siguiente estructura:

     acumula = funcion valor operador valor

     donde valor puede ser un número, una variable o
     un acumulador

     Siguiendo el ejemplo anterior sería:
     A = 2.92 - 9
     B = sen( 3 / x )
     C = B * A
     D = 1.68 + C

     Esas piezas se evalúan de arriba a abajo y así
     se interpreta la ecuación */
    private List<PiezaEvl4> Piezas = new List<PiezaEvl4>();

```

```

/* Analiza la expresión */
public int Analizar(string ExpresionOriginal) {
    Partes.Clear();
    Piezas.Clear();
    Valores.Clear();

    /* Hace espacio para las 26 variables que
     * puede manejar el evaluador */
    for (int Variables = 1; Variables <= 26; Variables++)
        Valores.Add(0);

    int Sintaxis = ChequeaSintaxis(ExpresionOriginal);
    if (Sintaxis == 0) {
        CrearPartes();
        CrearPiezas();
    }
    return Sintaxis;
}

/* Retorna mensaje de error sintáctico */
public string MensajeError(int CodigoError) {
    string Msj = "";
    switch (CodigoError) {
        case 1:
            Msj = "1. Número seguido de letra";
            break;

        case 2:
            Msj = "2. Número seguido de paréntesis que abre";
            break;

        case 3:
            Msj = "3. Doble punto seguido";
            break;

        case 4:
            Msj = "4. Punto seguido de operador";
            break;

        case 5:
            Msj = "5. Punto y sigue una letra";
            break;

        case 6:
            Msj = "6. Punto seguido de paréntesis que abre";
            break;
    }
}

```



```
case 7:
    Msj = "7. Punto seguido de paréntesis que cierra";
    break;

case 8:
    Msj = "8. Operador seguido de un punto";
    break;

case 9:
    Msj = "9. Dos operadores estén seguidos";
    break;

case 10:
    Msj = "10. Operador seguido de paréntesis que cierra";
    break;

case 11:
    Msj = "11. Letra seguida de número";
    break;

case 12:
    Msj = "12. Letra seguida de punto";
    break;

case 13:
    Msj = "13. Letra seguida de otra letra";
    break;

case 14:
    Msj = "14. Letra seguida de paréntesis que abre";
    break;

case 15:
    Msj = "15. Paréntesis que abre seguido de punto";
    break;

case 16:
    Msj = "16. Paréntesis que abre y sigue operador";
    break;

case 17:
    Msj = "17. Paréntesis que abre y luego cierra";
    break;

case 18:
    Msj = "18. Paréntesis que cierra y sigue número";
    break;
```

```

        case 19:
            Msj = "19. Paréntesis que cierra y sigue punto";
            break;

        case 20:
            Msj = "20. Paréntesis que cierra y sigue letra";
            break;

        case 21:
            Msj = "21. Paréntesis que cierra y luego abre";
            break;

        case 22:
            Msj = "22. Inicia con operador";
            break;

        case 23:
            Msj = "23. Finaliza con operador";
            break;

        case 24:
            Msj = "24. No hay correspondencia entre paréntesis";
            break;

        case 25:
            Msj = "25. Paréntesis desbalanceados";
            break;

        case 26:
            Msj = "26. Dos o más puntos en número real";
            break;
    }
    return Msj;
}

/* Da valor a las variables que tendrá
 * la expresión algebraica */
public void DarValorVariable(char varAlgebra, double Valor) {
    Valores[varAlgebra - 'a'] = Valor;
}

/* Evalúa la expresión convertida en piezas */
public double Evaluar() {
    double Resulta = 0;
    PiezaEvl4 tmp;

    /* Va de pieza en pieza */
    for (int Posicion = 0; Posicion < Piezas.Count; Posicion++) {

```

```

    tmp = Piezas[Posicion];

    switch (tmp.Operador) {
        case 0:
            Resulta = Valores[tmp.pA] + Valores[tmp.pB];
            break;
        case 1:
            Resulta = Valores[tmp.pA] - Valores[tmp.pB];
            break;
        case 2:
            Resulta = Valores[tmp.pA] * Valores[tmp.pB];
            break;
        case 3:
            Resulta = Valores[tmp.pA] / Valores[tmp.pB];
            break;
        default:
            Resulta = Math.Pow(Valores[tmp.pA], Valores[tmp.pB]);
            break;
    }

    switch (tmp.Funcion) {
        case 0: Resulta = Math.Sin(Resulta); break;
        case 1: Resulta = Math.Cos(Resulta); break;
        case 2: Resulta = Math.Tan(Resulta); break;
        case 3: Resulta = Math.Abs(Resulta); break;
        case 4: Resulta = Math.Asin(Resulta); break;
        case 5: Resulta = Math.Acos(Resulta); break;
        case 6: Resulta = Math.Atan(Resulta); break;
        case 7: Resulta = Math.Log(Resulta); break;
        case 8: Resulta = Math.Ceiling(Resulta); break;
        case 9: Resulta = Math.Exp(Resulta); break;
        case 10: Resulta = Math.Sqrt(Resulta); break;
    }

    Valores[tmp.PosResultado] = Resulta;
}
return Resulta;
}

/* Divide la expresión en partes, yendo de caracter en caracter */
private void CrearPartes() {
    StringBuilder Numero = new StringBuilder();
    for (int Pos = 0; Pos < this.Analizada.Length; Pos++) {
        char Letra = this.Analizada[Pos];
        switch (Letra) {
            case '.':
            case '0':
            case '1':

```

```

case '2':
case '3':
case '4':
case '5':
case '6':
case '7':
case '8':
case '9':    /* Si es un dígito o un punto
              * va acumulando el número */
    Numero.Append(Letra); break;
case '+':
case '-':
case '*':
case '/':
case '^':
    /* Si es un operador matemático entonces verifica
     * si hay un número que se ha acumulado */
    if (Numero.Length > 0) {
        Valores.Add(Convierte(Numero));
        Partes.Add(new ParteEvl4(ESNUMERO, Valores.Count - 1));
        Numero.Clear();
    }
    /* Agregar el operador matemático */
    Partes.Add(new ParteEvl4(Letra));
    break;

case '(':    /* Es paréntesis que abre */
    Partes.Add(new ParteEvl4(ESPARABRE, 0));
    break;

case ')':
    /* Si es un paréntesis que cierra entonces verifica
     * si hay un número que se ha acumulado */
    if (Numero.Length > 0) {
        Valores.Add(Convierte(Numero));
        Partes.Add(new ParteEvl4(ESNUMERO, Valores.Count - 1));
        Numero.Clear();
    }

