

28-4-2021

Evaluator de expresiones en nueve lenguajes de programación

En C#, C++, Delphi, Java, JavaScript, PHP, Python, TypeScript y Visual Basic .NET

$1.6 * r^2 + \text{sen}(8.3 - 2 * m)$

1.6 * r ^ 2 + sen (8.3 - 2 * m)

| Pieza | Función | A | operador | B |
|-------|---------|-----|----------|-----|
| [0] | | 2 | * | m |
| [1] | sen | 8.3 | - | [0] |
| [2] | | r | ^ | 2 |
| [3] | | 1.6 | * | [2] |
| [4] | | [3] | + | [1] |

Rafael Alberto Moreno Parra

Otros libros del autor

Libro 18: "C#: Árboles binarios, n-arios, grafos, listas simple y doblemente enlazadas". En Colombia 2021. Págs. 63. Libro y código fuente descargable en: <https://github.com/ramsoftware/C-Sharp-Arboles>

Libro 17: "C#. Estructuras dinámicas de memoria". En Colombia 2021. Págs. 82. Libro y código fuente descargable en: <https://github.com/ramsoftware/CSharpDinamica>

Libro 16: "C#. Programación Orientada a Objetos". En Colombia 2020. Págs. 90. Libro y código fuente descargable en: <https://github.com/ramsoftware/C-Sharp-POO>

Libro 15: "C#. Estructuras básicas de memoria.". En Colombia 2020. Págs. 60. Libro y código fuente descargable en: <https://github.com/ramsoftware/EstructuraBasicaMemoriaCSharp>

Libro 14: "Iniciando en C#". En Colombia 2020. Págs. 72. Libro y código fuente descargable en: <https://github.com/ramsoftware/C-Sharp-Iniciando>

Libro 13: "Algoritmos Genéticos". En Colombia 2020. Págs. 62. Libro y código fuente descargable en: <https://github.com/ramsoftware/LibroAlgoritmoGenetico2020>

Libro 12: "Redes Neuronales. Segunda Edición". En Colombia 2020. Págs. 108. Libro y código fuente descargable en: <https://github.com/ramsoftware/LibroRedNeuronal2020>

Libro 11: "Capacitándose en JavaScript". En Colombia 2020. Págs. 317. Libro y código fuente descargable en: <https://github.com/ramsoftware/JavaScript>

Libro 10: "Desarrollo de aplicaciones para Android usando MIT App Inventor 2". En Colombia 2016. Págs. 102. Ubicado en: <https://openlibra.com/es/book/desarrollo-de-aplicaciones-para-android-usando-mit-app-inventor-2>

Libro 9: "Redes Neuronales. Parte 1.". En Colombia 2016. Págs. 90. Libro descargable en: <https://openlibra.com/es/book/redes-neuronales-parte-1>

Libro 8: "Segunda parte de uso de algoritmos genéticos para la búsqueda de patrones". En Colombia 2015. Págs. 303. En publicación por la Universidad Libre – Cali.

Libro 7: "Desarrollo de un evaluador de expresiones algebraicas. **Versión 2.0.** C++, C#, Visual Basic .NET, Java, PHP, JavaScript y Object Pascal (Delphi)". En: Colombia 2013. Págs. 308. Ubicado en: <https://openlibra.com/es/book/evaluador-de-expresiones-algebraicas-ii>

Libro 6: "Un uso de algoritmos genéticos para la búsqueda de patrones". En Colombia 2013. En publicación por la Universidad Libre – Cali.

Libro 5: Desarrollo fácil y paso a paso de aplicaciones para Android usando MIT App Inventor. En Colombia 2013. Págs. 104. Estado: Obsoleto (No hay enlace).

Libro 4: "Desarrollo de un evaluador de expresiones algebraicas. C++, C#, Visual Basic .NET, Java, PHP, JavaScript y Object Pascal (Delphi)". En Colombia 2012. Págs. 308. Ubicado en: <https://openlibra.com/es/book/evaluador-de-expresiones-algebraicas>

Libro 3: "Simulación: Conceptos y Programación" En Colombia 2012. Págs. 81. Ubicado en: <https://openlibra.com/es/book/simulacion-conceptos-y-programacion>

Libro 2: "Desarrollo de videojuegos en 2D con Java y Microsoft XNA". En Colombia 2011. Págs. 260. Ubicado en: <https://openlibra.com/es/book/desarrollo-de-juegos-en-2d-usando-java-y-microsoft-xna> . ISBN: 978-958-8630-45-8

Libro 1: "Desarrollo de gráficos para PC, Web y dispositivos móviles" En Colombia 2009. ed.: Artes Gráficas Del Valle Editores Impresores Ltda. ISBN: 978-958-8308-95-1 v. 1 págs. 317

Artículo: "Programación Genética: La regresión simbólica". Entramado ISSN: 1900-3803 ed.: Universidad Libre Seccional Cali v.3 fasc.1 p.76 - 85, 2007

Página web del autor y canal en Youtube

Investigación sobre Vida Artificial: <http://darwin.50webs.com>

Canal en Youtube: <http://www.youtube.com/user/RafaelMorenoP> (dedicado principalmente al desarrollo en C#)

Sitio en GitHub

El código fuente se puede descargar en <https://github.com/ramsoftware/Evaluador3>

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 2019 ® Enlace: <https://visualstudio.microsoft.com/es/vs/>

A mis padres, a mi hermana....

Y a mi tropa gatuna: Sally, Suini, Vikingo, Grisú, Capuchina, Milú y mi recordada Tammy.

Contenido

Otros libros del autor 1

Página web del autor y canal en Youtube 2

Sitio en GitHub 2

Licencia del software 2

Marcas registradas 2

Dedicatoria 3

Introducción..... 6

El algoritmo 8

 El problema 8

 Etapa 1: Retirar espacios, tabuladores y convertirla a minúsculas..... 8

 Etapa 2: Verificando la sintaxis de la expresión algebraica 8

 Etapa 3: Transformando la expresión..... 8

 Etapa 4: Dividiendo la cadena en partes..... 9

 Etapa 5: Generando las Piezas desde las Partes..... 10

 Paso 1: 10

 Paso 2: 10

 Paso 3: 10

 Paso 4: 10

 Paso 5: 11

 Paso 6: 11

 Etapa 6: Evaluando a partir de las Piezas 11

 Paso 0: 11

 Paso 1: 11

 Paso 2: 12

 Paso 3: 12

 Paso 4: 12

 Paso 5: 12

 Escrito para velocidad de evaluación..... 13

C#..... 15

 Clase EvaluaSintaxis 15

 Clase Parte..... 21

 Clase Pieza..... 22

 Clase Evaluador3..... 23

 Como usar el evaluador..... 27

C++ 30

 Clase EvaluaSintaxis 30

 Clase Parte..... 37

 Clase Pieza..... 38

 Clase Evaluador3..... 39

 Como usar el evaluador..... 44

Delphi 48

 Clase EvaluaSintaxis 48

 Clase Parte..... 58

 Clase Pieza..... 59

 Clase Evaluador3..... 60

 Como usar el evaluador..... 65

Java..... 69

 Clase EvaluaSintaxis 69

 Clase Parte..... 75

 Clase Pieza..... 76

 Clase Evaluador3..... 77

 Como usar el evaluador..... 81

JavaScript..... 84

 Clase EvaluaSintaxis 84

Clase Parte..... 90

Clase Pieza..... 91

Clase Evaluador3..... 92

Como usar el evaluador..... 96

PHP 99

 Clase EvaluaSintaxis 99

 Clase Parte.....105

 Clase Pieza.....106

 Clase Evaluador3.....107

 Como usar el evaluador.....111

Python114

 Clase EvaluaSintaxis114

 Clase Parte.....120

 Clase Pieza.....121

 Clase Evaluador3.....122

 Como usar el evaluador.....126

TypeScript129

 Clase EvaluaSintaxis129

 Clase Parte.....135

 Clase Pieza.....136

 Clase Evaluador3.....137

 Como usar el evaluador.....141

Visual Basic .NET144

 Clase EvaluaSintaxis144

 Clase Parte.....151

 Clase Pieza.....152

 Clase Evaluador3.....153

 Como usar el evaluador.....157

Introducción

En 2013 escribí el libro “Evaluador de expresiones 2.0”, han pasado 8 años desde entonces y decidí revisar de nuevo este software por varias razones:

1. Algunos lenguajes de programación han cambiado considerablemente como es el caso de JavaScript y PHP.
2. Abordé dos lenguajes de programación nuevos: Python y TypeScript.
3. Consideré que el código podría ser más sencillo y legible.
4. Se podría optimizar parte del código para que la evaluación de la expresión algebraica fuese más rápida.
5. Quitar la confusión que algunos lectores me advirtieron con respecto al uso del menos unario en el evaluador del 2013.
6. Mejorar los ejemplos de uso del código al usar entornos de desarrollo.
7. Explicar mejor la parte algorítmica.
8. Poner el código para fácil acceso desde GitHub.
9. Como apoyo a la clase de Estructura de Datos que dicto en la Universidad. El algoritmo hace uso de estructuras dinámicas de memoria.
10. Usualmente se requiere tomar una expresión algebraica y evaluarla múltiples veces, por ejemplo, para hacer un gráfico matemático como $Y=F(X)$. En ese caso, lo que cambia es el valor de X , pero la expresión es la misma. Luego este evaluador fue escrito para hacer más rápido el obtener continuos valores al variar los valores de las variables independientes.

Un evaluador de expresiones algebraicas es un algoritmo que toma una expresión algebraica almacenada en una cadena o string y es capaz de interpretarla.

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, raíz cúbica).

El libro se dividirá en los siguientes capítulos:

1. Explicación algorítmica del evaluador de expresiones
2. Por cada lenguaje de programación habrá un capítulo.

El código fuente se puede descargar en: <https://github.com/ramsoftware/Evaluador3>

Algoritmo

El algoritmo

El problema

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.

Etapas 1: Retirar espacios, tabuladores y convertirla a minúsculas.

Como la expresión algebraica ha sido digitada por un usuario final, será necesario hacerle varios cambios: quitar los espacios y tabuladores y luego volverla a minúsculas.

Etapas 2: Verificando la sintaxis de la expresión algebraica

Posteriormente, es necesario verificar si esta cumple con las estrictas reglas sintácticas del álgebra. Se hacen 27 validaciones que son:

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

Etapas 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)*sen(k^x)"

Pasa a: "(0.004–(1.78/3.45+h)*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)*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 |
| Rcb | Raíz Cúbica | L |

Etapa 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))”

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

Para lograr eso, se hace uso de una lista dinámica. En cada nodo, estará un elemento de la expresión. La clase que define lo que debe llevar cada parte es la siguiente:

```
public class Parte {
    public int Tipo; /* Acumulador, función, paréntesis que abre, paréntesis que cierra, operador, número, variable */
    public int Funcion; /* 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 char Operador; /* + suma - resta * multiplicación / división ^ potencia */
    public double Numero; /* Número literal, por ejemplo: 3.141592 */
    public int UnaVariable; /* Variable algebraica */
    public int Acumulador; /* Usado cuando la expresión se convierte en piezas. Por ejemplo:
        3 + 2 / 5 se convierte así:
        |3| |+| |2| | / | |5|
        |3| |+| |A| A es un identificador de acumulador */

    public Parte(int tipo, int funcion, char operador, double numero, int unaVariable) {
        Tipo = tipo;
        Funcion = funcion;
        Operador = operador;
        Numero = numero;
        UnaVariable = unaVariable;
    }
}
```

Eta­pa 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

```
public class Pieza {
    public double ValorPieza; /* Almacena el valor que genera la pieza al evaluarse */
    public int Funcion; /* 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 TipoA; /* La primera parte es un número o una variable o trae el valor de otra pieza */
    public double NumeroA; /* Es un número literal */
    public int VariableA; /* Es una variable */
    public int PiezaA; /* Trae el valor de otra pieza */
    public char Operador; /* + suma - resta * multiplicación / división ^ potencia */
    public int TipoB; /* La segunda parte es un número o una variable o trae el valor de otra pieza */
    public double NumeroB; /* Es un número literal */
    public int VariableB; /* Es una variable */
    public int PiezaB; /* Trae el valor de otra pieza */

    public Pieza(int funcion, int tipoA, double numeroA, int variableA, int piezaA, char operador, int tipoB,
double numeroB, int variableB, int piezaB) {
        Funcion = funcion;

        TipoA = tipoA;
        NumeroA = numeroA;
        VariableA = variableA;
        PiezaA = piezaA;

        Operador = operador;

        TipoB = tipoB;
        NumeroB = numeroB;
        VariableB = variableB;
        PiezaB = piezaB;
    }
}
```

Esas Piezas se construyen desde las Partes. Esta sería la operación paso a paso:

Paso 1:

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

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------|---------|--------|----------|--------|
| | | | | |

Paso 2:

| | | | | | | | | | | | | |
|---|-------|---|---|------|---|------|---|---|---|---|-----|---|
| (| 0.004 | - | (| 1.78 | / | 3.45 | + | h |) | * | [0] |) |
|---|-------|---|---|------|---|------|---|---|---|---|-----|---|

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------|---------|--------|----------|--------|
| [0] | A | K | ^ | x |

Paso 3:

| | | | | | | | | | | |
|---|-------|---|---|-----|---|---|---|---|-----|---|
| (| 0.004 | - | (| [1] | + | h |) | * | [0] |) |
|---|-------|---|---|-----|---|---|---|---|-----|---|

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------|---------|--------|----------|--------|
| [0] | A | K | ^ | x |
| [1] | | 1.78 | / | 3.45 |

Paso 4:

| | | | | | | |
|---|-------|---|-----|---|-----|---|
| (| 0.004 | - | [2] | * | [0] |) |
|---|-------|---|-----|---|-----|---|

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------|---------|--------|----------|--------|
| [0] | A | k | ^ | x |
| [1] | | 1.78 | / | 3.45 |
| [2] | | [1] | + | h |

Paso 5:

| | | | | |
|---|-------|---|-----|---|
| (| 0.004 | - | [3] |) |
|---|-------|---|-----|---|

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------|---------|--------|----------|--------|
| [0] | A | k | ^ | x |
| [1] | | 1.78 | / | 3.45 |
| [2] | | [1] | + | h |
| [3] | | [2] | * | [0] |

Paso 6:

| |
|-----|
| [4] |
|-----|

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------|---------|--------|----------|--------|
| [0] | A | k | ^ | x |
| [1] | | 1.78 | / | 3.45 |
| [2] | | [1] | + | h |
| [3] | | [2] | * | [0] |
| [4] | | 0.004 | - | [3] |

Etapa 6: Evaluando a partir de las Piezas

Yendo de pieza en pieza se evalúa toda la expresión. Lo que hay que hacer es poner los valores de las variables, el resultado de la operación Función|Parte1|Operador|Parte2 se guarda en la Pieza.

Los valores de las variables, por ejemplo:

k = 3

x = 2

h = 5

Paso 0:

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------|---------|--------|----------|--------|
| [0] | A | k 3 | ^ | x 2 |
| [1] | | 1.78 | / | 3.45 |
| [2] | | [1] | + | h |
| [3] | | [2] | * | [0] |
| [4] | | 0.004 | - | [3] |

Paso 1:

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------------------|---------|--------|----------|--------|
| [0] 0.41211848 | A | k 3 | ^ | x 2 |
| [1] | | 1.78 | / | 3.45 |
| [2] | | [1] | + | h |
| [3] | | [2] | * | [0] |
| [4] | | 0.004 | - | [3] |

Paso 2:

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------------------|---------|--------|----------|--------|
| [0] 0.41211848 | A | k 3 | ^ | x 2 |
| [1] 0.51594202 | | 1.78 | / | 3.45 |
| [2] | | [1] | + | h |
| [3] | | [2] | * | [0] |
| [4] | | 0.004 | - | [3] |

Paso 3:

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------------------|---------|-------------------|----------|--------|
| [0] 0.41211848 | A | k 3 | ^ | x 2 |
| [1] 0.51594202 | | 1.78 | / | 3.45 |
| [2] 5.51594202 | | [1] 0.51594202 | + | h 5 |
| [3] | | [2] | * | [0] |
| [4] | | 0.004 | - | [3] |

Paso 4:

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------------------|-----------|-------------------|----------|-------------------|
| [0] 0.41211848 | A seno | k 3 | ^ | x 2 |
| [1] 0.51594202 | | 1.78 | / | 3.45 |
| [2] 5.51594202 | | [1] 0.51594202 | + | h 5 |
| [3] 2.27322167 | | [2] 5.51594202 | * | [0] 0.41211848 |
| [4] | | 0.004 | - | [3] |

Paso 5:

| Pieza | Función | Parte1 | Operador | Parte2 |
|-------------------|-----------|-------------------|----------|-------------------|
| [0] 0.41211848 | A seno | k 3 | ^ | x 2 |
| [1] 0.51594202 | | 1.78 | / | 3.45 |
| [2] 5.51594202 | | [1] 0.51594202 | + | h 5 |
| [3] 2.27322167 | | [2] 5.51594202 | * | [0] 0.41211848 |
| [4] -2.2692216 | | 0.004 | - | [3] 2.27322167 |

El resultado sería el valor obtenido en la última pieza: -2.2692216

Algunos puntos para tener en cuenta:

1. Cada vez que se evalúe una nueva expresión debe llamar al método Analizar(expresión);
2. El método Analizar(expresión) valida la sintaxis de la expresión. Este método debe retornar true para poder continuar con la evaluación, caso contrario, es que hay un error de sintaxis.
3. Si va a generar varios valores de una expresión, después de Analizar(expresión), se hace uso de Evaluar(); dentro del ciclo que genere los valores. Se cambia los valores de las variables con DarValorVariable(variable, valor);
4. Las variables de la expresión se tratan siempre en minúsculas.