    /* Si sólo había un número o variable
     * dentro del paréntesis le agrega + 0
     * (por ejemplo: sen(x) o 3*(2) ) */
    if (Partes[Partes.Count - 2].Tipo == ESPARABRE ||
        Partes[Partes.Count - 2].Tipo == ESFUNCION) {
        Partes.Add(new ParteEvl4(ESOPERADOR, 0));
        Valores.Add(0);
        Partes.Add(new ParteEvl4(ESNUMERO, Valores.Count - 1));
    }
}

```

```

        /* Adiciona el paréntesis que cierra */
        Partes.Add(new ParteEvl4(ESPARCIERRA, 0));
        break;
    case 'A':    /* Seno */
    case 'B':    /* Coseno */
    case 'C':    /* Tangente */
    case 'D':    /* Valor absoluto */
    case 'E':    /* Arcoseno */
    case 'F':    /* Arcocoseno */
    case 'G':    /* Arcotangente */
    case 'H':    /* Logaritmo natural */
    case 'I':    /* Exponencial */
    case 'J':    /* Raíz cuadrada */
        Partes.Add(new ParteEvl4(ESFUNCION, Letra - 'A'));
        Pos++;
        break;
    default:
        Partes.Add(new ParteEvl4(ESVARIABLE, Letra - 'a'));
        break;
    }
}
}

private double Convierte(StringBuilder Numero) {
    string Cad = Numero.ToString();
    return double.Parse(Cad, CultureInfo.InvariantCulture);
}

/* Convierte las partes en las piezas finales de ejecución */
private void CrearPiezas() {
    int Contador = Partes.Count - 1;
    do {
        ParteEvl4 tmpParte = Partes[Contador];
        if (tmpParte.Tipo == ESPARABRE || tmpParte.Tipo == ESFUNCION) {

            /* Evalúa las potencias */
            GeneraPiezaOpera(4, 4, Contador);

            /* Luego evalúa multiplicar y dividir */
            GeneraPiezaOpera(2, 3, Contador);

            /* Finalmente evalúa sumar y restar */
            GeneraPiezaOpera(0, 1, Contador);

            /* Agrega la función a la última pieza */
            if (tmpParte.Tipo == ESFUNCION) {
                Piezas[Piezas.Count - 1].Funcion = tmpParte.Funcion;
            }
        }
        Contador--;
    } while (Contador > 0);
}

```

```

    }

    /* Quita el paréntesis/función que abre y
     * el que cierra, dejando el centro */
    Partes.RemoveAt(Contador);
    Partes.RemoveAt(Contador + 1);
}
Contador--;
} while (Contador >= 0);
}

/* Genera las piezas buscando determinado operador */
private void GeneraPiezaOpera(int OperA, int OperB, int Inicia) {
    int Contador = Inicia + 1;
    do {
        ParteEvl4 tmpParte = Partes[Contador];
        if (tmpParte.Tipo == ESOPERADOR &&
            (tmpParte.Operador == OperA || tmpParte.Operador == OperB)) {
            ParteEvl4 tmpParteIzq = Partes[Contador - 1];
            ParteEvl4 tmpParteDer = Partes[Contador + 1];

            /* Crea Pieza */
            PiezaEvl4 NuevaPieza = new PiezaEvl4();
            NuevaPieza.Funcion = -1;

            switch (tmpParteIzq.Tipo) {
                case ESNUMERO:
                    NuevaPieza.pA = tmpParteIzq.posNumero;
                    break;

                case ESVARIABLE:
                    NuevaPieza.pA = tmpParteIzq.posVariable;
                    break;

                default:
                    NuevaPieza.pA = tmpParteIzq.posAcumula;
                    break;
            }

            NuevaPieza.Operador = tmpParte.Operador;

            switch (tmpParteDer.Tipo) {
                case ESNUMERO:
                    NuevaPieza.pB = tmpParteDer.posNumero;
                    break;

                case ESVARIABLE:
                    NuevaPieza.pB = tmpParteDer.posVariable;

```

```

        break;

    default:
        NuevaPieza.pB = tmpParteDer.posAcumula;
        break;
    }

    /* Añade a lista de piezas y crea una
     * nueva posición en Valores */
    Valores.Add(0);
    NuevaPieza.PosResultado = Valores.Count - 1;
    Piezas.Add(NuevaPieza);

    /* Elimina la parte del operador y la siguiente */
    Partes.RemoveAt(Contador);
    Partes.RemoveAt(Contador);

    /* Retorna el contador en uno para tomar
     * la siguiente operación */
    Contador--;

    /* Cambia la parte anterior por parte que acumula */
    tmpParteIzq.Tipo = ESACUMULA;
    tmpParteIzq.posAcumula = NuevaPieza.PosResultado;
}
Contador++;
} while (Partes[Contador].Tipo != ESPARCIERRA);
}

private int ChequeaSintaxis(string ExpOrig) {
    /* Primero a minúsculas */
    StringBuilder Minusculas = new StringBuilder(ExpOrig.ToLower());

    /* Sólo los caracteres válidos */
    string Valido = "abcdefghijklmnopqrstuvwxyz0123456789.+*/^()";
    HashSet<char> Permite = new HashSet<char>(Valido);
    StringBuilder ConLetrasValidas = new StringBuilder("");
    for (int Cont = 0; Cont < Minusculas.Length; Cont++)
        if (Permite.Contains(Minusculas[Cont]))
            ConLetrasValidas.Append(Minusculas[Cont]);
    ConLetrasValidas.Append(' ');