Escrito para velocidad de evaluación

Si se quiere hacer la gráfica generada por una expresión, se requiere cambiar sólo los valores de las variables independientes, no es necesario analizar toda la expresión de nuevo. Esa es una de las ventajas de este evaluador: que analiza la expresión y posteriormente se pueden hacer a gran velocidad múltiples operaciones cambiando el valor de las variables independientes.

Veamos un ejemplo:

```
//Analiza la expresión: valida sintaxis, si todo está bien la divide en partes y luego crea las piezas
if (evaluador.Analizar(expresion)) {

    //Si el análisis retornó true entonces se puede evaluar múltiples veces la expresión

    //Evalúa rápidamente con ciclos gracias a que la expresión ya fue analizada.
    Random azar = new Random();
    for (int num = 1; num <= 100000; num++) {
        double valor = azar.NextDouble();
        evaluador.DarValorVariable('x', valor);
        resultado = evaluador.Evaluar();
        Console.WriteLine(resultado);
    }
}
```

Explicación de los métodos:

Analizar() . Hace una evaluación de la sintaxis de la expresión. Si todo es correcto en sintaxis, entonces toma la expresión, la divide en partes y luego crea las piezas dejando todo listo para ser evaluada.

DarValorVariable() . Modifica el valor de las variables que hay en la expresión previamente analizada correctamente.

Evaluar(). Evalúa la expresión y retorna el valor calculado.

¡OJO! Sólo se puede usar los dos últimos métodos si Analizar() ha retornado true.

¡OJO! Si se cambia la expresión entonces hay que llamar otra vez a Analizar() .

C#

```
namespace EvaluadorExpresiones {
    public class EvaluaSintaxis {
        /* Mensajes de error de sintaxis */
        private readonly string[] _mensajeError = {
            "0. Caracteres no permitidos. Ejemplo: 3$5+2",
            "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)"
        };

        public bool[] EsCorrecto = new bool[27];

        /* Retorna si el caracter es un operador matemático */
        private bool EsUnOperador(char car) {
            return car == '+' || car == '-' || car == '*' || car == '/' || car == '^';
        }

        /* Retorna si el caracter es un número */
        private bool EsUnNumero(char car) {
            return car >= '0' && car <= '9';
        }

        /* Retorna si el caracter es una letra */
        private bool EsUnaLetra(char car) {
            return car >= 'a' && car <= 'z';
        }

        /* 0. Detecta si hay un caracter no válido */
        private bool BuenaSintaxis00(string expresion) {
            bool Resultado = true;
            const string permitidos = "abcdefghijklmnopqrstuvwxyz0123456789.+*/^()";
            for (int pos = 0; pos < expresion.Length && Resultado; pos++)
                if (permitidos.IndexOf(expresion[pos]) == -1)
                    Resultado = false;
            return Resultado;
        }

        /* 1. Un número seguido de una letra. Ejemplo: 2q-(*3) */
        private bool BuenaSintaxis01(string expresion) {
            bool Resultado = true;
            for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
                char carA = expresion[pos];
                char carB = expresion[pos + 1];
                if (EsUnNumero(carA) && EsUnaLetra(carB)) Resultado = false;
            }
            return Resultado;
        }

        /* 2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6) */
        private bool BuenaSintaxis02(string expresion) {
```



```

bool Resultado = true;
for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
    char carA = expresion[pos];
    char carB = expresion[pos + 1];
    if (EsUnNumero(carA) && carB == '(') Resultado = false;
}
return Resultado;
}

/* 3. Doble punto seguido. Ejemplo: 3..1 */
private bool BuenaSintaxis03(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (carA == '.' && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 4. Punto seguido de operador. Ejemplo: 3.*1 */
private bool BuenaSintaxis04(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (carA == '.' && EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 5. Un punto y sigue una letra. Ejemplo: 3+5.w-8 */
private bool BuenaSintaxis05(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (carA == '.' && EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3 */
private bool BuenaSintaxis06(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (carA == '.' && carB == '(') Resultado = false;
    }
    return Resultado;
}

/* 7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3 */
private bool BuenaSintaxis07(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (carA == '.' && carB == ')') Resultado = false;
    }
    return Resultado;
}

/* 8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7 */
private bool BuenaSintaxis08(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (EsUnOperador(carA) && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3 */
private bool BuenaSintaxis09(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {

```

```

    char carA = expresion[pos];
    char carB = expresion[pos + 1];
    if (EsUnOperador(carA) && EsUnOperador(carB)) Resultado = false;
}
return Resultado;
}

/* 10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7 */
private bool BuenaSintaxis10(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (EsUnOperador(carA) && carB == ')') Resultado = false;
    }
    return Resultado;
}

/* 11. Una letra seguida de número. Ejemplo: 7-2a-6 */
private bool BuenaSintaxis11(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (EsUnaLetra(carA) && EsUnNumero(carB)) Resultado = false;
    }
    return Resultado;
}

/* 12. Una letra seguida de punto. Ejemplo: 7-a.-6 */
private bool BuenaSintaxis12(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (EsUnaLetra(carA) && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 13. Un paréntesis que abre seguido de punto. Ejemplo: 7-(.4-6) */
private bool BuenaSintaxis13(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (carA == '(' && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 14. Un paréntesis que abre seguido de un operador. Ejemplo: 2-( *3) */
private bool BuenaSintaxis14(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (carA == '(' && EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 15. Un paréntesis que abre seguido de un paréntesis que cierra. Ejemplo: 7-()-6 */
private bool BuenaSintaxis15(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (carA == '(' && carB == ')') Resultado = false;
    }
    return Resultado;
}

/* 16. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7 */
private bool BuenaSintaxis16(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];

```

```

    if (carA == ')') && EsUnNumero(carB)) Resultado = false;
}
return Resultado;
}

/* 17. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5). */
private bool BuenaSintaxis17(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (carA == ')') && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 18. Un paréntesis que cierra y sigue una letra. Ejemplo: (3-5)t */
private bool BuenaSintaxis18(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (carA == ')') && EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 19. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: (3-5)(4*5) */
private bool BuenaSintaxis19(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (carA == ')') && carB == '(') Resultado = false;
    }
    return Resultado;
}

/* 20. Si hay dos letras seguidas (después de quitar las funciones), es un error */
private bool BuenaSintaxis20(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (EsUnaLetra(carA) && EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 21. Los paréntesis estén desbalanceados. Ejemplo: 3-(2*4)) */
private bool BuenaSintaxis21(string expresion) {
    int parabre = 0; /* Contador de paréntesis que abre */
    int parcierra = 0; /* Contador de paréntesis que cierra */
    for (int pos = 0; pos < expresion.Length; pos++) {
        switch (expresion[pos]) {
            case '(': parabre++; break;
            case ')': parcierra++; break;
        }
    }
    return parcierra == parabre;
}

/* 22. Doble punto en un número de tipo real. Ejemplo: 7-6.46.1+2 */
private bool BuenaSintaxis22(string expresion) {
    bool Resultado = true;
    int totalpuntos = 0; /* Validar los puntos decimales de un número real */
    for (int pos = 0; pos < expresion.Length && Resultado; pos++) {
        char carA = expresion[pos];
        if (EsUnOperador(carA)) totalpuntos = 0;
        if (carA == '.') totalpuntos++;
        if (totalpuntos > 1) Resultado = false;
    }
    return Resultado;
}

/* 23. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4 */
private bool BuenaSintaxis23(string expresion) {
    bool Resultado = true;
    int parabre = 0; /* Contador de paréntesis que abre */

```

```

int parcierra = 0; /* Contador de paréntesis que cierra */
for (int pos = 0; pos < expresion.Length && Resultado; pos++) {
    switch (expresion[pos]) {
        case '(': parabre++; break;
        case ')': parcierra++; break;
    }
    if (parcierra > parabre) Resultado = false;
}
return Resultado;
}

/* 24. Inicia con operador. Ejemplo: +3*5 */
private bool BuenaSintaxis24(string expresion) {
    char carA = expresion[0];
    return !EsUnOperador(carA);
}

/* 25. Finaliza con operador. Ejemplo: 3*5* */
private bool BuenaSintaxis25(string expresion) {
    char carA = expresion[expresion.Length - 1];
    return !EsUnOperador(carA);
}

/* 26. Encuentra una letra seguida de paréntesis que abre. Ejemplo: 3-a(7)-5 */
private bool BuenaSintaxis26(string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.Length - 1 && Resultado; pos++) {
        char carA = expresion[pos];
        char carB = expresion[pos + 1];
        if (EsUnaLetra(carA) && carB == '(') Resultado = false;
    }
    return Resultado;
}

public bool SintaxisCorrecta(string expresionA) {
    /* Reemplaza las funciones de tres letras por una variable que suma */
    string expresionB = expresionA.Replace("sen(", "a+(").Replace("cos(", "a+(").Replace("tan(",
"a+(").Replace("abs(", "a+(").Replace("asn(", "a+(").Replace("acs(", "a+(").Replace("atn(",
"a+(").Replace("log(", "a+(").Replace("cei(", "a+(").Replace("exp(", "a+(").Replace("sqr(",
"a+(").Replace("rcb(", "a+(");

    /* Hace las pruebas de sintaxis */
    EsCorrecto[0] = BuenaSintaxis00(expresionB);
    EsCorrecto[1] = BuenaSintaxis01(expresionB);
    EsCorrecto[2] = BuenaSintaxis02(expresionB);
    EsCorrecto[3] = BuenaSintaxis03(expresionB);
    EsCorrecto[4] = BuenaSintaxis04(expresionB);
    EsCorrecto[5] = BuenaSintaxis05(expresionB);
    EsCorrecto[6] = BuenaSintaxis06(expresionB);
    EsCorrecto[7] = BuenaSintaxis07(expresionB);
    EsCorrecto[8] = BuenaSintaxis08(expresionB);
    EsCorrecto[9] = BuenaSintaxis09(expresionB);
    EsCorrecto[10] = BuenaSintaxis10(expresionB);
    EsCorrecto[11] = BuenaSintaxis11(expresionB);
    EsCorrecto[12] = BuenaSintaxis12(expresionB);
    EsCorrecto[13] = BuenaSintaxis13(expresionB);
    EsCorrecto[14] = BuenaSintaxis14(expresionB);
    EsCorrecto[15] = BuenaSintaxis15(expresionB);
    EsCorrecto[16] = BuenaSintaxis16(expresionB);
    EsCorrecto[17] = BuenaSintaxis17(expresionB);
    EsCorrecto[18] = BuenaSintaxis18(expresionB);
    EsCorrecto[19] = BuenaSintaxis19(expresionB);
    EsCorrecto[20] = BuenaSintaxis20(expresionB);
    EsCorrecto[21] = BuenaSintaxis21(expresionB);
    EsCorrecto[22] = BuenaSintaxis22(expresionB);
    EsCorrecto[23] = BuenaSintaxis23(expresionB);
    EsCorrecto[24] = BuenaSintaxis24(expresionB);
    EsCorrecto[25] = BuenaSintaxis25(expresionB);
    EsCorrecto[26] = BuenaSintaxis26(expresionB);

    bool Resultado = true;
    for (int cont = 0; cont < EsCorrecto.Length && Resultado; cont++)
        if (EsCorrecto[cont] == false) Resultado = false;
    return Resultado;
}

/* Transforma la expresión para ser chequeada y analizada */
public string Transforma(string expresion) {
    /* Quita espacios, tabuladores y la vuelve a minúsculas */

```

```
string nuevo = "";
for (int num = 0; num < expresion.Length; num++) {
    char letra = expresion[num];
    if (letra >= 'A' && letra <= 'Z') letra += ' ';
    if (letra != ' ' && letra != ' ') nuevo += letra.ToString();
}

/* Cambia los )) por )+0) porque es requerido al crear las piezas */
while (nuevo.IndexOf("))") != -1) nuevo = nuevo.Replace("))", ") +0)");

return nuevo;
}

/* Muestra mensaje de error sintáctico */
public string MensajesErrorSintaxis(int codigoError) {
    return _mensajeError[codigoError];
}
}
```

```
namespace EvaluadorExpresiones {
    public class Parte {
        public int Tipo; /* Acumulador, función, paréntesis que abre, paréntesis que cierra, operador, número,
variable */
        public int Funcion; /* 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 char Operador; /* + suma - resta * multiplicación / división ^ potencia */
        public double Numero; /* Número literal, por ejemplo: 3.141592 */
        public int UnaVariable; /* Variable algebraica */
        public int Acumulador; /* Usado cuando la expresión se convierte en piezas. Por ejemplo:
    3 + 2 / 5   se convierte así:
    |3| |+| |2| | / | |5|
    |3| |+| |A|   A es un identificador de acumulador */

        public Parte(int tipo, int funcion, char operador, double numero, int unaVariable) {
            Tipo = tipo;
            Funcion = funcion;
            Operador = operador;
            Numero = numero;
            UnaVariable = unaVariable;
        }
    }
}
```

```
namespace EvaluadorExpresiones {
    public class Pieza {
        public double ValorPieza; /* Almacena el valor que genera la pieza al evaluarse */
        public int Funcion; /* 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 TipoA; /* La primera parte es un número o una variable o trae el valor de otra pieza */
        public double NumeroA; /* Es un número literal */
        public int VariableA; /* Es una variable */
        public int PiezaA; /* Trae el valor de otra pieza */
        public char Operador; /* + suma - resta * multiplicación / división ^ potencia */
        public int TipoB; /* La segunda parte es un número o una variable o trae el valor de otra pieza */
        public double NumeroB; /* Es un número literal */
        public int VariableB; /* Es una variable */
        public int PiezaB; /* Trae el valor de otra pieza */

        public Pieza(int funcion, int tipoA, double numeroA, int variableA, int piezaA, char operador, int
tipoB, double numeroB, int variableB, int piezaB) {
            Funcion = funcion;

            TipoA = tipoA;
            NumeroA = numeroA;
            VariableA = variableA;
            PiezaA = piezaA;

            Operador = operador;

            TipoB = tipoB;
            NumeroB = numeroB;
            VariableB = variableB;
            PiezaB = piezaB;
        }
    }
}
```



```
/* Evaluador de expresiones. Versión 3.0
 * Autor: Rafael Alberto Moreno Parra
 * Fecha: 23 de abril de 2021
 *
 * Pasos para la evaluación de expresiones algebraicas
 * I. Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
 * [3.14] [+] [sen() [4] [/] [x] []] [*] [(] [7.2] [^] [3] [-] [1] [])
 *
 * II. Toma las partes y las divide en piezas con la siguiente estructura:
 * acumula = funcion numero/variable/acumula operador numero/variable/acumula
 * Siguiendo el ejemplo anterior sería:
 * A = 7.2 ^ 3
 * B = A - 1
 * C = seno ( 4 / x )
 * D = C * B
 * E = 3.14 + D
 *
 * Esas piezas se evalúan de arriba a abajo y así se interpreta la expresión
 *
 * */
using System;
using System.Collections.Generic;

namespace EvaluadorExpresiones {
    public class Evaluador3 {
        /* 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;

        /* Listado de partes en que se divide la expresión
         Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
         [3.14] [+] [sen() [4] [/] [x] []] [*] [(] [7.2] [^] [3] [-] [1] [])
         Cada parte puede tener un número, un operador, una función, un paréntesis que abre o un paréntesis
         que cierra */
        private List<Parte> Partes = new List<Parte>();

        /* Listado de piezas que ejecutan
         Toma las partes y las divide en piezas con la siguiente estructura:
         acumula = funcion numero/variable/acumula operador numero/variable/acumula
         Siguiendo el ejemplo anterior sería:
         A = 7.2 ^ 3
         B = A - 1
         C = seno ( 4 / x )
         D = C * B
         E = 3.14 + D

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

        /* El arreglo unidimensional que lleva el valor de las variables */
        private double[] VariableAlgebra = new double[26];

        /* Uso del chequeo de sintaxis */
        public EvaluaSintaxis Sintaxis = new EvaluaSintaxis();

        /* Analiza la expresión */
        public bool Analizar(string expresionA) {
            string expresionB = Sintaxis.Transforma(expresionA);
            bool chequeo = Sintaxis.SintaxisCorrecta(expresionB);
            if (chequeo) {
                Partes.Clear();
                Piezas.Clear();
                CrearPartes(expresionB);
                CrearPiezas();
            }
            return chequeo;
        }

        /* Divide la expresión en partes */
        private void CrearPartes(string expresion) {
            /* Se evalúa entre paréntesis */
        }
    }
}
```



```

string NuevoA = "(" + expresion + ")";

/* Reemplaza las funciones de tres letras por una letra mayúscula */
string NuevoB = NuevoA.Replace("sen", "A").Replace("cos", "B").Replace("tan", "C").Replace("abs",
"D").Replace("asn", "E").Replace("acs", "F").Replace("atn", "G").Replace("log", "H").Replace("cei",
"I").Replace("exp", "J").Replace("sqr", "K").Replace("rcb", "L");

/* Va de caracter en caracter */
string Numero = "";
for (int pos = 0; pos < NuevoB.Length; pos++) {
    char car = NuevoB[pos];
    /* Si es un número lo va acumulando en una cadena */
    if ((car >= '0' && car <= '9') || car == '.') {
        Numero += car.ToString();
    }
    /* Si es un operador entonces agrega número (si existía) */
    else if (car == '+' || car == '-' || car == '*' || car == '/' || car == '^') {
        if (Numero.Length > 0) {
            Partes.Add(new Parte(ESNUMERO, -1, '0', CadenaAReal(Numero), 0));
            Numero = "";
        }
        Partes.Add(new Parte(ESOPERADOR, -1, car, 0, 0));
    }
    /* Si es variable */
    else if (car >= 'a' && car <= 'z') {
        Partes.Add(new Parte(ESVARIABLE, -1, '0', 0, car - 'a'));
    }
    /* Si es una función (seno, coseno, tangente, ...) */
    else if (car >= 'A' && car <= 'L') {
        Partes.Add(new Parte(ESFUNCION, car - 'A', '0', 0, 0));
        pos++;
    }
    /* Si es un paréntesis que abre */
    else if (car == '(') {
        Partes.Add(new Parte(ESPARABRE, -1, '0', 0, 0));
    }
    /* Si es un paréntesis que cierra */
    else {
        if (Numero.Length > 0) {
            Partes.Add(new Parte(ESNUMERO, -1, '0', CadenaAReal(Numero), 0));
            Numero = "";
        }
        /* 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 Parte(ESOPERADOR, -1, '+', 0, 0));
            Partes.Add(new Parte(ESNUMERO, -1, '0', 0, 0));
        }

        Partes.Add(new Parte(ESPARCIERRA, -1, '0', 0, 0));
    }
}

/* Convierte un número almacenado en una cadena a su valor real */
private double CadenaAReal(string Numero) {
    /* Parte entera */
    double parteEntera = 0;
    int cont;
    for (cont = 0; cont < Numero.Length; cont++) {
        if (Numero[cont] == '.') break;
        parteEntera = parteEntera * 10 + (Numero[cont] - '0');
    }

    /* Parte decimal */
    double parteDecimal = 0;
    double multiplica = 1;
    for (int num = cont + 1; num < Numero.Length; num++) {
        parteDecimal = parteDecimal * 10 + (Numero[num] - '0');
        multiplica *= 10;
    }

    double numero = parteEntera + parteDecimal / multiplica;
    return numero;
}

/* Ahora convierte las partes en las piezas finales de ejecución */
private void CrearPiezas() {
    int cont = Partes.Count - 1;

```

```

do {
    Parte tmpParte = Partes[cont];
    if (tmpParte.Tipo == ESPARABRE || tmpParte.Tipo == ESFUNCION) {
        GenerarPiezasOperador('^', '^', cont); /* Evalúa las potencias */
        GenerarPiezasOperador('*', '/', cont); /* Luego evalúa multiplicar y dividir */
        GenerarPiezasOperador('+', '-', cont); /* Finalmente evalúa sumar y restar */

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

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

/* Genera las piezas buscando determinado operador */
private void GenerarPiezasOperador(char operA, char operB, int inicia) {
    int cont = inicia + 1;
    do {
        Parte tmpParte = Partes[cont];
        if (tmpParte.Tipo == ESOPERADOR && (tmpParte.Operador == operA || tmpParte.Operador == operB)) {
            Parte tmpParteIzq = Partes[cont - 1];
            Parte tmpParteDer = Partes[cont + 1];

            /* Crea Pieza */
            Piezas.Add(new Pieza(-1,
                tmpParteIzq.Tipo, tmpParteIzq.Numero,
                tmpParteIzq.UnaVariable, tmpParteIzq.Acumulador,
                tmpParte.Operador,
                tmpParteDer.Tipo, tmpParteDer.Numero,
                tmpParteDer.UnaVariable, tmpParteDer.Acumulador));

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

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

            /* Cambia la parte anterior por parte que acumula */
            tmpParteIzq.Tipo = ESACUMULA;
            tmpParteIzq.Acumulador = Piezas.Count-1;
        }
        cont++;
    } while (Partes[cont].Tipo != ESPARCIERRA);
}

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

    for (int pos = 0; pos < Piezas.Count; pos++) {
        Pieza tmpPieza = Piezas[pos];
        double numA, numB;

        switch (tmpPieza.TipoA) {
            case ESNUMERO: numA = tmpPieza.NumeroA; break;
            case ESVARIABLE: numA = VariableAlgebra[tmpPieza.VariableA]; break;
            default: numA = Piezas[tmpPieza.PiezaA].ValorPieza; break;
        }

        switch (tmpPieza.TipoB) {
            case ESNUMERO: numB = tmpPieza.NumeroB; break;
            case ESVARIABLE: numB = VariableAlgebra[tmpPieza.VariableB]; break;
            default: numB = Piezas[tmpPieza.PiezaB].ValorPieza; break;
        }

        switch (tmpPieza.Operador) {
            case '*': resultado = numA * numB; break;
            case '/': resultado = numA / numB; break;
            case '+': resultado = numA + numB; break;
            case '-': resultado = numA - numB; break;
            default: resultado = Math.Pow(numA, numB); break;
        }
    }
}

```


Como usar el evaluador

En Visual Studio 2019, se añaden las clases al proyecto:

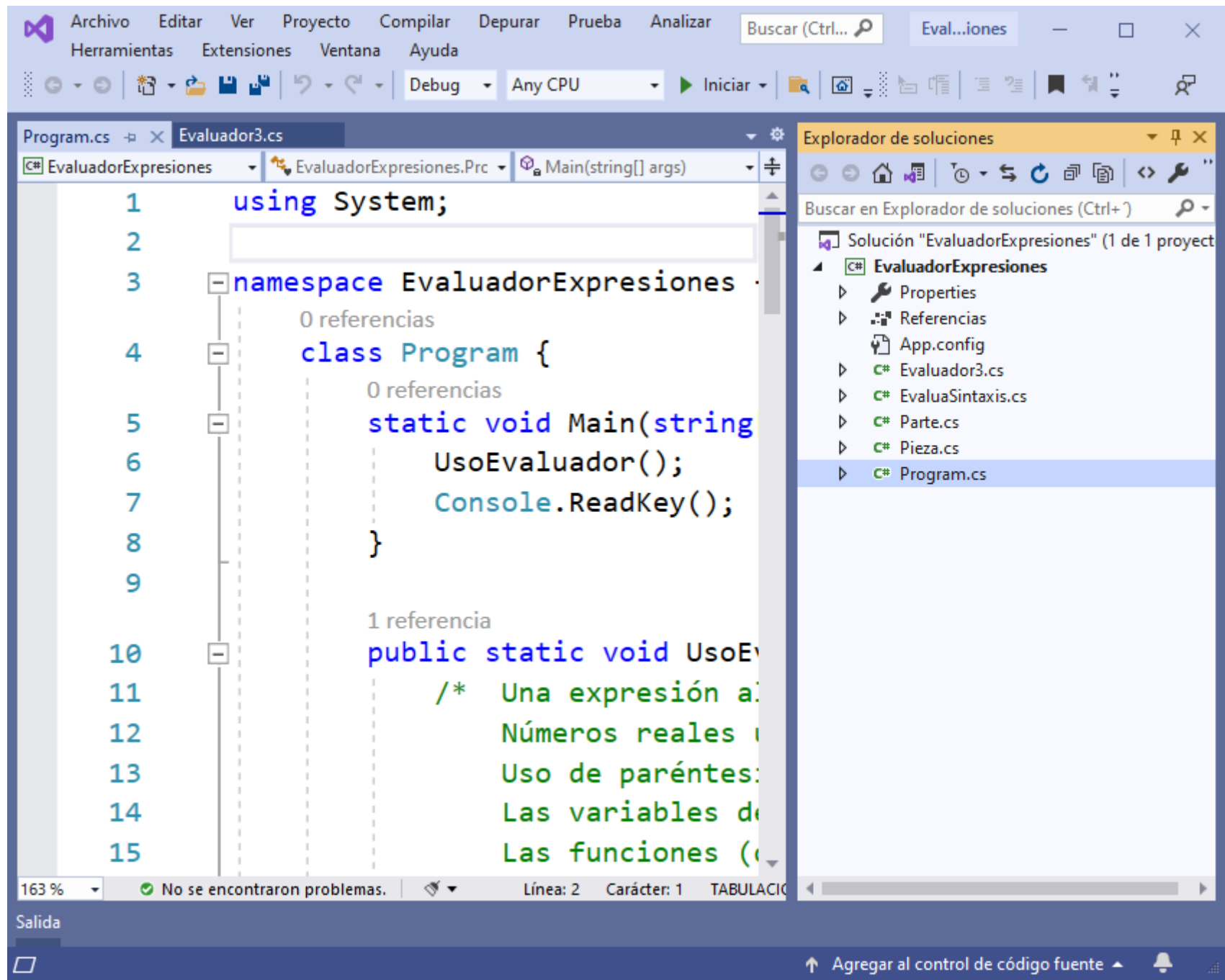


Ilustración 1: En Microsoft Visual Studio 2019

En el programa principal se escribe el código de ejemplo a continuación.

Program.cs

```
using System;

namespace EvaluadorExpresiones {
    class Program {
        static void Main(string[] args) {
            UsoEvaluador();
            Console.ReadKey();
        }

        public static void UsoEvaluador() {
            /* Una expresión algebraica:
            Números reales usan el punto decimal
            Uso de paréntesis
            Las variables deben estar en minúsculas van de la 'a' a la 'z' excepto ñ
            Las funciones (de tres letras) son:
            Sen Seno
            Cos Coseno
            Tan Tangente
            Abs Valor absoluto
            Asn Arcoseno
            Acs Arcocoseno
            Atn Arcotangente
            Log Logaritmo Natural
            Cei Valor techo
            Exp Exponencial
            Sqr Raíz cuadrada
            Rcb Raíz Cúbica
            Los operadores son:
```

```
+ (suma)
- (resta)
* (multiplicación)
/ (división)
^ (potencia)
No se acepta el "-" unario. Luego expresiones como: 4*-2 o (-5+3) o (-x^2) o (-x)^2 son inválidas.
*/
string expresion = "Cos(0.004 * x) - (Tan(1.78 / k + h) * SEN(k ^ x) + abs (k^3-h^2))";

//Instancia el evaluador
Evaluador3 evaluador = new Evaluador3();

//Analiza la expresión (valida sintaxis)
if (evaluador.Analizar(expresion)) {

    //Si no hay fallos de sintaxis, puede evaluar la expresión

    //Da valores a las variables que deben estar en minúsculas
    evaluador.DarValorVariable('k', 1.6);
    evaluador.DarValorVariable('x', -8.3);
    evaluador.DarValorVariable('h', 9.29);

    //Evalúa la expresión
    double resultado = evaluador.Evaluar();
    Console.WriteLine(resultado);

    //Evalúa con ciclos
    Random azar = new Random();
    for (int num = 1; num <= 10; num++) {
        double valor = azar.NextDouble();
        evaluador.DarValorVariable('k', valor);
        resultado = evaluador.Evaluar();
        Console.WriteLine(resultado);
    }
}
else {
    //Si se detectó un error de sintaxis
    for (int unError = 0; unError < evaluador.Sintaxis.EsCorrecto.Length; unError++) {
        //Muestra que error de sintaxis se produjo
        if (evaluador.Sintaxis.EsCorrecto[unError] == false)
            Console.WriteLine(evaluador.Sintaxis.MensajesErrorSintaxis(unError));
    }
}
}
```

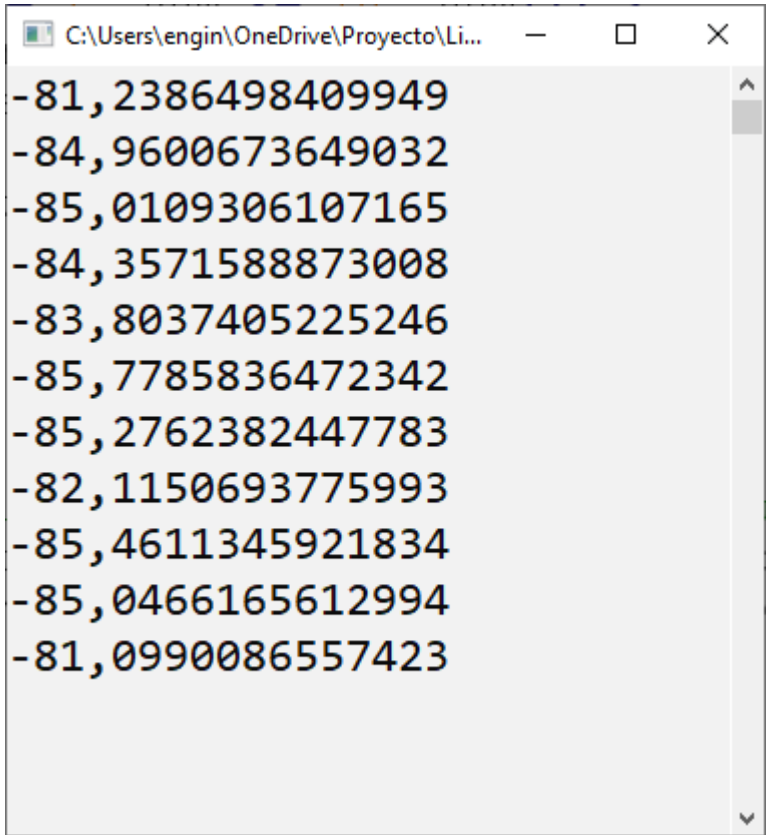


Ilustración 2: Ejecución del programa en C#

C++

```
#include <string>
#include <algorithm>
#include <iostream>

class EvaluaSintaxis{
private:
    /* Mensajes de error de sintaxis */
    std::string _mensajeError[27] = {
        "0. Caracteres no permitidos. Ejemplo: 3$5+2",
        "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)*"
    };

    bool EsUnOperador(char car);
    bool EsUnNumero(char car);
    bool EsUnaLetra(char car);
    bool BuenaSintaxis00(std::string expresion);
    bool BuenaSintaxis01(std::string expresion);
    bool BuenaSintaxis02(std::string expresion);
    bool BuenaSintaxis03(std::string expresion);
    bool BuenaSintaxis04(std::string expresion);
    bool BuenaSintaxis05(std::string expresion);
    bool BuenaSintaxis06(std::string expresion);
    bool BuenaSintaxis07(std::string expresion);
    bool BuenaSintaxis08(std::string expresion);
    bool BuenaSintaxis09(std::string expresion);
    bool BuenaSintaxis10(std::string expresion);
    bool BuenaSintaxis11(std::string expresion);
    bool BuenaSintaxis12(std::string expresion);
    bool BuenaSintaxis13(std::string expresion);
    bool BuenaSintaxis14(std::string expresion);
    bool BuenaSintaxis15(std::string expresion);
    bool BuenaSintaxis16(std::string expresion);
    bool BuenaSintaxis17(std::string expresion);
    bool BuenaSintaxis18(std::string expresion);
    bool BuenaSintaxis19(std::string expresion);
    bool BuenaSintaxis20(std::string expresion);
    bool BuenaSintaxis21(std::string expresion);
    bool BuenaSintaxis22(std::string expresion);
    bool BuenaSintaxis23(std::string expresion);
    bool BuenaSintaxis24(std::string expresion);
    bool BuenaSintaxis25(std::string expresion);
    bool BuenaSintaxis26(std::string expresion);
    std::string ReplaceAll(std::string Cadena, const std::string& Buscar, const std::string& Reemplazar);
public:
    bool EsCorrecto[27];
    int SintaxisCorrecta(std::string expresion);
```



```
std::string Transforma(std::string expresion);
std::string MensajesErrorSintaxis(int CodigoError);
};
```

EvaluaSintaxis.cpp

```
#include "EvaluaSintaxis.h"

/* Retorna si el caracter es un operador matemático */
bool EvaluaSintaxis::EsUnOperador(char car) {
    return car == '+' || car == '-' || car == '*' || car == '/' || car == '^';
}

/* Retorna si el caracter es un número */
bool EvaluaSintaxis::EsUnNumero(char car) {
    return car >= '0' && car <= '9';
}

/* Retorna si el caracter es una letra */
bool EvaluaSintaxis::EsUnaLetra(char car) {
    return car >= 'a' && car <= 'z';
}

/* 0. Detecta si hay un caracter no válido */
bool EvaluaSintaxis::BuenaSintaxis00(std::string expresion){
    bool Resultado = true;
    std::string permitidos = "abcdefghijklmnopqrstuvwxyz0123456789.+*/^()";
    for (int pos = 0; pos < expresion.length() && Resultado; pos++){
        char caracter = expresion.at(pos);
        std::size_t encuentra = permitidos.find(caracter);
        if (encuentra==std::string::npos)
            Resultado = false; //No encontró el caracter
    }
    return Resultado;
}

/* 1. Un número seguido de una letra. Ejemplo: 2q-(*3) */
bool EvaluaSintaxis::BuenaSintaxis01(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (EsUnNumero(carA) && EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6) */
bool EvaluaSintaxis::BuenaSintaxis02(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (EsUnNumero(carA) && carB == '(') Resultado = false;
    }
    return Resultado;
}

/* 3. Doble punto seguido. Ejemplo: 3..1 */
bool EvaluaSintaxis::BuenaSintaxis03(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (carA == '.' && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 4. Punto seguido de operador. Ejemplo: 3.*1 */
bool EvaluaSintaxis::BuenaSintaxis04(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (carA == '.' && EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}
```



```

/* 5. Un punto y sigue una letra. Ejemplo: 3+5.w-8 */
bool EvaluaSintaxis::BuenaSintaxis05(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (carA == '.' && EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3 */
bool EvaluaSintaxis::BuenaSintaxis06(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (carA == '.' && carB == '(') Resultado = false;
    }
    return Resultado;
}

/* 7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3 */
bool EvaluaSintaxis::BuenaSintaxis07(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (carA == '.' && carB == ')') Resultado = false;
    }
    return Resultado;
}

/* 8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7 */
bool EvaluaSintaxis::BuenaSintaxis08(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (EsUnOperador(carA) && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3 */
bool EvaluaSintaxis::BuenaSintaxis09(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (EsUnOperador(carA) && EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7 */
bool EvaluaSintaxis::BuenaSintaxis10(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (EsUnOperador(carA) && carB == ')') Resultado = false;
    }
    return Resultado;
}

/* 11. Una letra seguida de número. Ejemplo: 7-2a-6 */
bool EvaluaSintaxis::BuenaSintaxis11(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (EsUnaLetra(carA) && EsUnNumero(carB)) Resultado = false;
    }
    return Resultado;
}

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