    /* Agrega +0) donde exista )) porque es
     * necesario para crear las piezas */
    string nuevo = ConLetrasValidas.ToString();
    while (nuevo.IndexOf("))") != -1)
        nuevo = nuevo.Replace("))", ") +0)");
    ConLetrasValidas = new StringBuilder(nuevo);
}

```

```

/* Validación de sintaxis, se genera una copia
 * y allí se reemplaza las funciones por a+( */
StringBuilder sbSintax = new StringBuilder();
sbSintax.Append(ConLetrasValidas);
sbSintax.Replace("sen(", "a+(");
sbSintax.Replace("cos(", "a+(");
sbSintax.Replace("tan(", "a+(");
sbSintax.Replace("abs(", "a+(");
sbSintax.Replace("asn(", "a+(");
sbSintax.Replace("acs(", "a+(");
sbSintax.Replace("atn(", "a+(");
sbSintax.Replace("log(", "a+(");
sbSintax.Replace("cei(", "a+(");
sbSintax.Replace("exp(", "a+(");
sbSintax.Replace("sqr(", "a+(");

for (int Cont = 1; Cont < sbSintax.Length - 2; Cont++) {
    char cA = sbSintax[Cont];
    char cB = sbSintax[Cont + 1];

    if (Char.IsDigit(cA) && Char.IsLower(cB)) return 1;
    if (Char.IsDigit(cA) && cB == '(') return 2;
    if (cA == '.' && cB == '.') return 3;
    if (cA == '.' && EsUnOperador(cB)) return 4;
    if (cA == '.' && Char.IsLower(cB)) return 5;
    if (cA == '.' && cB == '(') return 6;
    if (cA == '.' && cB == ')') return 7;
    if (EsUnOperador(cA) && cB == '.') return 8;
    if (EsUnOperador(cA) && EsUnOperador(cB)) return 9;
    if (EsUnOperador(cA) && cB == ')') return 10;
    if (Char.IsLower(cA) && Char.IsDigit(cB)) return 11;
    if (Char.IsLower(cA) && cB == '.') return 12;
    if (Char.IsLower(cA) && Char.IsLower(cB)) return 13;
    if (Char.IsLower(cA) && cB == '(') return 14;
    if (cA == '(' && cB == '.') return 15;
    if (cA == '(' && EsUnOperador(cB)) return 16;
    if (cA == '(' && cB == ')') return 17;
    if (cA == ')' && Char.IsDigit(cB)) return 18;
    if (cA == ')' && cB == '.') return 19;
    if (cA == ')' && Char.IsLower(cB)) return 20;
    if (cA == ')' && cB == '(') return 21;
}

/* Valida el inicio y fin de la expresión */
if (EsUnOperador(sbSintax[1])) return 22;
if (EsUnOperador(sbSintax[sbSintax.Length - 2])) return 23;

```



```

/* Valida balance de paréntesis */
int ParentesisAbre = 0; /* Contador de paréntesis que abre */
int ParentesisCierra = 0; /* Contador de paréntesis que cierra */
for (int Cont = 1; Cont < sbSyntax.Length - 1; Cont++) {
    switch (sbSyntax[Cont]) {
        case '(': ParentesisAbre++; break;
        case ')': ParentesisCierra++; break;
    }
    if (ParentesisCierra > ParentesisAbre) return 24;
}
if (ParentesisAbre != ParentesisCierra) return 25;

/* Validar los puntos decimales de un número real */
int TotalPuntos = 0;
for (int Cont = 0; Cont < sbSyntax.Length; Cont++) {
    if (EsUnOperador(sbSyntax[Cont])) TotalPuntos = 0;
    if (sbSyntax[Cont] == '.') TotalPuntos++;
    if (TotalPuntos > 1) return 26;
}

/* Deja la expresión para ser analizada.
 * Reemplaza las funciones de tres letras
 * por una letra mayúscula. Cambia los ))
 * por )+0) porque es requerido al crear las piezas */
this.Analizada.Length = 0;
this.Analizada.Append(ConLetrasValidas);
this.Analizada.Replace("sen", "A");
this.Analizada.Replace("cos", "B");
this.Analizada.Replace("tan", "C");
this.Analizada.Replace("abs", "D");
this.Analizada.Replace("asn", "E");
this.Analizada.Replace("acs", "F");
this.Analizada.Replace("atn", "G");
this.Analizada.Replace("log", "H");
this.Analizada.Replace("exp", "I");
this.Analizada.Replace("sqr", "J");

/* Sintaxis correcta */
return 0;
}

/* Retorna si el Caracter es un operador matemático */
private static bool EsUnOperador(char Caracter) {
    switch (Caracter) {
        case '+':
        case '-':
        case '*':
        case '/':
    }
}

```