```

```

bool EvaluaSintaxis::BuenaSintaxis12(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (EsUnaLetra(carA) && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 13. Un paréntesis que abre seguido de punto. Ejemplo: 7-(.4-6) */
bool EvaluaSintaxis::BuenaSintaxis13(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (carA == '(' && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 14. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3) */
bool EvaluaSintaxis::BuenaSintaxis14(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (carA == '(' && EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 15. Un paréntesis que abre seguido de un paréntesis que cierra. Ejemplo: 7-()-6 */
bool EvaluaSintaxis::BuenaSintaxis15(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (carA == '(' && carB == ')') Resultado = false;
    }
    return Resultado;
}

/* 16. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7 */
bool EvaluaSintaxis::BuenaSintaxis16(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (carA == ')' && EsUnNumero(carB)) Resultado = false;
    }
    return Resultado;
}

/* 17. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5). */
bool EvaluaSintaxis::BuenaSintaxis17(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (carA == ')' && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 18. Un paréntesis que cierra y sigue una letra. Ejemplo: (3-5)t */
bool EvaluaSintaxis::BuenaSintaxis18(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (carA == ')' && EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 19. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: (3-5)(4*5) */
bool EvaluaSintaxis::BuenaSintaxis19(std::string expresion) {
    bool Resultado = true;

```

```

for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
    char carA = expresion.at(pos);
    char carB = expresion.at(pos+1);
    if (carA == ')' && carB == '(') Resultado = false;
}
return Resultado;
}

/* 20. Si hay dos letras seguidas (después de quitar las funciones), es un error */
bool EvaluaSintaxis::BuenaSintaxis20(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (EsUnaLetra(carA) && EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 21. Los paréntesis estén desbalanceados. Ejemplo: 3-(2*4)) */
bool EvaluaSintaxis::BuenaSintaxis21(std::string expresion) {
    int parabre = 0; /* Contador de paréntesis que abre */
    int parcierra = 0; /* Contador de paréntesis que cierra */
    for (int pos = 0; pos < expresion.length(); pos++) {
        char carA = expresion.at(pos);
        if (carA == '(') parabre++;
        if (carA == ')') parcierra++;
    }
    return parcierra == parabre;
}

/* 22. Doble punto en un número de tipo real. Ejemplo: 7-6.46.1+2 */
bool EvaluaSintaxis::BuenaSintaxis22(std::string expresion) {
    bool Resultado = true;
    int totalpuntos = 0; /* Validar los puntos decimales de un número real */
    for (int pos = 0; pos < expresion.length() && Resultado; pos++) {
        char carA = expresion.at(pos);
        if (EsUnOperador(carA)) totalpuntos = 0;
        if (carA == '.') totalpuntos++;
        if (totalpuntos > 1) Resultado = false;
    }
    return Resultado;
}

/* 23. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4"; */
bool EvaluaSintaxis::BuenaSintaxis23(std::string expresion) {
    bool Resultado = true;
    int parabre = 0; /* Contador de paréntesis que abre */
    int parcierra = 0; /* Contador de paréntesis que cierra */
    for (int pos = 0; pos < expresion.length() && Resultado; pos++) {
        char carA = expresion.at(pos);
        if (carA == '(') parabre++;
        if (carA == ')') parcierra++;
        if (parcierra > parabre) Resultado = false;
    }
    return Resultado;
}

/* 24. Inicia con operador. Ejemplo: +3*5 */
bool EvaluaSintaxis::BuenaSintaxis24(std::string expresion) {
    char carA = expresion[0];
    return !EsUnOperador(carA);
}

/* 25. Finaliza con operador. Ejemplo: 3*5* */
bool EvaluaSintaxis::BuenaSintaxis25(std::string expresion) {
    char carA = expresion[expresion.length() - 1];
    return !EsUnOperador(carA);
}

/* 26. Encuentra una letra seguida de paréntesis que abre. Ejemplo: 3-a(7)-5 */
bool EvaluaSintaxis::BuenaSintaxis26(std::string expresion) {
    bool Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.at(pos);
        char carB = expresion.at(pos+1);
        if (EsUnaLetra(carA) && carB == '(') Resultado = false;
    }
    return Resultado;
}

```

```

}

int EvaluaSintaxis::SintaxisCorrecta(std::string expresion) {
    /* Arreglo de funciones matemáticas */
    std::string funciones[] = { "sen(", "cos(", "tan(", "abs(", "asn(", "acs(", "atn(", "log(", "cei(",
    "exp(", "sqr(", "rcb(" };

    /* Reemplaza las funciones por a+. Ejemplo: 3*cos(2+x) => 3*a+(2+x) */
    for (int fncn = 0; fncn < sizeof(funciones)/sizeof(funciones[0]); fncn++)
        expresion = ReplaceAll(expresion, funciones[fncn], std::string("a+("));

    EsCorrecto[0] = BuenaSintaxis00(expresion);
    EsCorrecto[1] = BuenaSintaxis01(expresion);
    EsCorrecto[2] = BuenaSintaxis02(expresion);
    EsCorrecto[3] = BuenaSintaxis03(expresion);
    EsCorrecto[4] = BuenaSintaxis04(expresion);
    EsCorrecto[5] = BuenaSintaxis05(expresion);
    EsCorrecto[6] = BuenaSintaxis06(expresion);
    EsCorrecto[7] = BuenaSintaxis07(expresion);
    EsCorrecto[8] = BuenaSintaxis08(expresion);
    EsCorrecto[9] = BuenaSintaxis09(expresion);
    EsCorrecto[10] = BuenaSintaxis10(expresion);
    EsCorrecto[11] = BuenaSintaxis11(expresion);
    EsCorrecto[12] = BuenaSintaxis12(expresion);
    EsCorrecto[13] = BuenaSintaxis13(expresion);
    EsCorrecto[14] = BuenaSintaxis14(expresion);
    EsCorrecto[15] = BuenaSintaxis15(expresion);
    EsCorrecto[16] = BuenaSintaxis16(expresion);
    EsCorrecto[17] = BuenaSintaxis17(expresion);
    EsCorrecto[18] = BuenaSintaxis18(expresion);
    EsCorrecto[19] = BuenaSintaxis19(expresion);
    EsCorrecto[20] = BuenaSintaxis20(expresion);
    EsCorrecto[21] = BuenaSintaxis21(expresion);
    EsCorrecto[22] = BuenaSintaxis22(expresion);
    EsCorrecto[23] = BuenaSintaxis23(expresion);
    EsCorrecto[24] = BuenaSintaxis24(expresion);
    EsCorrecto[25] = BuenaSintaxis25(expresion);
    EsCorrecto[26] = BuenaSintaxis26(expresion);
    bool Resultado = true;
    int cont = 0;

    while (cont < sizeof(EsCorrecto)/sizeof(EsCorrecto[0]) && Resultado){
        if (EsCorrecto[cont] == false)
            Resultado = false;
        cont += 1;
    }

    return Resultado;
}

/* Transforma la expresión para ser chequeada y analizada */
std::string EvaluaSintaxis::Transforma(std::string expresion) {
    /* Quita espacios, tabuladores y la vuelve a minúsculas */
    std::string nuevo = "";
    for (int num = 0; num < expresion.length(); num++) {
        char letra = expresion[num];
        if (letra >= 'A' && letra <= 'Z') letra += ' ';
        if (letra != ' ' && letra != '\t') nuevo += letra;
    }

    /* Cambia los )) por )+0) porque es requerido al crear las piezas */
    while (nuevo.find(std::string("))"), 0) != -1) nuevo = ReplaceAll(nuevo, std::string("))"),
    std::string(")+0)"));

    return nuevo;
}

/* Reemplaza todas las ocurrencias de Buscar por Reemplazar */
std::string EvaluaSintaxis::ReplaceAll(std::string Cadena, const std::string& Buscar, const std::string&
Reemplazar) {
    size_t posicion = 0;
    while((posicion = Cadena.find(Buscar, posicion)) != std::string::npos) {
        Cadena.replace(posicion, Buscar.length(), Reemplazar);
        posicion += Reemplazar.length();
    }
    return Cadena;
}

/* Muestra mensaje de error sintáctico */

```

```
std::string EvaluaSintaxis::MensajesErrorSintaxis(int CodigoError) {  
    return _mensajeError[CodigoError];  
}
```

```
class Parte{
public:
    int Tipo; /* Acumulador, función, paréntesis que abre, paréntesis que cierra, operador, número, variable */
    int Funcion; /* Código de la función 1:seno, 2:coseno, 3:tangente, 4: valor absoluto, 5: arcoseno, 6: arcocoseno, 7: arcotangente, 8: logaritmo natural, 9: valor tope, 10: exponencial, 11: raíz cuadrada, 12: raíz cúbica */
    char Operador; /* + suma - resta * multiplicación / división ^ potencia */
    double Numero; /* Número literal, por ejemplo: 3.141592 */
    int UnaVariable; /* Variable algebraica */
    int Acumulador; /* Usado cuando la expresión se convierte en piezas. Por ejemplo:
        3 + 2 / 5 se convierte así:
        |3| |+| |2| | / | |5|
        |3| |+| |A| A es un identificador de acumulador */
    Parte(int tipo, int funcion, char operador, double numero, int unaVariable);
};
```

```
#include "Parte.h"

/* Constructor */
Parte::Parte(int tipo, int funcion, char operador, double numero, int unaVariable){
    Tipo = tipo;
    Funcion = funcion;
    Operador = operador;
    Numero = numero;
    UnaVariable = unaVariable;
}
```

```
class Pieza{
public:
    double ValorPieza; /* Almacena el valor que genera la pieza al evaluarse */
    int Funcion; /* Código de la función 1:seno, 2:coseno, 3:tangente, 4: valor absoluto, 5: arcoseno, 6:
arcocoseno, 7: arcotangente, 8: logaritmo natural, 9: valor tope, 10: exponencial, 11: raíz cuadrada, 12:
raíz cúbica */
    int TipoA; /* La primera parte es un número o una variable o trae el valor de otra pieza */
    double NumeroA; /* Es un número literal */
    int VariableA; /* Es una variable */
    int PiezaA; /* Trae el valor de otra pieza */
    char Operador; /* + suma - resta * multiplicación / división ^ potencia */
    int TipoB; /* La segunda parte es un número o una variable o trae el valor de otra pieza */
    double NumeroB; /* Es un número literal */
    int VariableB; /* Es una variable */
    int PiezaB; /* Trae el valor de otra pieza */

    Pieza(int funcion, int tipoA, double numeroA, int variableA, int piezaA, char operador, int tipoB, double
numeroB, int variableB, int piezaB);
};
```

```
#include "Pieza.h"

/* Constructor */
Pieza::Pieza(int funcion, int tipoA, double numeroA, int variableA, int piezaA, char operador, int tipoB,
double numeroB, int variableB, int piezaB) {
    Funcion = funcion;

    TipoA = tipoA;
    NumeroA = numeroA;
    VariableA = variableA;
    PiezaA = piezaA;

    Operador = operador;

    TipoB = tipoB;
    NumeroB = numeroB;
    VariableB = variableB;
    PiezaB = piezaB;
}
```



```
/* Evaluador de expresiones. Versión 3.0
 * Autor: Rafael Alberto Moreno Parra
 * Fecha: 24 de abril de 2021
 *
 * Pasos para la evaluación de expresiones algebraicas
 * I. Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
 * [3.14] [+] [sen() [4] [/] [x] []) [*] [(] [7.2] [^] [3] [-] [1] [])
 *
 * II. Toma las partes y las divide en piezas con la siguiente estructura:
 * acumula = funcion numero/variable/acumula operador numero/variable/acumula
 * Siguiendo el ejemplo anterior sería:
 * A = 7.2 ^ 3
 * B = A - 1
 * C = seno ( 4 / x )
 * D = C * B
 * E = 3.14 + D
 *
 * Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación
 *
 * */

#include <string>
#include <algorithm>
#include <iostream>
#include <vector>
#include "Parte.h"
#include "Pieza.h"
#include "EvaluaSintaxis.h"

class Evaluador3{
private:
    static const int ESFUNCION = 1;
    static const int ESPARABRE = 2;
    static const int ESPARCIERRA = 3;
    static const int ESOPERADOR = 4;
    static const int ESNUMERO = 5;
    static const int ESVARIABLE = 6;
    static const int ESACUMULA = 7;

    /* Listado de partes en que se divide la expresión
    Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
    [3.14] [+] [sen() [4] [/] [x] []) [*] [(] [7.2] [^] [3] [-] [1] [])
    Cada parte puede tener un número, un operador, una función, un paréntesis que abre o un paréntesis
que cierra */
    std::vector<Parte> Partes;

    /* Listado de piezas que ejecutan
    Toma las partes y las divide en piezas con la siguiente estructura:
    acumula = funcion numero/variable/acumula operador numero/variable/acumula
    Siguiendo el ejemplo anterior sería:
    A = 7.2 ^ 3
    B = A - 1
    C = seno ( 4 / x )
    D = C * B
    E = 3.14 + D

    Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación */
    std::vector<Pieza> Piezas;

    /* El arreglo unidimensional que lleva el valor de las variables */
    double VariableAlgebra[26];

    void CrearPartes(std::string expresion);
    double CadenaAReal(std::string Numero);
    void CrearPiezas();
    void GenerarPiezasOperador(char operA, char operB, int inicia);
    std::string ReplaceAll(std::string Cadena, const std::string& Buscar, const std::string& Reemplazar);
public:
    /* Uso del chequeo de sintaxis */
    EvaluaSintaxis Sintaxis;

    bool Analizar(std::string expresionA);
    double Evaluar();
    void DarValorVariable(char varAlgebra, double valor);
};
```



```

/* Evaluador de expresiones. Versión 3.0
 * Autor: Rafael Alberto Moreno Parra
 * Fecha: 24 de abril de 2021
 *
 * Pasos para la evaluación de expresiones algebraicas
 * I. Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:-
 * [3.14] [+] [sen() [4] [/] [x] []) [*] [(] [7.2] [^] [3] [-] [1] [])
 *
 * II. Toma las partes y las divide en piezas con la siguiente estructura:
 * acumula = funcion numero/variable/acumula operador numero/variable/acumula
 * Siguiendo el ejemplo anterior será-a:
 * A = 7.2 ^ 3
 * B = A - 1
 * C = seno ( 4 / x )
 * D = C * B
 * E = 3.14 + D
 *
 * Esas piezas se evalúan de arriba a abajo y así- se interpreta la ecuación
 *
 * */
#include "Evaluador3.h"
#include <string.h>
#include <math.h>
#include <iostream>
#include <string>

/* Analiza la expresión */
bool Evaluador3::Analizar(std::string expresionA) {
    std::string expresionB = Sintaxis.Transforma(expresionA);
    bool chequeo = Sintaxis.SintaxisCorrecta(expresionB);
    if (chequeo) {
        Partes.clear();
        Piezas.clear();
        CrearPartes(expresionB);
        CrearPiezas();
    }
    return chequeo;
}

/* Divide la expresión en partes */
void Evaluador3::CrearPartes(std::string expresion) {
    /* Reemplaza las funciones de tres letras por una letra mayúscula */
    std::string NuevoA = "(" + expresion + ")";
    std::string NuevoB = ReplaceAll(NuevoA, "sen", "A");
    NuevoB = ReplaceAll(NuevoB, "cos", "B");
    NuevoB = ReplaceAll(NuevoB, "tan", "C");
    NuevoB = ReplaceAll(NuevoB, "abs", "D");
    NuevoB = ReplaceAll(NuevoB, "asn", "E");
    NuevoB = ReplaceAll(NuevoB, "acs", "F");
    NuevoB = ReplaceAll(NuevoB, "atn", "G");
    NuevoB = ReplaceAll(NuevoB, "log", "H");
    NuevoB = ReplaceAll(NuevoB, "cei", "I");
    NuevoB = ReplaceAll(NuevoB, "exp", "J");
    NuevoB = ReplaceAll(NuevoB, "sqr", "K");
    NuevoB = ReplaceAll(NuevoB, "rcb", "L");

    std::string Numero = "";
    for (int pos = 0; pos < NuevoB.length(); pos++) {
        char car = NuevoB[pos];
        /* Si es un número lo va acumulando en una cadena */
        if ((car >= '0' && car <= '9') || car == '.') {
            Numero += car;
        }
        /* Si es un operador entonces agrega número (si existí-a) */
        else if (car == '+' || car == '-' || car == '*' || car == '/' || car == '^') {
            if (Numero.length() > 0) {
                Parte objeto(ESNUMERO, -1, '0', CadenaAReal(Numero), 0);
                Partes.push_back(objeto);
                Numero = "";
            }
            Parte objeto(ESOPERADOR, -1, car, 0, 0);
            Partes.push_back(objeto);
        }
        /* Si es variable */
        else if (car >= 'a' && car <= 'z') {
            Parte objeto(ESVARIABLE, -1, '0', 0, car - 'a');

```

```

    Partes.push_back(objeto);
}
/* Si es una función (seno, coseno, tangente, ...) */
else if (car >= 'A' && car <= 'L') {
    Parte objeto(ESFUNCION, car - 'A', '0', 0, 0);
    Partes.push_back(objeto);
    pos++;
}
/* Si es un paréntesis que abre */
else if (car == '(') {
    Parte objeto(ESPARABRE, -1, '0', 0, 0);
    Partes.push_back(objeto);
}
/* Si es un paréntesis que cierra */
else {
    if (Numero.length() > 0) {
        Parte objeto(ESNUMERO, -1, '0', CadenaAReal(Numero), 0);
        Partes.push_back(objeto);
        Numero = "";
    }
    /* Si sólo había un número o variable dentro del paréntesis le agrega + 0 */
    if (Partes[Partes.size() - 2].Tipo == ESPARABRE || Partes[Partes.size() - 2].Tipo == ESFUNCION) {
        Parte objeto(ESOPERADOR, -1, '+', 0, 0);
        Partes.push_back(objeto);
        Parte objetoB(ESNUMERO, -1, '0', 0, 0);
        Partes.push_back(objetoB);
    }
    Parte objeto(ESPARCIERRA, -1, '0', 0, 0);
    Partes.push_back(objeto);
}
}
}

/* Reemplaza todas las ocurrencias de Buscar por Reemplazar */
std::string Evaluador3::ReplaceAll(std::string Cadena, const std::string& Buscar, const std::string&
Reemplazar) {
    size_t posicion = 0;
    while ((posicion = Cadena.find(Buscar, posicion)) != std::string::npos) {
        Cadena.replace(posicion, Buscar.length(), Reemplazar);
        posicion += Reemplazar.length();
    }
    return Cadena;
}

/* Convierte un número almacenado en una cadena a su valor real */
double Evaluador3::CadenaAReal(std::string Numero) {
    /* Parte entera */
    double parteEntera = 0;
    int cont;
    for (cont = 0; cont < Numero.length(); cont++) {
        if (Numero[cont] == '.') break;
        parteEntera = parteEntera * 10 + (Numero[cont] - '0');
    }

    /* Parte decimal */
    double parteDecimal = 0;
    double multiplica = 1;
    for (int num = cont + 1; num < Numero.length(); num++) {
        parteDecimal = parteDecimal * 10 + (Numero[num] - '0');
        multiplica *= 10;
    }

    double numero = parteEntera + parteDecimal / multiplica;
    return numero;
}

/* Ahora convierte las partes en las piezas finales de ejecución */
void Evaluador3::CrearPiezas() {
    int cont = Partes.size() - 1;
    do {
        Parte tmpParte = Partes[cont];
        if (tmpParte.Tipo == ESPARABRE || tmpParte.Tipo == ESFUNCION) {
            GenerarPiezasOperador('^', '^', cont); /* Evalúa las potencias */
            GenerarPiezasOperador('*', '/', cont); /* Luego evalúa multiplicar y dividir */
            GenerarPiezasOperador('+', '-', cont); /* Finalmente evalúa sumar y restar */

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

```

```

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

/* Genera las piezas buscando determinado operador */
void Evaluador3::GenerarPiezasOperador(char operA, char operB, int inicia) {
    int cont = inicia + 1;
    do {
        if (Partes[cont].Tipo == ESOPERADOR && (Partes[cont].Operador == operA || Partes[cont].Operador ==
operB)) {

            /* Crea Pieza */
            Pieza objeto(-1,
                Partes[cont - 1].Tipo, Partes[cont - 1].Numero,
                Partes[cont - 1].UnaVariable, Partes[cont - 1].Acumulador,
                Partes[cont].Operador,
                Partes[cont + 1].Tipo, Partes[cont + 1].Numero,
                Partes[cont + 1].UnaVariable, Partes[cont + 1].Acumulador);
            Piezas.push_back(objeto);

            /* Elimina la parte del operador y la siguiente */
            Partes.erase(Partes.begin() + cont);
            Partes.erase(Partes.begin() + cont);

            /* Cambia la parte anterior por parte que acumula */
            Partes[cont - 1].Tipo = ESACUMULA;
            Partes[cont - 1].Acumulador = Piezas.size() - 1;

            /* Retorna el contador en uno para tomar la siguiente operación */
            cont--;
        }
        cont++;
    } while (Partes[cont].Tipo != ESPARCIERRA);
}

/* Evalúa la expresión convertida en piezas */
double Evaluador3::Evaluar() {
    double resultado = 0;

    for (int pos = 0; pos < Piezas.size(); pos++) {
        double numA, numB;

        switch (Piezas[pos].TipoA) {
            case ESNUMERO: numA = Piezas[pos].NumeroA; break;
            case ESVARIABLE: numA = VariableAlgebra[Piezas[pos].VariableA]; break;
            default: numA = Piezas[Piezas[pos].PiezaA].ValorPieza; break;
        }

        switch (Piezas[pos].TipoB) {
            case ESNUMERO: numB = Piezas[pos].NumeroB; break;
            case ESVARIABLE: numB = VariableAlgebra[Piezas[pos].VariableB]; break;
            default: numB = Piezas[Piezas[pos].PiezaB].ValorPieza; break;
        }

        switch (Piezas[pos].Operador) {
            case '*': resultado = numA * numB; break;
            case '/': resultado = numA / numB; break;
            case '+': resultado = numA + numB; break;
            case '-': resultado = numA - numB; break;
            default: resultado = pow(numA, numB); break;
        }

        if (!_isnan(resultado) || !_finite(resultado)) return resultado;

        switch (Piezas[pos].Funcion) {
            case 0: resultado = sin(resultado); break;
            case 1: resultado = cos(resultado); break;
            case 2: resultado = tan(resultado); break;
            case 3: resultado = abs(resultado); break;
            case 4: resultado = asin(resultado); break;
            case 5: resultado = acos(resultado); break;
            case 6: resultado = atan(resultado); break;
            case 7: resultado = log(resultado); break;
            case 8: resultado = ceil(resultado); break;
        }
    }
}

```

```

    case 9: resultado = exp(resultado); break;
    case 10: resultado = sqrt(resultado); break;
    case 11: resultado = pow(resultado, 0.33333333333333333333); break;
    }
    if (!_isnan(resultado) || !_finite(resultado)) return resultado;

    Piezas[pos].ValorPieza = resultado;
}
return resultado;
}

/* Da valor a las variables que tendrá la expresión algebraica */
void Evaluador3::DarValorVariable(char varAlgebra, double valor) {
    VariableAlgebra[varAlgebra - 'a'] = valor;
}

```

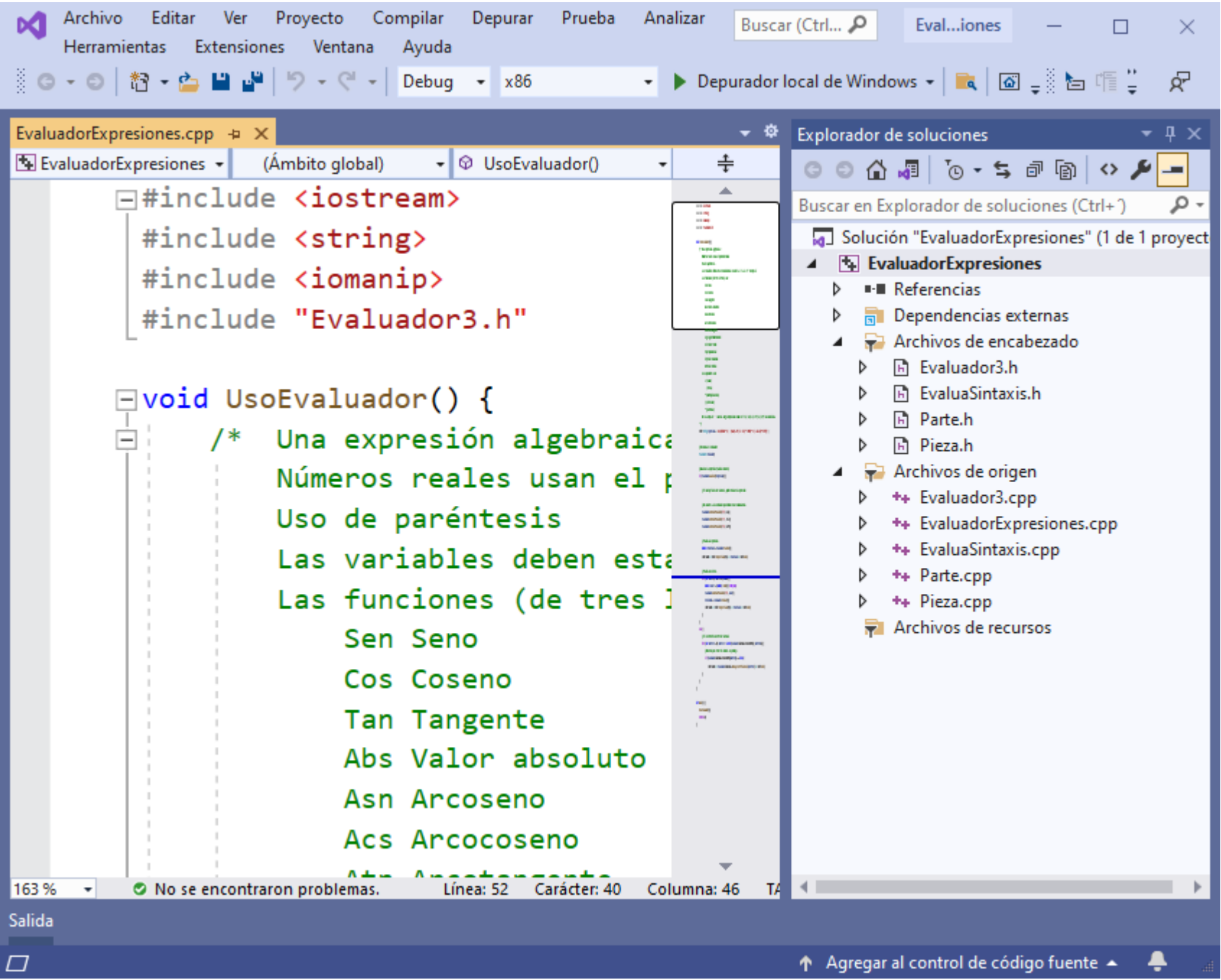


Ilustración 3: En Visual Studio 2019 se cargan las clases y los encabezados (*.cpp y *.h)

```

#include <iostream>
#include <string>
#include <iomanip>
#include "Evaluador3.h"

void UsoEvaluador() {
    /* Una expresión algebraica:
    Números reales usan el punto decimal
    Uso de paréntesis
    Las variables deben estar en minúsculas van de la 'a' a la 'z' excepto ñ
    Las funciones (de tres letras) son:
        Sen  Seno
        Cos  Coseno
        Tan  Tangente
        Abs  Valor absoluto
        Asn  Arcoseno
        Acs  Arcocoseno
        Atn  Arcotangente
        Log  Logaritmo Natural
        Cei  Valor techo
        Exp  Exponencial
        Sqr  Raíz cuadrada
        Rcb  Raíz Cúbica
    Los operadores son:
        + (suma)
        - (resta)
        * (multiplicación)
        / (división)
        ^ (potencia)
    No se acepta el "-" unario. Luego expresiones como: 4*-2 o (-5+3) o (-x^2) o (-x)^2 son inválidas.
    */
    std::string expresion = "Cos(0.004 * x) - (Tan(1.78 / k + h) * SEN(k ^ x) + abs (k^3-h^2))";

    //Instancia el evaluador
    Evaluador3 evaluador;

    //Analiza la expresión (valida sintaxis)
    if (evaluador.Analizar(expresion)) {

        //Si no hay fallos de sintaxis, puede evaluar la expresión

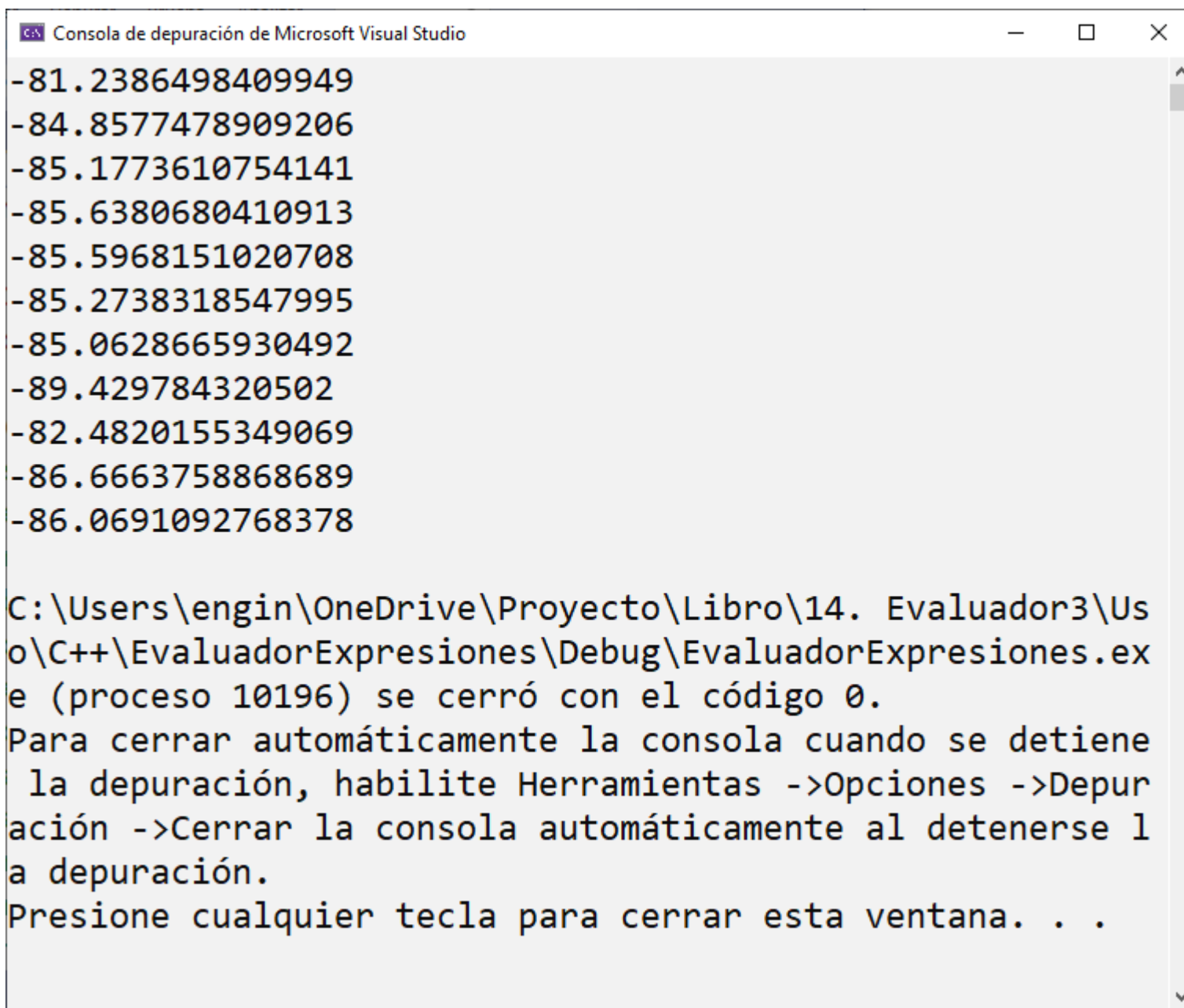
        //Da valores a las variables que deben estar en minúsculas
        evaluador.DarValorVariable('k', 1.6);
        evaluador.DarValorVariable('x', -8.3);
        evaluador.DarValorVariable('h', 9.29);

        //Evalúa la expresión
        double resultado = evaluador.Evaluar();
        std::cout << std::setprecision(15) << resultado << std::endl;

        //Evalúa con ciclos
        for (int num = 1; num <= 10; num++) {
            double valor = (double) rand() / RAND_MAX;
            evaluador.DarValorVariable('k', valor);
            resultado = evaluador.Evaluar();
            std::cout << std::setprecision(15) << resultado << std::endl;
        }
    }
    else {
        //Si se detectó un error de sintaxis
        for (int unError = 0; unError < sizeof(evaluador.Sintaxis.EsCorrecto); unError++) {
            //Muestra que error de sintaxis se produjo
            if (evaluador.Sintaxis.EsCorrecto[unError] == false)
                std::cout << evaluador.Sintaxis.MensajesErrorSintaxis(unError) << std::endl;
        }
    }
}

int main() {
    UsoEvaluador();
    return 0;
}

```



```
Consola de depuración de Microsoft Visual Studio

-81.2386498409949
-84.8577478909206
-85.1773610754141
-85.6380680410913
-85.5968151020708
-85.2738318547995
-85.0628665930492
-89.429784320502
-82.4820155349069
-86.6663758868689
-86.0691092768378

C:\Users\engin\OneDrive\Proyecto\Libro\14. Evaluador3\Us
o\C++\EvaluadorExpresiones\Debug\EvaluadorExpresiones.ex
e (proceso 10196) se cerró con el código 0.
Para cerrar automáticamente la consola cuando se detiene
la depuración, habilite Herramientas ->Opciones ->Depur
ación ->Cerrar la consola automáticamente al detenerse l
a depuración.
Presione cualquier tecla para cerrar esta ventana. . .
```