```
case T ^ T :  
    return true;  
default:  
    return false;
```

```
}
```

```
}
```

```
}
```

```
}
```

Como usar el evaluador de expresiones

En Visual Studio 2022, se añade la clase al proyecto:

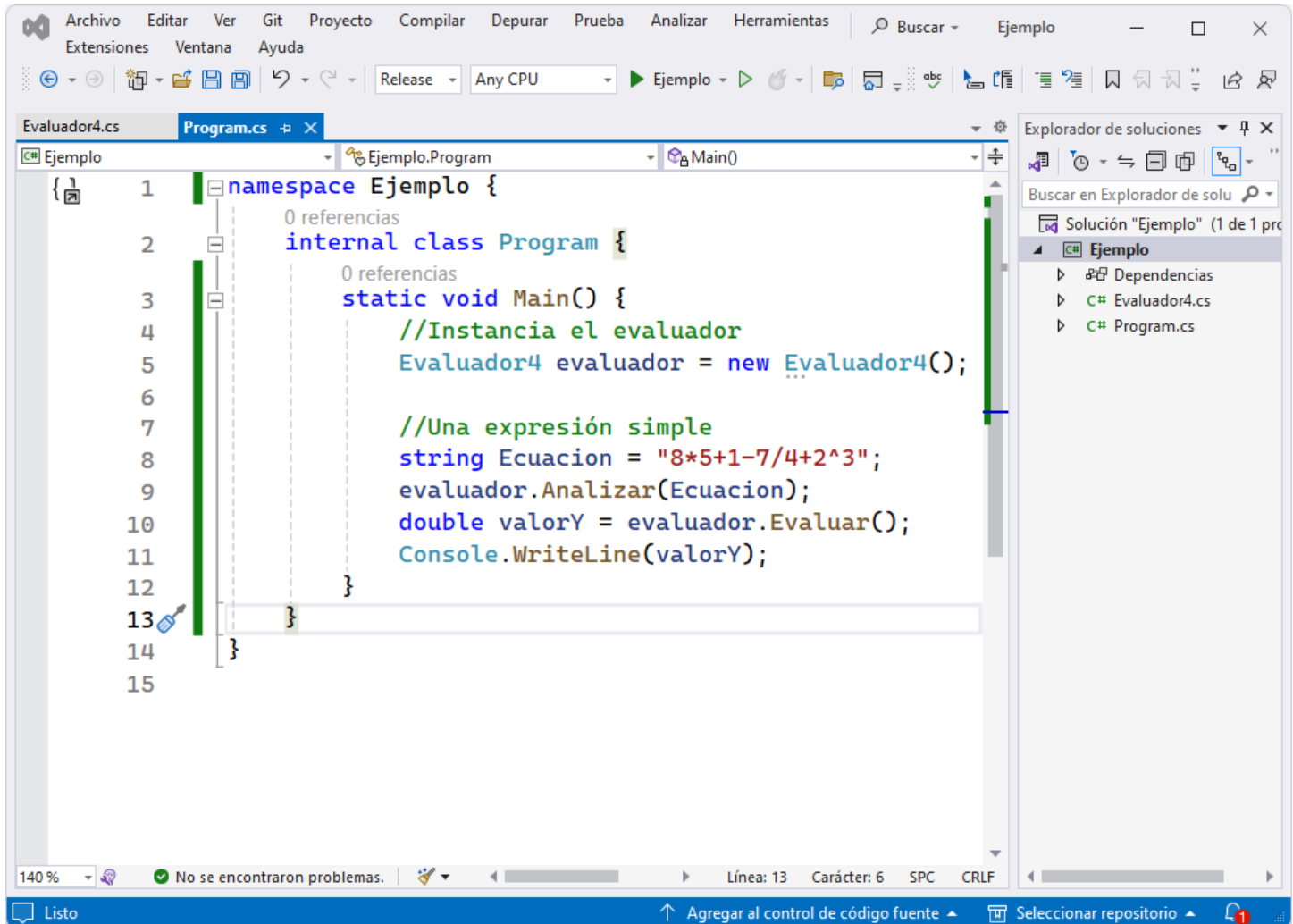


Ilustración 1: Uso del evaluador de expresiones

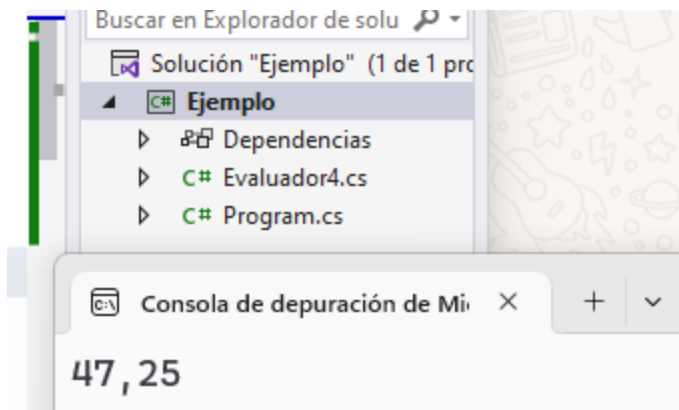


Ilustración 2: Resultado obtenido con el evaluador

G/001.cs

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            //Instancia el evaluador  
            Evaluador4 evaluador = new();  
  
            //Una expresión simple  
            string Ecuacion = "8*5+1-7/4+2^3";  
            evaluador.Analizar(Ecuacion);  
            double valorY = evaluador.Evaluar();  
            Console.WriteLine(valorY);  
        }  
    }  
}
```

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            //Instancia el evaluador  
            Evaluador4 evaluador = new();  
  
            //Una expresión simple con números reales  
            string Ecuacion = "7.318+5.0045-9.071^2*8.04961";  
            evaluador.Analizar(Ecuacion);  
            double valorY = evaluador.Evaluar();  
            Console.WriteLine(valorY);  
        }  
    }  
}
```

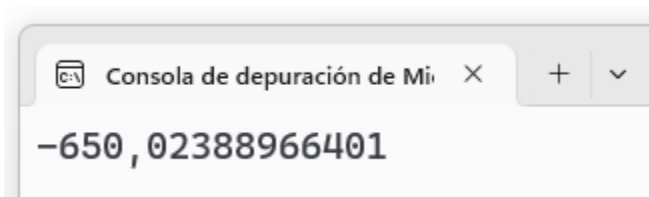


Ilustración 3: Operando números reales

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            //Instancia el evaluador  
            Evaluador4 evaluador = new();  
  
            //Una expresión con paréntesis  
            string Ecuacion = "5*(3+2.5)";  
            Ecuacion += "-7/(8.03*2-5^3)";  
            Ecuacion += "+4.1*(3-(5.8*2.3))";  
            evaluador.Analizar(Ecuacion);  
            double valorY = evaluador.Evaluar();  
            Console.WriteLine(valorY);  
        }  
    }  
}
```

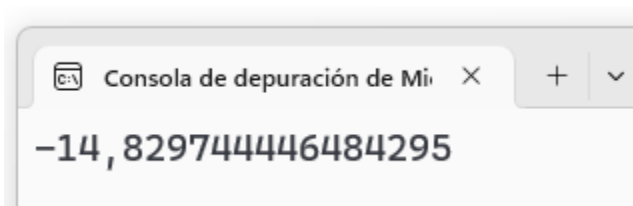


Ilustración 4: Operando con paréntesis

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            //Instancia el evaluador  
            Evaluador4 evaluador = new();  
  
            //Una expresión con funciones  
            string Ecuacion = "sen(4.90+2.34)-cos(1.89)";  
            Ecuacion += "*tan(3)/abs(4-12)+asn(0.12)";  
            Ecuacion += "-acs(0-0.4)+atn(0.03)*log(1.3)";  
            Ecuacion += "+cei(3.4)+exp(2.8)-sqr(9)";  
            evaluador.Analizar(Ecuacion);  
            double valorY = evaluador.Evaluar();  
            Console.WriteLine(valorY);  
        }  
    }  
}
```

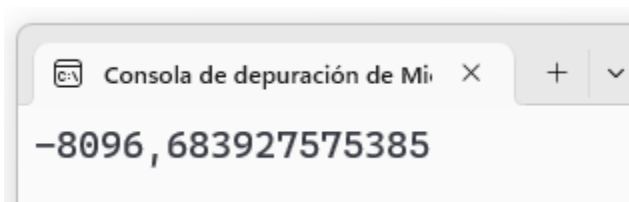


Ilustración 5: Operando con funciones

```
namespace Ejemplo {  
    internal class Program {  
        static void Main() {  
            //Instancia el evaluador  
            Evaluador4 evaluador = new();  
  
            //Una expresión con uso de variables  
            //(deben estar en minúsculas)  
            string Ecuacion = "3*x+2*y";  
            evaluador.Analizar(Ecuacion);  
  
            //Le da valor a las variables  
            //(deben estar en minúsculas)  
            evaluador.DarValorVariable('x', 1.8);  
            evaluador.DarValorVariable('y', 3.5);  
            double valorY = evaluador.Evaluar();  
            Console.WriteLine(valorY);  
        }  
    }  
}
```

Ilustración 6: Operando con uso de variables


```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            Random Azar = new();

            //Instancia el evaluador
            Evaluador4 evaluador = new();

            //Una expresión con uso de variables
            //(deben estar en minúsculas)
            string Ecuacion = "3*cos(2*x+4)-5*sen(4*y-7)";

            //Se analiza primero la ecuación
            evaluador.Analizar(Ecuacion);

            //Después de ser analizada, se le dan los
            //valores a las variables, esto hace que
            //el evaluador sea muy rápido
            for (int cont = 1; cont <= 20; cont++) {
                evaluador.DarValorVariable('x', Azar.NextDouble());
                evaluador.DarValorVariable('y', Azar.NextDouble());
                double valorY = evaluador.Evaluar();
                Console.WriteLine(valorY);
            }
        }
    }
}
```

```
Consola de depuración de Mi X
-0,4197910124938974
0,22744472406067784
0,4930701866765128
0,604453559582212
1,0317579190273118
-1,0981087571111874
-0,6594478278023597
-3,718174689910168
-2,3470909725871536
-5,406657113170899
-1,9365223469480537
-4,713506913008152
-5,446409726256411
-5,530778422625837
-3,332409299172522
-1,982683328184644
3,303251362003876
0,7223984492676445
3,1800843027444268
-2,821154224259754
```

Ilustración 7: Generar múltiples valores de una ecuación al cambiar los valores.

Validando la sintaxis

El evaluador 4.0 verifica si la expresión algebraica es sintácticamente correcta al llamar al método Analizar(), si este método retorna el valor cero, significa que la expresión es correcta o no tiene errores de sintaxis, caso contrario, retorna un código de error.

G/007.cs

```
namespace Ejemplo {
    internal class Program {
        static void Main() {
            string[] expAlg = new string[] {
                "2q-(*3)", //0
                "7-2(5-6)", //1
                "3..1", //2
                "3.*1", //3
                "3+5.w-8", //4
                "2-5.(4+1)*3", //5
                "2-(5.)*3", //6
                "2-(4+.1)-7", //7
                "5-*3", //8
                "2-(4+)-7", //9
                "7-a2-6", //10
                "7-a.4*3", //11
                "7-qw*9", //12
                "2-u(7-3)", //13
                "7-(.8+4)-6", //14
                "(+3-5)*7", //15
                "4+()*2", //16
                "(3-5)8", //17
                "(3-5).+2", //18
                "2-(7*3)k+7", //19
                "(4-3)(2+1)", //20
                "*3+5", //21
                "3*5*", //22
                "9*4)+(2-6", //23
                "((2+4)", //24
                "2.71*3.56.01" }; //25

            Evaluador4 evaluador = new();

            for (int num = 0; num < expAlg.Length; num++) {
                Console.WriteLine("\r\nExpr " + num + ": " + expAlg[num]);
                int Sintaxis = evaluador.Analizar(expAlg[num]);
                if (Sintaxis >= 0) {
                    Console.WriteLine(evaluador.MensajeError(Sintaxis));
                }
            }
        }
    }
}
```