Ilustración 4: Ejecución del programa

Delphi


```
unit EvaluaSintaxis;

{$MODE Delphi}

interface
uses
  SysUtils;
type
  TEvaluaSintaxis = class
  private
    //Mensajes de error de sintaxis
  const
    _mensajeError : array[0..26] of string = (
      '0. Caracteres no permitidos. Ejemplo: 3$5+2',
      '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)'
    );

    function EsUnOperador(car: char): boolean;
    function EsUnNumero(car: char): boolean;
    function EsUnaLetra(car: char): boolean;
    function BuenaSintaxis00(expresion: string): boolean;
    function BuenaSintaxis01(expresion: string): boolean;
    function BuenaSintaxis02(expresion: string): boolean;
    function BuenaSintaxis03(expresion: string): boolean;
    function BuenaSintaxis04(expresion: string): boolean;
    function BuenaSintaxis05(expresion: string): boolean;
    function BuenaSintaxis06(expresion: string): boolean;
    function BuenaSintaxis07(expresion: string): boolean;
    function BuenaSintaxis08(expresion: string): boolean;
    function BuenaSintaxis09(expresion: string): boolean;
    function BuenaSintaxis10(expresion: string): boolean;
    function BuenaSintaxis11(expresion: string): boolean;
    function BuenaSintaxis12(expresion: string): boolean;
    function BuenaSintaxis13(expresion: string): boolean;
    function BuenaSintaxis14(expresion: string): boolean;
    function BuenaSintaxis15(expresion: string): boolean;
    function BuenaSintaxis16(expresion: string): boolean;
    function BuenaSintaxis17(expresion: string): boolean;
    function BuenaSintaxis18(expresion: string): boolean;
    function BuenaSintaxis19(expresion: string): boolean;
    function BuenaSintaxis20(expresion: string): boolean;
    function BuenaSintaxis21(expresion: string): boolean;
    function BuenaSintaxis22(expresion: string): boolean;
    function BuenaSintaxis23(expresion: string): boolean;
    function BuenaSintaxis24(expresion: string): boolean;
    function BuenaSintaxis25(expresion: string): boolean;
```

```

    function BuenaSintaxis26(expresion: string): boolean;
public
    EsCorrecto : array[0..26] of boolean;
    function SintaxisCorrecta(ecuacion: string): boolean;
    function Transforma(expresion:string): string;
    function MensajesErrorSintaxis(codigoError: integer): string;
end;
implementation

//Retorna si el caracter es un operador matemático */
function TEvaluaSintaxis.EsUnOperador(car: char): boolean;
begin
    Result := (car = '+') or (car = '-') or (car = '*') or (car = '/') or (car = '^');
end;

// Retorna si el caracter es un número
function TEvaluaSintaxis.EsUnNumero(car: char): boolean;
begin
    Result := (car >= '0') and (car <= '9');
end;

// Retorna si el caracter es una letra
function TEvaluaSintaxis.EsUnaLetra(car: char): boolean;
begin
    Result := (car >= 'a') and (car <= 'z');
end;

// 0. Detecta si hay un caracter no válido
function TEvaluaSintaxis.BuenaSintaxis00(expresion: string): boolean;
var
    Resultado: boolean;
    permitidos: string;
    pos: integer;
begin
    Resultado := true;
    permitidos := 'abcdefghijklmnopqrstuvwxyz0123456789.+*/^()';
    for pos := 1 to length(expresion) do
    begin
        if (permitidos.IndexOf(expresion[pos]) = -1) then
        begin
            Resultado := false;
            break;
        end;
    end;
    Result := Resultado;
end;

// 1. Un número seguido de una letra. Ejemplo: 2q-(*3)
function TEvaluaSintaxis.BuenaSintaxis01(expresion: string): boolean;
var
    Resultado: boolean;
    pos: integer;
    carA: char;
    carB: char;
begin
    Resultado := true;
    for pos := 1 to length(expresion)-1 do
    begin
        carA := expresion[pos];
        carB := expresion[pos+1];
        if (EsUnNumero(carA)) and (EsUnaLetra(carB)) then
        begin
            Resultado := false;
            break;
        end;
    end;
    Result := Resultado;
end;

// 2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
function TEvaluaSintaxis.BuenaSintaxis02(expresion: string): boolean;
var
    Resultado: boolean;
    pos: integer;
    carA: char;
    carB: char;
begin
    Resultado := true;
    for pos := 1 to length(expresion)-1 do

```

```

begin
  carA := expresion[pos];
  carB := expresion[pos+1];
  if (EsUnNumero(carA)) and (carB = '(') then
    begin
      Resultado := false;
      break;
    end;
  end;
  Result := Resultado;
end;

// 3. Doble punto seguido. Ejemplo: 3..1
function TEvaluaSintaxis.BuenaSintaxis03(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  carA: char;
  carB: char;
begin
  Resultado := true;
  for pos := 1 to length(expresion)-1 do
    begin
      carA := expresion[pos];
      carB := expresion[pos+1];
      if (carA = '.') and (carB = '.') then
        begin
          Resultado := false;
          break;
        end;
      end;
    end;
  Result := Resultado;
end;

// 4. Punto seguido de operador. Ejemplo: 3.*1
function TEvaluaSintaxis.BuenaSintaxis04(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  carA: char;
  carB: char;
begin
  Resultado := true;
  for pos := 1 to length(expresion)-1 do
    begin
      carA := expresion[pos];
      carB := expresion[pos+1];
      if (carA = '.') and (EsUnOperador(carB)) then
        begin
          Resultado := false;
          break;
        end;
      end;
    end;
  Result := Resultado;
end;

// 5. Un punto y sigue una letra. Ejemplo: 3+5.w-8
function TEvaluaSintaxis.BuenaSintaxis05(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  carA: char;
  carB: char;
begin
  Resultado := true;
  for pos := 1 to length(expresion)-1 do
    begin
      carA := expresion[pos];
      carB := expresion[pos+1];
      if (carA = '.') and (EsUnaLetra(carB)) then
        begin
          Resultado := false;
          break;
        end;
      end;
    end;
  Result := Resultado;
end;

// 6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3

```

```

function TEvaluaSintaxis.BuenaSintaxis06(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  carA: char;
  carB: char;
begin
  Resultado := true;
  for pos := 1 to length(expresion)-1 do
  begin
    carA := expresion[pos];
    carB := expresion[pos+1];
    if (carA = '.') and (carB = '(') then
    begin
      Resultado := false;
      break;
    end;
  end;
  Result := Resultado;
end;

// 7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3
function TEvaluaSintaxis.BuenaSintaxis07(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  carA: char;
  carB: char;
begin
  Resultado := true;
  for pos := 1 to length(expresion)-1 do
  begin
    carA := expresion[pos];
    carB := expresion[pos+1];
    if (carA = '.') and (carB = ')') then
    begin
      Resultado := false;
      break;
    end;
  end;
  Result := Resultado;
end;

// 8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7
function TEvaluaSintaxis.BuenaSintaxis08(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  carA: char;
  carB: char;
begin
  Resultado := true;
  for pos := 1 to length(expresion)-1 do
  begin
    carA := expresion[pos];
    carB := expresion[pos+1];
    if (EsUnOperador(carA)) and (carB = '.') then
    begin
      Resultado := false;
      break;
    end;
  end;
  Result := Resultado;
end;

// 9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3
function TEvaluaSintaxis.BuenaSintaxis09(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  carA: char;
  carB: char;
begin
  Resultado := true;
  for pos := 1 to length(expresion)-1 do
  begin
    carA := expresion[pos];
    carB := expresion[pos+1];
    if (EsUnOperador(carA)) and (EsUnOperador(carB)) then

```

```

    begin
        Resultado := false;
        break;
    end;
end;
Result := Resultado;
end;

// 10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
function TEvaluaSintaxis.BuenaSintaxis10(expresion: string): boolean;
var
    Resultado: boolean;
    pos: integer;
    carA: char;
    carB: char;
begin
    Resultado := true;
    for pos := 1 to length(expresion)-1 do
        begin
            carA := expresion[pos];
            carB := expresion[pos+1];
            if (EsUnOperador(carA)) and (carB = ')') then
                begin
                    Resultado := false;
                    break;
                end;
            end;
        end;
    Result := Resultado;
end;

// 11. Una letra seguida de número. Ejemplo: 7-2a-6
function TEvaluaSintaxis.BuenaSintaxis11(expresion: string): boolean;
var
    Resultado: boolean;
    pos: integer;
    carA: char;
    carB: char;
begin
    Resultado := true;
    for pos := 1 to length(expresion)-1 do
        begin
            carA := expresion[pos];
            carB := expresion[pos+1];
            if (EsUnaLetra(carA)) and (EsUnNumero(carB)) then
                begin
                    Resultado := false;
                    break;
                end;
            end;
        end;
    Result := Resultado;
end;

// 12. Una letra seguida de punto. Ejemplo: 7-a.-6
function TEvaluaSintaxis.BuenaSintaxis12(expresion: string): boolean;
var
    Resultado: boolean;
    pos: integer;
    carA: char;
    carB: char;
begin
    Resultado := true;
    for pos := 1 to length(expresion)-1 do
        begin
            carA := expresion[pos];
            carB := expresion[pos+1];
            if (EsUnaLetra(carA)) and (carB = '.') then
                begin
                    Resultado := false;
                    break;
                end;
            end;
        end;
    Result := Resultado;
end;

// 13. Un paréntesis que abre seguido de punto. Ejemplo: 7-(.4-6)
function TEvaluaSintaxis.BuenaSintaxis13(expresion: string): boolean;
var
    Resultado: boolean;
    pos: integer;

```

```

carA: char;
carB: char;
begin
  Resultado := true;
  for pos := 1 to length(expresion)-1 do
    begin
      carA := expresion[pos];
      carB := expresion[pos+1];
      if (carA = '(') and (carB = '.') then
        begin
          Resultado := false;
          break;
        end;
      end;
    end;
  Result := Resultado;
end;

// 14. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
function TEvaluaSintaxis.BuenaSintaxis14(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  carA: char;
  carB: char;
begin
  Resultado := true;
  for pos := 1 to length(expresion)-1 do
    begin
      carA := expresion[pos];
      carB := expresion[pos+1];
      if (carA = '(') and (EsUnOperador(carB)) then
        begin
          Resultado := false;
          break;
        end;
      end;
    end;
  Result := Resultado;
end;

// 15. Un paréntesis que abre seguido de un paréntesis que cierra. Ejemplo: 7-()-6
function TEvaluaSintaxis.BuenaSintaxis15(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  carA: char;
  carB: char;
begin
  Resultado := true;
  for pos := 1 to length(expresion)-1 do
    begin
      carA := expresion[pos];
      carB := expresion[pos+1];
      if (carA = '(') and (carB = ')') then
        begin
          Resultado := false;
          break;
        end;
      end;
    end;
  Result := Resultado;
end;

// 16. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7
function TEvaluaSintaxis.BuenaSintaxis16(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  carA: char;
  carB: char;
begin
  Resultado := true;
  for pos := 1 to length(expresion)-1 do
    begin
      carA := expresion[pos];
      carB := expresion[pos+1];
      if (carA = ')') and (EsUnNumero(carB)) then
        begin
          Resultado := false;
          break;
        end;
      end;
    end;
  end;
end;

```

```

    Result := Resultado;
end;

// 17. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5).
function TEvaluaSintaxis.BuenaSintaxis17(expresion: string): boolean;
var
    Resultado: boolean;
    pos: integer;
    carA: char;
    carB: char;
begin
    Resultado := true;
    for pos := 1 to length(expresion)-1 do
    begin
        carA := expresion[pos];
        carB := expresion[pos+1];
        if (carA = ')') and (carB = '.') then
        begin
            Resultado := false;
            break;
        end;
    end;
    Result := Resultado;
end;

// 18. Un paréntesis que cierra y sigue una letra. Ejemplo: (3-5)t
function TEvaluaSintaxis.BuenaSintaxis18(expresion: string): boolean;
var
    Resultado: boolean;
    pos: integer;
    carA: char;
    carB: char;
begin
    Resultado := true;
    for pos := 1 to length(expresion)-1 do
    begin
        carA := expresion[pos];
        carB := expresion[pos+1];
        if (carA = ')') and (EsUnaLetra(carB)) then
        begin
            Resultado := false;
            break;
        end;
    end;
    Result := Resultado;
end;

// 19. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: (3-5)(4*5)
function TEvaluaSintaxis.BuenaSintaxis19(expresion: string): boolean;
var
    Resultado: boolean;
    pos: integer;
    carA: char;
    carB: char;
begin
    Resultado := true;
    for pos := 1 to length(expresion)-1 do
    begin
        carA := expresion[pos];
        carB := expresion[pos+1];
        if (carA = ')') and (carB = '(') then
        begin
            Resultado := false;
            break;
        end;
    end;
    Result := Resultado;
end;

// 20. Si hay dos letras seguidas (después de quitar las funciones), es un error
function TEvaluaSintaxis.BuenaSintaxis20(expresion: string): boolean;
var
    Resultado: boolean;
    pos: integer;
    carA: char;
    carB: char;
begin
    Resultado := true;
    for pos := 1 to length(expresion)-1 do

```



```

begin
  carA := expresion[pos];
  carB := expresion[pos+1];
  if (EsUnaLetra(carA)) and (EsUnaLetra(carB)) then
    begin
      Resultado := false;
      break;
    end;
  end;
  Result := Resultado;
end;

// 21. Los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
function TEvaluaSintaxis.BuenaSintaxis21(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  parabre: integer;
  parcierra: integer;
begin
  Resultado := true;
  parabre := 0; // Contador de paréntesis que abre
  parcierra := 0; // Contador de paréntesis que cierra
  for pos := 1 to length(expresion) do
    begin
      if (expresion[pos] = '(') then begin Inc(parabre); end;
      if (expresion[pos] = ')') then begin Inc(parcierra); end;
    end;
  if (parabre <> parcierra) then begin Resultado := false; end;
  Result := Resultado;
end;

// 22. Doble punto en un número de tipo real. Ejemplo: 7-6.46.1+2
function TEvaluaSintaxis.BuenaSintaxis22(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  carA: char;
  totalpuntos: integer;
begin
  Resultado := true;
  totalpuntos := 0; // Validar los puntos decimales de un número real
  for pos := 1 to length(expresion) do
    begin
      carA := expresion[pos];
      if (EsUnOperador(carA)) then begin totalpuntos := 0; end;
      if (carA = '.') then begin Inc(totalpuntos); end;
      if (totalpuntos > 1) then
        begin
          Resultado := false;
          break;
        end;
    end;
  end;
  Result := Resultado;
end;

// 23. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4
function TEvaluaSintaxis.BuenaSintaxis23(expresion: string): boolean;
var
  Resultado: boolean;
  pos: integer;
  parabre: integer;
  parcierra: integer;
begin
  Resultado := true;
  parabre := 0; // Contador de paréntesis que abre
  parcierra := 0; // Contador de paréntesis que cierra
  for pos := 1 to length(expresion) do
    begin
      if (expresion[pos] = '(') then begin Inc(parabre); end;
      if (expresion[pos] = ')') then begin Inc(parcierra); end;
      if (parcierra > parabre) then
        begin
          Resultado := false;
          break;
        end;
    end;
  end;
  Result := Resultado;
end;

```



```

// 24. Inicia con operador. Ejemplo: +3*5
function TEvaluaSintaxis.BuenaSintaxis24(expresion: string): boolean;
var
    Resultado: boolean;
begin
    Resultado := not EsUnOperador(expresion[1]);
    Result := Resultado;
end;

// 25. Finaliza con operador. Ejemplo: 3*5*
function TEvaluaSintaxis.BuenaSintaxis25(expresion: string): boolean;
var
    Resultado: boolean;
begin
    Resultado := not EsUnOperador(expresion[length(expresion)]);
    Result := Resultado;
end;

// 26. Encuentra una letra seguida de paréntesis que abre. Ejemplo: 3-a(7)-5
function TEvaluaSintaxis.BuenaSintaxis26(expresion: string): boolean;
var
    Resultado: boolean;
    pos: integer;
    carA: char;
    carB: char;
begin
    Resultado := true;
    for pos := 1 to length(expresion)-1 do
        begin
            carA := expresion[pos];
            carB := expresion[pos+1];
            if (EsUnaLetra(carA)) and (carB = '(') then
                begin
                    Resultado := false;
                    break;
                end;
        end;
    end;
    Result := Resultado;
end;

function TEvaluaSintaxis.SintaxisCorrecta(ecuacion: string): boolean;
var
    expresion: string;
    Resultado: boolean;
    cont: integer;
begin
    // Reemplaza las funciones de tres letras por una variable que suma
    expresion := StringReplace(ecuacion, 'sen(', 'a+', [rfReplaceAll, rfIgnoreCase]);
    expresion := StringReplace(expresion, 'cos(', 'a+', [rfReplaceAll, rfIgnoreCase]);
    expresion := StringReplace(expresion, 'tan(', 'a+', [rfReplaceAll, rfIgnoreCase]);
    expresion := StringReplace(expresion, 'abs(', 'a+', [rfReplaceAll, rfIgnoreCase]);
    expresion := StringReplace(expresion, 'asn(', 'a+', [rfReplaceAll, rfIgnoreCase]);
    expresion := StringReplace(expresion, 'acs(', 'a+', [rfReplaceAll, rfIgnoreCase]);
    expresion := StringReplace(expresion, 'atn(', 'a+', [rfReplaceAll, rfIgnoreCase]);
    expresion := StringReplace(expresion, 'log(', 'a+', [rfReplaceAll, rfIgnoreCase]);
    expresion := StringReplace(expresion, 'cei(', 'a+', [rfReplaceAll, rfIgnoreCase]);
    expresion := StringReplace(expresion, 'exp(', 'a+', [rfReplaceAll, rfIgnoreCase]);
    expresion := StringReplace(expresion, 'sqr(', 'a+', [rfReplaceAll, rfIgnoreCase]);
    expresion := StringReplace(expresion, 'rcb(', 'a+', [rfReplaceAll, rfIgnoreCase]);

    // Hace las pruebas de sintaxis
    EsCorrecto[0] := BuenaSintaxis00(expresion);
    EsCorrecto[1] := BuenaSintaxis01(expresion);
    EsCorrecto[2] := BuenaSintaxis02(expresion);
    EsCorrecto[3] := BuenaSintaxis03(expresion);
    EsCorrecto[4] := BuenaSintaxis04(expresion);
    EsCorrecto[5] := BuenaSintaxis05(expresion);
    EsCorrecto[6] := BuenaSintaxis06(expresion);
    EsCorrecto[7] := BuenaSintaxis07(expresion);
    EsCorrecto[8] := BuenaSintaxis08(expresion);
    EsCorrecto[9] := BuenaSintaxis09(expresion);
    EsCorrecto[10] := BuenaSintaxis10(expresion);
    EsCorrecto[11] := BuenaSintaxis11(expresion);
    EsCorrecto[12] := BuenaSintaxis12(expresion);
    EsCorrecto[13] := BuenaSintaxis13(expresion);
    EsCorrecto[14] := BuenaSintaxis14(expresion);
    EsCorrecto[15] := BuenaSintaxis15(expresion);

```

```

EsCorrecto[16] := BuenaSintaxis16(expresion);
EsCorrecto[17] := BuenaSintaxis17(expresion);
EsCorrecto[18] := BuenaSintaxis18(expresion);
EsCorrecto[19] := BuenaSintaxis19(expresion);
EsCorrecto[20] := BuenaSintaxis20(expresion);
EsCorrecto[21] := BuenaSintaxis21(expresion);
EsCorrecto[22] := BuenaSintaxis22(expresion);
EsCorrecto[23] := BuenaSintaxis23(expresion);
EsCorrecto[24] := BuenaSintaxis24(expresion);
EsCorrecto[25] := BuenaSintaxis25(expresion);
EsCorrecto[26] := BuenaSintaxis26(expresion);

Resultado := true;
for cont := 0 to 26 do
begin
  if (EsCorrecto[cont] = false) then begin Resultado := false; end;
end;
Result := Resultado;
end;

function TEvaluaSintaxis.Transforma(expresion:string): string;
var
  nuevo: string;
  num: integer;
  letra: char;
begin
  //Quita espacios, tabuladores y la vuelve a minúsculas */
  nuevo := '';
  for num := 1 to expresion.Length do
  begin
    letra := expresion[num];
    if (letra >= 'A') and (letra <= 'Z') then begin letra := chr(ord(letra) + ord(' ')); end;
    if (letra <> ' ') and (letra <> '') then begin nuevo := nuevo + letra; end;
  end;

  //Cambia los )) por )+0) porque es requerido al crear las piezas
  while (nuevo.IndexOf('))') <> -1) do
  begin
    nuevo := StringReplace(nuevo, '))', ')+0)', [rfReplaceAll, rfIgnoreCase]);
  end;

  Result := nuevo;
end;

// Muestra mensaje de error sintáctico
function TEvaluaSintaxis.MensajesErrorSintaxis(codigoError: integer): string;
begin
  Result := _mensajeError[codigoError];
end;

end.