```
}  
}  
}
```

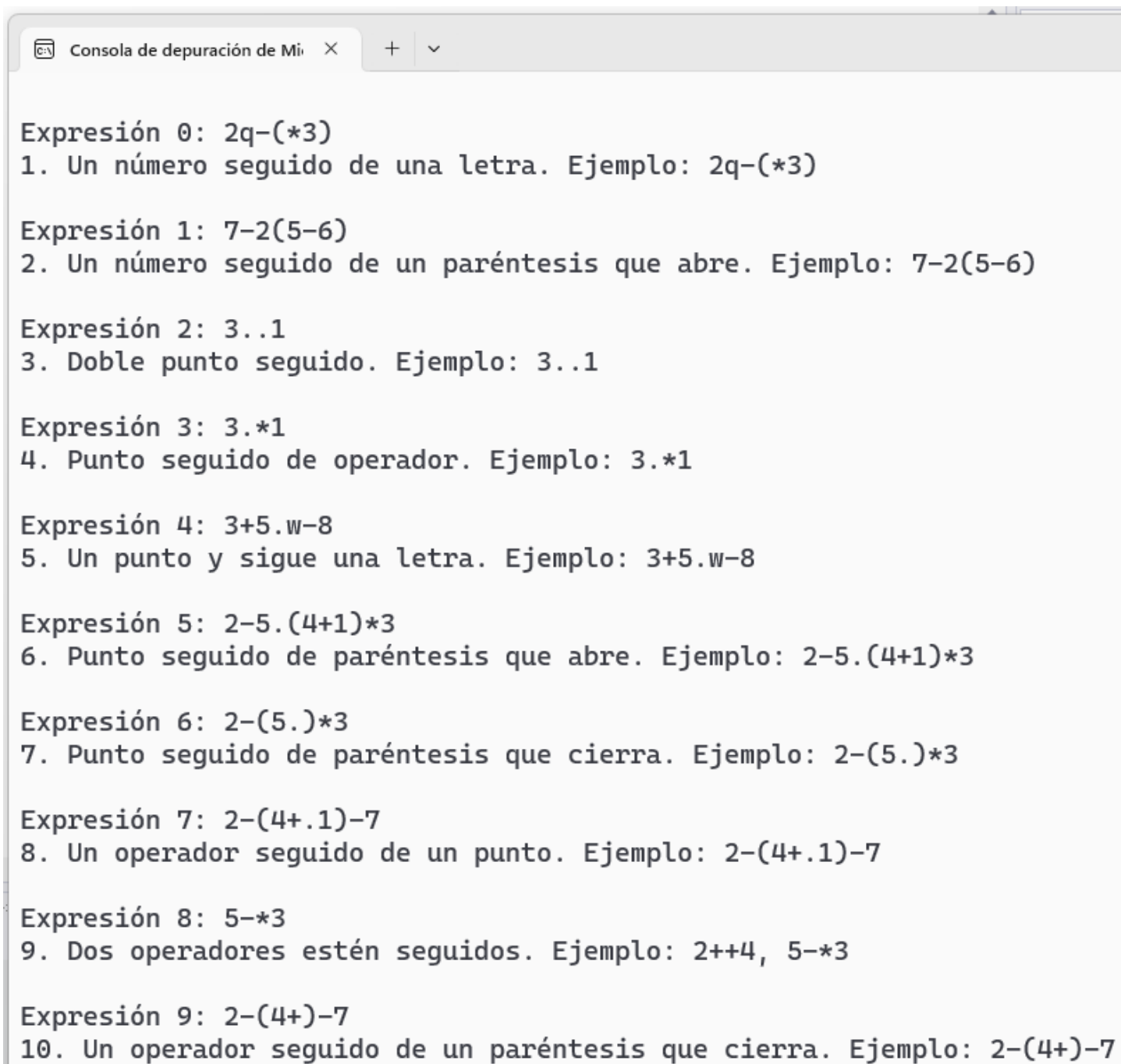


Ilustración 8: Evaluando la sintaxis

Expresión 10: $7-a^2-6$

11. Una letra seguida de número. Ejemplo: $7-2a-6$

Expresión 11: $7-a.4*3$

12. Una letra seguida de punto. Ejemplo: $7-a.-6$

Expresión 12: $7-qw*9$

13. Una letra seguida de otra letra. Ejemplo: $4-xy+3$

Expresión 13: $2-u(7-3)$

14. Una letra seguida de un paréntesis que abre. Ejemplo: $2-a(8*3)$

Expresión 14: $7-(.8+4)-6$

15. Un paréntesis que abre seguido de un punto. Ejemplo: $7-(.8+4)-6$

Expresión 15: $(+3-5)*7$

16. Un paréntesis que abre y sigue un operador. Ejemplo: $(+3-5)*7$

Expresión 16: $4+()*2$

17. Un paréntesis que abre y sigue un paréntesis que cierra. Ejemplo: $4+()*2$

Expresión 17: $(3-5)8$

18. Un paréntesis que cierra y sigue un número. Ejemplo: $(3-5)8$

Expresión 18: $(3-5).+2$

19. Un paréntesis que cierra y sigue un punto. Ejemplo: $(3-5).+2$

Expresión 19: $2-(7*3)k+7$

20. Un paréntesis que cierra y sigue una letra. Ejemplo: $2-(7*3)k+7$

Ilustración 9: Evaluando la sintaxis

Expresión 20: $(4-3)(2+1)$

21. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: $(4-3)(2+1)$

Expresión 21: $*3+5$

22. Inicia con un operador. Ejemplo: $*3+5$

Expresión 22: $3*5*$

23. Finaliza con un operador. Ejemplo: $7+9*$

Expresión 23: $9*4)+(2-6$

24. No hay correspondencia entre paréntesis que cierran y abren

Expresión 24: $((2+4)$

25. El número de paréntesis que cierran no es igual al número de paréntesis que abren

Expresión 25: $2.71*3.56.01$

26. Dos o más puntos en número real

Ilustración 10: Evaluando la sintaxis

Métricas: Desempeño con el evaluador interno de C#

.NET 8 tiene un evaluador de expresiones propio, así que una forma de probar que el evaluador está funcionando bien es generando ecuaciones al azar y comparar los resultados entre el evaluador propio de C# y el evaluador 4.0, si son iguales, significa que todo marcha bien. A continuación, el código que genera ecuaciones al azar y usa ambos evaluadores, compara los resultados y además el tiempo que tarda uno y otro.

G/008.cs

```
using System.Data;
using System.Diagnostics;

namespace Ejemplo {
    internal class Program {
        static void Main() {
            Random Azar = new();

            //Versión 2024
            Evaluador4 evaluador2024 = new();

            //Arreglos que guardan valores de X, Y, Z
            double[] arregloX = new double[200];
            double[] arregloY = new double[200];
            double[] arregloZ = new double[200];
            for (int cont = 0; cont < arregloX.Length; cont++) {
                arregloX[cont] = Azar.NextDouble();
                arregloY[cont] = Azar.NextDouble();
                arregloZ[cont] = Azar.NextDouble();
            }

            //Prueba evaluador
            long TotalTiempo2024Evalua = 0, TotalTiempo2024Analiza = 0;
            double valor2024, AcumValor2024 = 0;

            //Evaluador interno
            long TotalTiempoInterno = 0;
            double valorInterno, AcumInterno = 0;

            //Toma el tiempo
            Stopwatch temporizador = new();

            for (int num = 1; num <= 1000; num++) {
                string ecuacion = EcuacionAzar(350, Azar);
                //Console.WriteLine(ecuacion);

                //Versión 2024. Análisis.
                temporizador.Reset();
```

```

temporizador.Start();
evaluador2024.Analizar(ecuacion);
temporizador.Stop();
TotalTiempo2024Analiza += temporizador.ElapsedTicks;

//Versión 2024. Evaluación
temporizador.Reset();
temporizador.Start();
valor2024 = 0;
for (int cont = 0; cont < arregloX.Length; cont++) {
    evaluador2024.DarValorVariable('x', arregloX[cont]);
    evaluador2024.DarValorVariable('y', arregloY[cont]);
    evaluador2024.DarValorVariable('z', arregloZ[cont]);
    valor2024 += Math.Abs(evaluador2024.Evaluar());
}
temporizador.Stop();
TotalTiempo2024Evalua += temporizador.ElapsedTicks;

//Compara contra el evaluador de
//expresiones propio que tiene C#
var EvaluadorInterno = new DataTable();
temporizador.Reset();
temporizador.Start();
valorInterno = 0;
for (int cont = 0; cont < arregloX.Length; cont++) {
    string ec1 = ecuacion;
    string ec2 = ec1.Replace("x", arregloX[cont].ToString());
    string ec3 = ec2.Replace("y", arregloY[cont].ToString());
    string ec4 = ec3.Replace("z", arregloZ[cont].ToString());
    string ec5 = ec4.Replace(",", ".");
    var Result = EvaluadorInterno.Compute(ec5, "");
    valorInterno += Math.Abs(Convert.ToDouble(Result));
}
temporizador.Stop();
TotalTiempoInterno += temporizador.ElapsedTicks;

if (Math.Abs(valor2024) > 10000000) continue;
if (double.IsNaN(valor2024) || double.IsInfinity(valor2024))
    continue;

AcumValor2024 += valor2024;
AcumInterno += valorInterno;
}
Console.WriteLine("Evaluador 2024 acumula: " + AcumValor2024);
Console.WriteLine("Interno C# acumula: " + AcumInterno);

Console.Write("\r\nEvaluador 2024 tiempo para evaluar: ");
Console.WriteLine(TotalTiempo2024Evalua);

```



```

Console.Write("Evaluador 2024 tiempo para analizar: ");
Console.WriteLine(TotalTiempo2024Analiza);

long TotalTiempo = TotalTiempo2024Evalua + TotalTiempo2024Analiza;

Console.Write("\r\nEvaluador 2024 tiempo: analizar y evaluar: ");
Console.WriteLine(TotalTiempo);

Console.Write("Interno tiempo: analizar y evaluar: ");
Console.WriteLine(TotalTiempoInterno);
}

public static string EcuacionAzar(int Longitud, Random Azar) {
    int cont = 0;
    int numParentesisAbre = 0;

    string Ecuacion = "";
    while (cont < Longitud) {

        //Función o paréntesis o nada
        if (Azar.NextDouble() < 0.5) {
            Ecuacion += "(";
            numParentesisAbre++;
            cont++;
        }

        //Variable o número
        cont++;
        switch (Azar.Next(4)) {
            case 0: Ecuacion += NumeroAzar(Azar); break;
            case 1: Ecuacion += "x"; break;
            case 2: Ecuacion += "y"; break;
            case 3: Ecuacion += "z"; break;
        }

        //Paréntesis que cierra
        int numParentesisCierra = Azar.Next(numParentesisAbre + 1);
        for (int num = 1; num <= numParentesisCierra; num++) {
            Ecuacion += ")";
            numParentesisAbre--;
            cont++;
        }

        //Operador
        cont++;
        Ecuacion += OperadorAzar(Azar);
    }
}

```

```

//Variable o número
switch (Azar.Next(4)) {
    case 0: Ecuacion += NumeroAzar(Azar); break;
    case 1: Ecuacion += "x"; break;
    case 2: Ecuacion += "y"; break;
    case 3: Ecuacion += "z"; break;
}

for (int num = 0; num < numParentesisAbre; num++)
    Ecuacion += ")";

return Ecuacion;
}

private static string OperadorAzar(Random azar) {
    string[] operadores = { "+", "-", "*" };
    return operadores[azar.Next(operadores.Length)];
}