```

```
unit Partes;

{$MODE Delphi}

interface
type
  TParte = class
  public
    Tipo: integer; // Acumulador, función, paréntesis que abre, paréntesis que cierra, operador, número,
variable
    Funcion: integer; // Código de la función 1:seno, 2:coseno, 3:tangente, 4: valor absoluto, 5:
arcoseno, 6: arcocoseno, 7: arcotangente, 8: logaritmo natural, 9: valor tope, 10: exponencial, 11: raíz
cuadrada, 12: raíz cúbica
    Operador: char; // + suma - resta * multiplicación / división ^ potencia
    Numero: double; // Número literal, por ejemplo: 3.141592
    UnaVariable: integer; // Variable algebraica */
    Acumulador: integer; { Usado cuando la expresión se convierte en piezas. Por ejemplo:
      3 + 2 / 5 se convierte así:
      |3| |+| |2| | / | |5|
      |3| |+| |A| A es un identificador de acumulador }
  Constructor Create(Tipo: integer; Funcion: integer; Operador: char; Numero: double; UnaVariable:
integer);
  end;

implementation

Constructor TParte.Create(Tipo: integer; Funcion: integer; Operador: char; Numero: double; UnaVariable:
integer);
begin
  self.Tipo := Tipo;
  self.Funcion := Funcion;
  self.Operador := Operador;
  self.Numero := Numero;
  self.UnaVariable := UnaVariable;
  self.Acumulador := 0;
end;
end.
```

```
unit Piezas;

{$MODE Delphi}

interface
type
    TPieza = class
    public
        ValorPieza: double; // Almacena el valor que genera la pieza al evaluarse
        Funcion: integer; // Código de la función 1:seno, 2:coseno, 3:tangente, 4: valor absoluto, 5:
arcoseno, 6: arcocoseno, 7: arcotangente, 8: logaritmo natural, 9: valor tope, 10: exponencial, 11: raíz
cuadrada, 12: raíz cúbica
        TipoA: integer; // La primera parte es un número o una variable o trae el valor de otra pieza
        NumeroA: double; // Es un número literal
        VariableA: integer; // Es una variable
        PiezaA: integer; // Trae el valor de otra pieza
        Operador: char; // + suma - resta * multiplicación / división ^ potencia
        TipoB: integer; // La segunda parte es un número o una variable o trae el valor de otra pieza
        NumeroB: double; // Es un número literal
        VariableB: integer; // Es una variable
        PiezaB: integer; // Trae el valor de otra pieza
        Constructor Create(Funcion: integer; TipoA: integer; NumeroA: double; VariableA: integer; PiezaA:
integer; Operador: char; TipoB: integer; NumeroB: double; VariableB: integer; PiezaB: integer);
    end;

implementation

Constructor TPieza.Create(Funcion: integer; TipoA: integer; NumeroA: double; VariableA: integer; PiezaA:
integer; Operador: char; TipoB: integer; NumeroB: double; VariableB: integer; PiezaB: integer);
begin
    self.Funcion := Funcion;

    self.TipoA := TipoA;
    self.NumeroA := NumeroA;
    self.VariableA := VariableA;
    self.PiezaA := PiezaA;

    self.Operador := Operador;

    self.TipoB := TipoB;
    self.NumeroB := NumeroB;
    self.VariableB := VariableB;
    self.PiezaB := PiezaB;
end;

end.
```

```
unit Evaluador3;

{$MODE Delphi}

interface
uses
    //Requerido para el TObjectList
    Contnrs, SysUtils, Math, Partes, Piezas, EvaluaSintaxis;
type
    TEvaluador3 = class
    private
        { Autor: Rafael Alberto Moreno Parra. 10 de abril de 2021 }

        { Constantes de los diferentes tipos de datos que tendrán las piezas }
        ESFUNCION: integer;
        ESPARABRE: integer;
        ESPARCIERRA: integer;
        ESOPERADOR: integer;
        ESNUMERO: integer;
        ESVARIABLE: integer;
        ESACUMULA: integer;

        { Listado de partes en que se divide la expresión
          Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
          |3.14| |+| |sen(| |4| | / | |x| |)| |*| |( | |7.2| | ^ | |3| |-| |1| |)|
          Cada parte puede tener un número, un operador, una función, un paréntesis que abre o un paréntesis
          que cierra }
        Partes: TObjectList;

        { Listado de piezas que ejecutan
          Toma las partes y las divide en piezas con la siguiente estructura:
          acumula = funcion  numero/variable/acumula  operador  numero/variable/acumula
          Siguiendo el ejemplo anterior sería:
          A = 7.2 ^ 3
          B = A - 1
          C = seno ( 4 / x )
          D = C * B
          E = 3.14 + D

          Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación }
        Piezas: TObjectList;

        { El arreglo unidimensional que lleva el valor de las variables }
        VariableAlgebra: array[0..26] of double;

        procedure CrearPartes(expresion: string);
        function CadenaAReal(Numero: string): double;
        procedure CrearPiezas();
        procedure GenerarPiezasOperador(operA: char; operB: char; inicia: integer);

    public
        // Uso del chequeo de sintaxis
        Sintaxis: TEvaluaSintaxis;

        Constructor Create();
        function Analizar(expresionA: string): boolean;
        function Evaluar(): double;
        procedure DarValorVariable(varAlgebra: char; valor: double);
    end;

implementation

Constructor TEvaluador3.Create;
begin
    //Constantes de los diferentes tipos de datos que tendrán las piezas
    self.ESFUNCION := 1;
    self.ESPARABRE := 2;
    self.ESPARCIERRA := 3;
    self.ESOPERADOR := 4;
    self.ESNUMERO := 5;
    self.ESVARIABLE := 6;
    self.ESACUMULA := 7;
end;
```

```

// Analiza la expresión
function TEvaluador3.Analizar(expresionA: string): boolean;
var
    expresionB: string;
    chequeo: boolean;
begin
    Sintaxis := TEvaluaSintaxis.Create();
    expresionB := Sintaxis.Transforma(expresionA);
    chequeo := Sintaxis.SintaxisCorrecta(expresionB);
    if (chequeo) then
    begin
        Partes.Free;
        Piezas.Free;
        Partes := TObjectList.Create;
        Piezas := TObjectList.Create;
        CrearPartes(expresionB);
        CrearPiezas();
    end;
    Result := chequeo;
end;

// Divide la expresión en partes
procedure TEvaluador3.CrearPartes(expresion: string);
var
    NuevoA: string;
    NuevoB: string;
    Numero: string;
    pos: integer;
    car: char;
    objeto: TParte;
begin
    // Debe analizarse con paréntesis
    NuevoA := '(' + expresion + ')';

    // Reemplaza las funciones de tres letras por una letra mayúscula
    NuevoB := StringReplace(NuevoA, 'sen', 'A', [rfReplaceAll, rfIgnoreCase]);
    NuevoB := StringReplace(NuevoB, 'cos', 'B', [rfReplaceAll, rfIgnoreCase]);
    NuevoB := StringReplace(NuevoB, 'tan', 'C', [rfReplaceAll, rfIgnoreCase]);
    NuevoB := StringReplace(NuevoB, 'abs', 'D', [rfReplaceAll, rfIgnoreCase]);
    NuevoB := StringReplace(NuevoB, 'asn', 'E', [rfReplaceAll, rfIgnoreCase]);
    NuevoB := StringReplace(NuevoB, 'acs', 'F', [rfReplaceAll, rfIgnoreCase]);
    NuevoB := StringReplace(NuevoB, 'atn', 'G', [rfReplaceAll, rfIgnoreCase]);
    NuevoB := StringReplace(NuevoB, 'log', 'H', [rfReplaceAll, rfIgnoreCase]);
    NuevoB := StringReplace(NuevoB, 'cei', 'I', [rfReplaceAll, rfIgnoreCase]);
    NuevoB := StringReplace(NuevoB, 'exp', 'J', [rfReplaceAll, rfIgnoreCase]);
    NuevoB := StringReplace(NuevoB, 'sqr', 'K', [rfReplaceAll, rfIgnoreCase]);
    NuevoB := StringReplace(NuevoB, 'rcb', 'L', [rfReplaceAll, rfIgnoreCase]);

    // Va de caracter en caracter
    Numero := '';
    pos := 1;
    while (pos <= Length(NuevoB)) do
    begin
        car := NuevoB[pos];

        // Si es un número lo va acumulando en una cadena
        if (car >= '0') and (car <= '9') or (car = '.') then
        begin
            Numero := Numero + car;
        end

        // Si es un operador entonces agrega número (si existía)
        else if (car = '+') or (car = '-') or (car = '*') or (car = '/') or (car = '^') then
        begin
            if (Length(Numero) > 0) then
            begin
                objeto := TParte.Create(ESNUMERO, -1, '0', CadenaAReal(Numero), 0);
                Partes.Add(objeto);
                Numero := '';
            end;
            objeto := TParte.Create(ESOPERADOR, -1, car, 0, 0);
            Partes.Add(objeto);
        end

        // Si es variable
        else if (car >= 'a') and (car <= 'z') then
        begin
            objeto := TParte.Create(ESVARIABLE, -1, '0', 0, ord(car) - ord('a'));

```

```

    Partes.Add(objeto);
end

// Si es una función (seno, coseno, tangente, ...)
else if (car >= 'A') and (car <= 'L') then
begin
    objeto := TParte.Create(ESFUNCION, ord(car) - ord('A'), '0', 0, 0);
    Partes.Add(objeto);
    Inc(pos);
end

// Si es un paréntesis que abre
else if (car = '(') then
begin
    objeto := TParte.Create(ESPARABRE, -1, '0', 0, 0);
    Partes.Add(objeto);
end

// Si es un paréntesis que cierra
else
begin
    if (Length(Número) > 0) then
    begin
        objeto := TParte.Create(ESNUMERO, -1, '0', CadenaAReal(Número), 0);
        Partes.Add(objeto);
        Número := '';
    end;

    // 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] as TParte).Tipo = ESPARABRE) or ((Partes[Partes.Count - 2] as
TParte).Tipo = ESFUNCION) then
    begin
        objeto := TParte.Create(ESOPERADOR, -1, '+', 0, 0);
        Partes.Add(objeto);
        objeto := TParte.Create(ESNUMERO, -1, '0', 0, 0);
        Partes.Add(objeto);
    end;

    objeto := TParte.Create(ESPARCIERRA, -1, '0', 0, 0);
    Partes.Add(objeto);
end;
Inc(pos);
end;
end;

// Convierte un número almacenado en una cadena a su valor real
function TEvaluador3.CadenaAReal(Número: string): double;
var
    parteEntera: double;
    cont: integer;
    parteDecimal: double;
    multiplica: double;
    numeroB: double;
    num: integer;
begin
    // Parte entera
    parteEntera := 0;
    for cont := 1 to length(Número) do
    begin
        if (Número[cont] = '.') then break;
        parteEntera := parteEntera * 10 + (ord(Número[cont]) - ord('0'));
    end;

    // Parte decimal
    parteDecimal := 0;
    multiplica := 1;
    for num := cont + 1 to length(Número) do
    begin
        parteDecimal := parteDecimal * 10 + (ord(Número[num]) - ord('0'));
        multiplica := multiplica * 10;
    end;

    numeroB := parteEntera + parteDecimal / multiplica;
    Result := numeroB;
end;

// Ahora convierte las partes en las piezas finales de ejecución
procedure TEvaluador3.CrearPiezas();

```



```

var
  cont: integer;
begin
  cont := Partes.Count - 1;
  repeat
    if ((Partes[cont] as TParte).Tipo = ESPARABRE) or ((Partes[cont] as TParte).Tipo = ESFUNCION) then
    begin
      GenerarPiezasOperador('^', '^', cont); // Evalúa las potencias
      GenerarPiezasOperador('*', '/', cont); // Luego evalúa multiplicar y dividir
      GenerarPiezasOperador('+', '-', cont); // Finalmente evalúa sumar y restar

      if ((Partes[cont] as TParte).Tipo = ESFUNCION) then // Agrega la función a la última pieza
      begin
        (Piezas[Piezas.Count - 1] as TPieza).Funcion := (Partes[cont] as TParte).Funcion;
      end;

      // Quita el paréntesis/función que abre y el que cierra, dejando el centro
      Partes.Delete(cont);
      Partes.Delete(cont + 1);
    end;
    Dec(cont);
  until not (cont >= 0);
end;

// Genera las piezas buscando determinado operador
procedure TEvaluador3.GenerarPiezasOperador(operA: char; operB: char; inicia: integer);
var
  cont: integer;
  objeto: TPieza;
begin
  cont := inicia + 1;
  repeat
    if ((Partes[cont] as TParte).Tipo = ESOPERADOR) and ((Partes[cont] as TParte).Operador = operA) or
    ((Partes[cont] as TParte).Operador = operB) then
    begin
      // Crea Pieza
      objeto := TPieza.Create(-1,
        (Partes[cont - 1] as TParte).Tipo, (Partes[cont - 1] as TParte).Numero,
        (Partes[cont - 1] as TParte).UnaVariable, (Partes[cont - 1] as TParte).Acumulador,
        (Partes[cont] as TParte).Operador,
        (Partes[cont + 1] as TParte).Tipo, (Partes[cont + 1] as TParte).Numero,
        (Partes[cont + 1] as TParte).UnaVariable, (Partes[cont + 1] as TParte).Acumulador);
      Piezas.Add(objeto);

      // Elimina la parte del operador y la siguiente
      Partes.Delete(cont);
      Partes.Delete(cont);

      // Cambia la parte anterior por parte que acumula
      (Partes[cont - 1] as TParte).Tipo := ESACUMULA;
      (Partes[cont - 1] as TParte).Acumulador := Piezas.Count-1;

      // Retorna el contador en uno para tomar la siguiente operación
      Dec(cont);
    end;
    Inc(cont);
  until not ((Partes[cont] as TParte).Tipo <> ESPARCIERRA);
end;

// Evalúa la expresión convertida en piezas
function TEvaluador3.Evaluar(): double;
var
  resultado: double;
  numA: double;
  numB: double;
  pos: integer;
begin
  resultado := 0;

  for pos := 0 to Piezas.Count-1 do
  begin
    if (Piezas[pos] as TPieza).TipoA = ESNUMERO then begin numA := (Piezas[pos] as TPieza).NumeroA; end
    else if ((Piezas[pos] as TPieza).TipoA = ESVARIABLE) then begin numA := VariableAlgebra[(Piezas[pos] as
TPieza).VariableA]; end
    else begin numA := (Piezas[(Piezas[pos] as TPieza).PiezaA] as TPieza).ValorPieza; end;

    if ((Piezas[pos] as TPieza).TipoB = ESNUMERO) then begin numB := (Piezas[pos] as TPieza).NumeroB; end

```



```

else if ((Piezas[pos] as TPieza).TipoB = ESVARIABLE) then begin numB := VariableAlgebra[(Piezas[pos] as
TPieza).VariableB]; end
else begin numB := (Piezas[(Piezas[pos] as TPieza).PiezaB] as TPieza).ValorPieza; end;

try
if ((Piezas[pos] as TPieza).Operador = '*') then begin resultado := numA * numB; end
else if ((Piezas[pos] as TPieza).Operador = '/') then begin resultado := numA / numB; end
else if ((Piezas[pos] as TPieza).Operador = '+') then begin resultado := numA + numB; end
else if ((Piezas[pos] as TPieza).Operador = '-') then begin resultado := numA - numB; end
else begin resultado := power(numA, numB); end;

case ((Piezas[pos] as TPieza).Funcion) of
0: begin resultado := sin(resultado); end;
1: begin resultado := cos(resultado); end;
2: begin resultado := tan(resultado); end;
3: begin resultado := abs(resultado); end;
4: begin resultado := arcsin(resultado); end;
5: begin resultado := arccos(resultado); end;
6: begin resultado := arctan(resultado); end;
7: begin resultado := ln(resultado); end;
8: begin resultado := ceil(resultado); end;
9: begin resultado := exp(resultado); end;
10: begin resultado := sqrt(resultado); end;
11: begin resultado := power(resultado, 0.33333333333333333333); end;
end;

except //Captura el error matemático
on EMathError do
begin
Result := NaN;
Exit;
end;
end;
(Piezas[pos] as TPieza).ValorPieza := resultado;
end;
Result := resultado;
end;

// Da valor a las variables que tendrá la expresión algebraica
procedure TEvaluador3.DarValorVariable(varAlgebra: char; valor: double);
begin
VariableAlgebra[ord(varAlgebra) - ord('a')] := valor;
end;

end.