private static string NumeroAzar(Random azar) {
    return "0." + Convert.ToString(azar.Next(1000000) + 1);
}
}
}

```

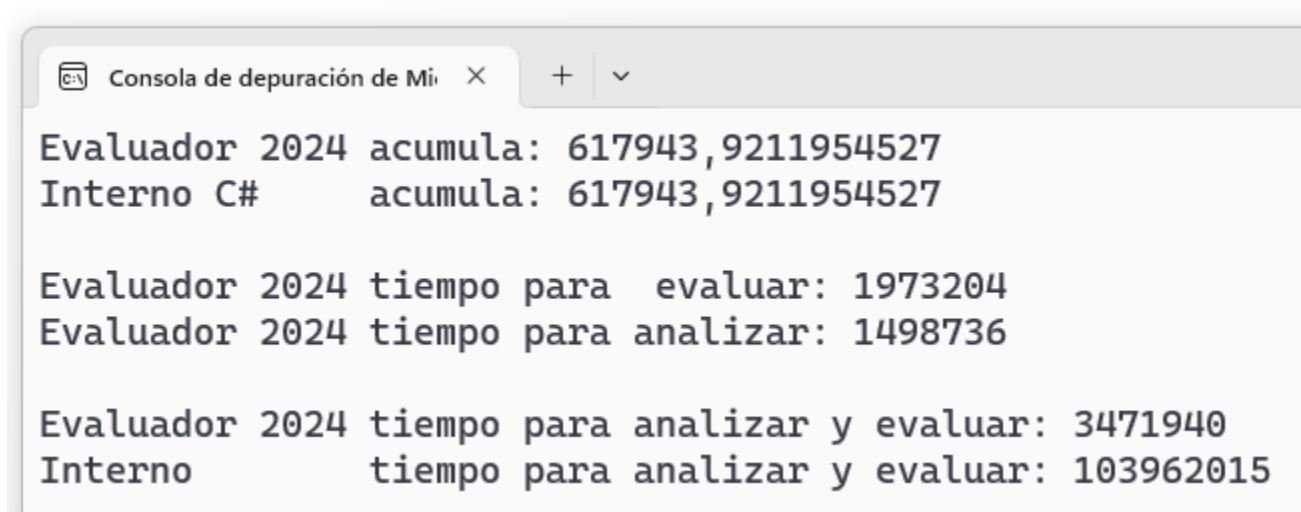
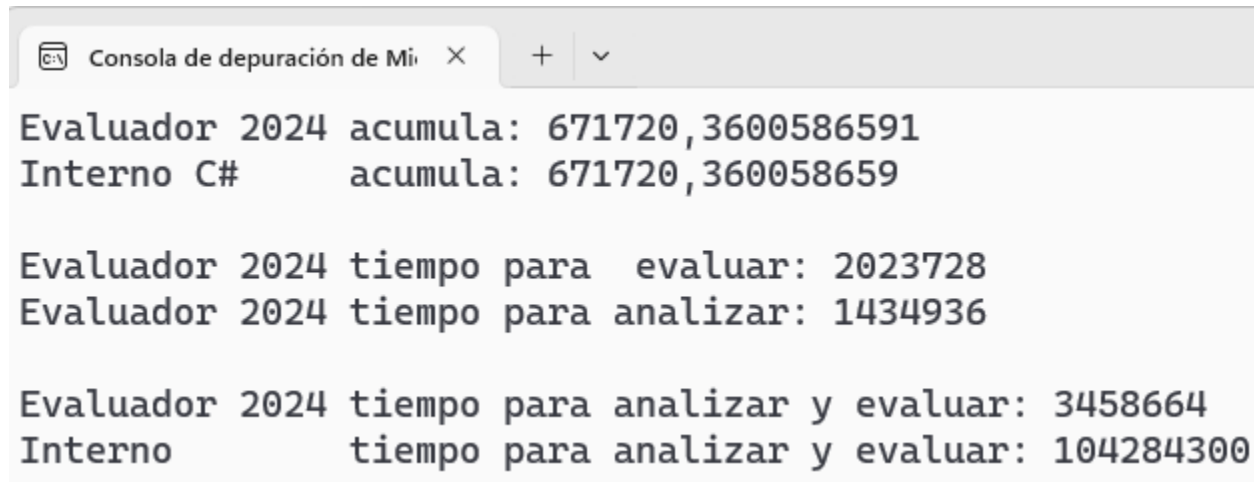


Ilustración 11: Métrica compara ambos evaluadores

Como se puede observar, ambos evaluadores obtienen los mismos resultados evaluando 1000 ecuaciones, cada una del tamaño de 350 caracteres y 300 valores distintos de variables por ecuación. Luego está correcto.

En el lado de desempeño, el Evaluador 4.0 es más rápido que el evaluador interno. La explicación a esta diferencia tan alta (a favor del Evaluador 4) es que el evaluador está diseñado, escrito y optimizado sólo para ecuaciones algebraicas, mientras el evaluador interno de C# es más genérico para diversos tipos de expresiones. La especialización vence.

Otras pruebas:



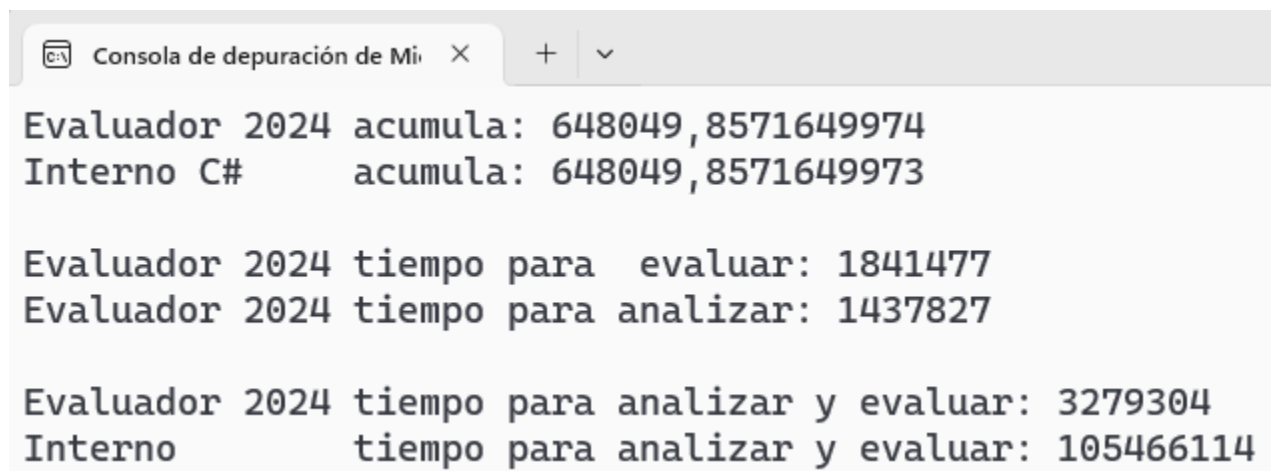
```
Consola de depuración de Mi... X + v

Evaluador 2024 acumula: 671720,3600586591
Interno C#      acumula: 671720,360058659

Evaluador 2024 tiempo para evaluar: 2023728
Evaluador 2024 tiempo para analizar: 1434936

Evaluador 2024 tiempo para analizar y evaluar: 3458664
Interno        tiempo para analizar y evaluar: 104284300
```

Ilustración 12: Métrica compara ambos evaluadores



```
Consola de depuración de Mi... X + v

Evaluador 2024 acumula: 648049,8571649974
Interno C#      acumula: 648049,8571649973

Evaluador 2024 tiempo para evaluar: 1841477
Evaluador 2024 tiempo para analizar: 1437827

Evaluador 2024 tiempo para analizar y evaluar: 3279304
Interno        tiempo para analizar y evaluar: 105466114
```

Ilustración 13: Métrica compara ambos evaluadores