```

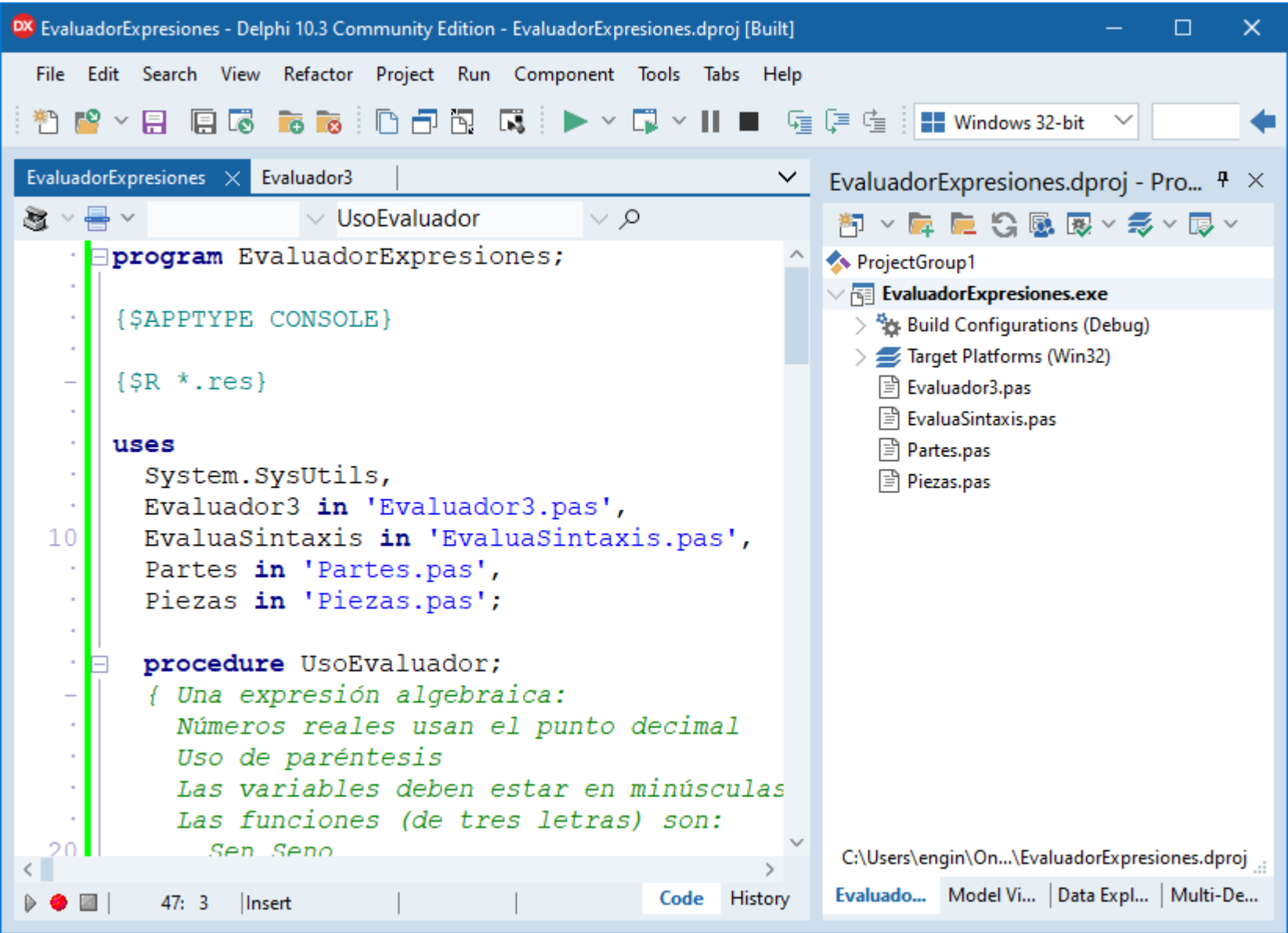


Ilustración 5: Proyecto en Delphi 10.3 Community Edition

EvaluadorExpresiones.dpr

```
program EvaluadorExpresiones;

{$APPTYPE CONSOLE}

{$R *.res}

uses
  System.SysUtils,
  Evaluador3 in 'Evaluador3.pas',
  EvaluaSintaxis in 'EvaluaSintaxis.pas',
  Partes in 'Partes.pas',
  Piezas in 'Piezas.pas';

procedure UsoEvaluador;
{ Una expresión algebraica:
  Números reales usan el punto decimal
  Uso de paréntesis
  Las variables deben estar en minúsculas van de la 'a' a la 'z' excepto ñ
  Las funciones (de tres letras) son:
    Sen Seno
    Cos Coseno
    Tan Tangente
    Abs Valor absoluto
    Asn Arcoseno
    Acs Arcocoseno
    Atn Arcotangente
    Log Logaritmo Natural
    Ceil Valor techo
    Exp Exponencial
    Sqr Raíz cuadrada
    Rcb Raíz Cúbica
  Los operadores son:
    + (suma)
    - (resta)
    * (multiplicación)
    / (división)
    ^ (potencia)
  No se acepta el "-" unario. Luego expresiones como: 4*-2 o (-5+3) o (-x^2) o (-x)^2 son inválidas.
}
```

```

var
  expresion: string;
  evaluador: TEvaluador3;
  resultado: double;
  num: integer;
  valor: double;
  unError: integer;
begin
  expresion := 'Cos(0.004 * x) - (Tan(1.78 / k + h) * SEN(k ^ x) + abs (k^3-h^2))';

  //Instancia el evaluador
  evaluador := TEvaluador3.Create;

  Randomize;

  //Analiza la expresión (valida sintaxis)
  if evaluador.Analizar(expresion) then
  begin
    //Si no hay fallos de sintaxis, puede evaluar la expresión

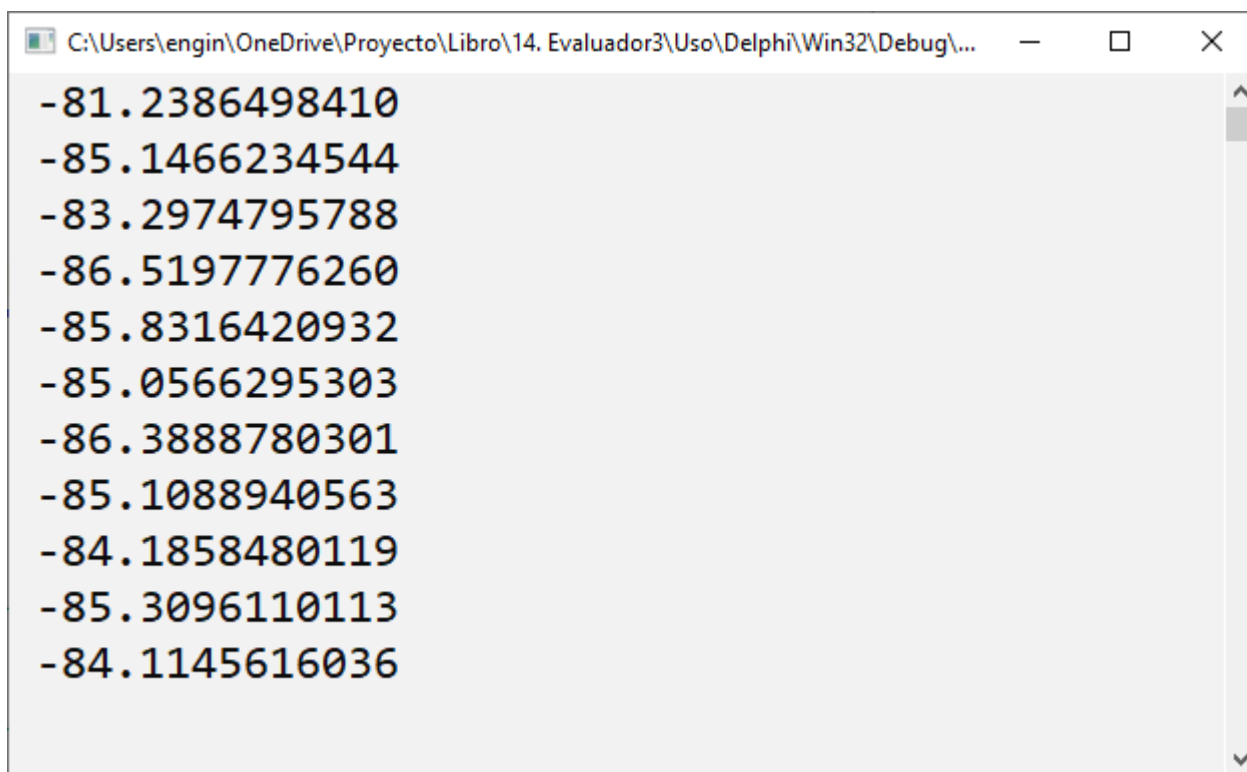
    //Da valores a las variables que deben estar en minúsculas
    evaluador.DarValorVariable('k', 1.6);
    evaluador.DarValorVariable('x', -8.3);
    evaluador.DarValorVariable('h', 9.29);

    //Evalúa la expresión
    resultado := evaluador.Evaluar();
    WriteLn(resultado:15:10);

    //Evalúa con ciclos
    for num := 1 to 10 do
    begin
      valor := Random();
      evaluador.DarValorVariable('k', valor);
      resultado := evaluador.Evaluar();
      WriteLn(resultado:15:10);
    end;
  end
else
begin
  //Si se detectó un error de sintaxis
  for unError := 0 to Length(evaluador.Sintaxis.EsCorrecto) do
  begin
    //Muestra que error de sintaxis se produjo
    if evaluador.Sintaxis.EsCorrecto[unError] = false then
    begin
      WriteLn(evaluador.Sintaxis.MensajesErrorSintaxis(unError));
    end;
  end;
end;
end;
end;

begin
  UsoEvaluador;
  ReadLn;
end.

```



A screenshot of a Windows command prompt window. The title bar shows the file path: C:\Users\engin\OneDrive\Proyecto\Libro\14. Evaluador3\Uso\Delphi\Win32\Debug\... The window contains a list of 11 negative floating-point numbers, each on a new line. The numbers are: -81.2386498410, -85.1466234544, -83.2974795788, -86.5197776260, -85.8316420932, -85.0566295303, -86.3888780301, -85.1088940563, -84.1858480119, -85.3096110113, and -84.1145616036. The window has a standard Windows interface with a title bar, a maximize button, and a scrollbar on the right side.

```
C:\Users\engin\OneDrive\Proyecto\Libro\14. Evaluador3\Uso\Delphi\Win32\Debug\...  
-81.2386498410  
-85.1466234544  
-83.2974795788  
-86.5197776260  
-85.8316420932  
-85.0566295303  
-86.3888780301  
-85.1088940563  
-84.1858480119  
-85.3096110113  
-84.1145616036
```

Java

```
package evaluador3;

public class EvaluaSintaxis {
    /* Mensajes de error de sintaxis */
    private String[] _mensajeError = {
        "0. Caracteres no permitidos. Ejemplo: 3$5+2",
        "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)"
    };

    public boolean[] EsCorrecto = new boolean[27];

    /* Retorna si el caracter es un operador matemático */
    private boolean EsUnOperador(char car) {
        return car == '+' || car == '-' || car == '*' || car == '/' || car == '^';
    }

    /* Retorna si el caracter es un número */
    private boolean EsUnNumero(char car) {
        return car >= '0' && car <= '9';
    }

    /* Retorna si el caracter es una letra */
    private boolean EsUnaLetra(char car) {
        return car >= 'a' && car <= 'z';
    }

    /* 0. Detecta si hay un caracter no válido */
    private boolean BuenaSintaxis00(String expresion) {
        boolean Resultado = true;
        String permitidos = "abcdefghijklmnopqrstuvwxyz0123456789.+*/^()";
        for (int pos = 0; pos < expresion.length() && Resultado; pos++)
            if (permitidos.indexOf(expresion.charAt(pos)) == -1)
                Resultado = false;
        return Resultado;
    }

    /* 1. Un número seguido de una letra. Ejemplo: 2q-(*3) */
    private boolean BuenaSintaxis01(String expresion) {
        boolean Resultado = true;
        for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
            char carA = expresion.charAt(pos);
            char carB = expresion.charAt(pos+1);
            if (EsUnNumero(carA) && EsUnaLetra(carB)) Resultado = false;
        }
        return Resultado;
    }
}
```

```

/* 2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6) */
private boolean BuenaSintaxis02(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (EsUnNumero(carA) && carB == '(') Resultado = false;
    }
    return Resultado;
}

/* 3. Doble punto seguido. Ejemplo: 3..1 */
private boolean BuenaSintaxis03(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == '.' && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 4. Punto seguido de operador. Ejemplo: 3.*1 */
private boolean BuenaSintaxis04(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == '.' && EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 5. Un punto y sigue una letra. Ejemplo: 3+5.w-8 */
private boolean BuenaSintaxis05(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == '.' && EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3 */
private boolean BuenaSintaxis06(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == '.' && carB == '(') Resultado = false;
    }
    return Resultado;
}

/* 7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3 */
private boolean BuenaSintaxis07(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == '.' && carB == ')') Resultado = false;
    }
    return Resultado;
}

/* 8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7 */
private boolean BuenaSintaxis08(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (EsUnOperador(carA) && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3 */

```

```

private boolean BuenaSintaxis09(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (EsUnOperador(carA) && EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7 */
private boolean BuenaSintaxis10(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (EsUnOperador(carA) && carB == ')') Resultado = false;
    }
    return Resultado;
}

/* 11. Una letra seguida de número. Ejemplo: 7-2a-6 */
private boolean BuenaSintaxis11(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (EsUnaLetra(carA) && EsUnNumero(carB)) Resultado = false;
    }
    return Resultado;
}

/* 12. Una letra seguida de punto. Ejemplo: 7-a.-6 */
private boolean BuenaSintaxis12(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (EsUnaLetra(carA) && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 13. Un paréntesis que abre seguido de punto. Ejemplo: 7-(.4-6) */
private boolean BuenaSintaxis13(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == '(' && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 14. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3) */
private boolean BuenaSintaxis14(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == '(' && EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 15. Un paréntesis que abre seguido de un paréntesis que cierra. Ejemplo: 7-()-6 */
private boolean BuenaSintaxis15(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == '(' && carB == ')') Resultado = false;
    }
    return Resultado;
}

/* 16. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7 */
private boolean BuenaSintaxis16(String expresion) {
    boolean Resultado = true;

```



```

    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == ')') && EsUnNumero(carB)) Resultado = false;
    }
    return Resultado;
}

/* 17. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5). */
private boolean BuenaSintaxis17(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == ')') && carB == '.') Resultado = false;
    }
    return Resultado;
}

/* 18. Un paréntesis que cierra y sigue una letra. Ejemplo: (3-5)t */
private boolean BuenaSintaxis18(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == ')') && EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 19. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: (3-5)(4*5) */
private boolean BuenaSintaxis19(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (carA == ')') && carB == '(') Resultado = false;
    }
    return Resultado;
}

/* 20. Si hay dos letras seguidas (después de quitar las funciones), es un error */
private boolean BuenaSintaxis20(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (EsUnaLetra(carA) && EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 21. Los paréntesis estén desbalanceados. Ejemplo: 3-(2*4)) */
private boolean BuenaSintaxis21(String expresion) {
    int parabre = 0; /* Contador de paréntesis que abre */
    int parcierra = 0; /* Contador de paréntesis que cierra */
    for (int pos = 0; pos < expresion.length(); pos++) {
        switch (expresion.charAt(pos)) {
            case '(': parabre++; break;
            case ')': parcierra++; break;
        }
    }
    return parcierra == parabre;
}

/* 22. Doble punto en un número de tipo real. Ejemplo: 7-6.46.1+2 */
private boolean BuenaSintaxis22(String expresion) {
    boolean Resultado = true;
    int totalpuntos = 0; /* Validar los puntos decimales de un número real */
    for (int pos = 0; pos < expresion.length() && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        if (EsUnOperador(carA)) totalpuntos = 0;
        if (carA == '.') totalpuntos++;
        if (totalpuntos > 1) Resultado = false;
    }
    return Resultado;
}

/* 23. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4"; */

```

```

private boolean BuenaSintaxis23(String expresion) {
    boolean Resultado = true;
    int parabre = 0; /* Contador de paréntesis que abre */
    int parcierra = 0; /* Contador de paréntesis que cierra */
    for (int pos = 0; pos < expresion.length() && Resultado; pos++) {
        switch (expresion.charAt(pos)) {
            case '(': parabre++; break;
            case ')': parcierra++; break;
        }
        if (parcierra > parabre) Resultado = false;
    }
    return Resultado;
}

/* 24. Inicia con operador. Ejemplo: +3*5 */
private boolean BuenaSintaxis24(String expresion) {
    char carA = expresion.charAt(0);
    return !EsUnOperador(carA);
}

/* 25. Finaliza con operador. Ejemplo: 3*5* */
private boolean BuenaSintaxis25(String expresion) {
    char carA = expresion.charAt(expresion.length() - 1);
    return !EsUnOperador(carA);
}

/* 26. Encuentra una letra seguida de paréntesis que abre. Ejemplo: 3-a(7)-5 */
private boolean BuenaSintaxis26(String expresion) {
    boolean Resultado = true;
    for (int pos = 0; pos < expresion.length() - 1 && Resultado; pos++) {
        char carA = expresion.charAt(pos);
        char carB = expresion.charAt(pos+1);
        if (EsUnaLetra(carA) && carB == '(') Resultado = false;
    }
    return Resultado;
}

public boolean SintaxisCorrecta(String ecuacion) {
    /* Reemplaza las funciones de tres letras por una letra */
    String expresion = ecuacion.replace("sen(", "a+(").replace("cos(", "a+(").replace("tan(",
"a+(").replace("abs(", "a+(").replace("asn(", "a+(").replace("acs(", "a+(").replace("atn(",
"a+(").replace("log(", "a+(").replace("cei(", "a+(").replace("exp(", "a+(").replace("sqr(",
"a+(").replace("rcb(", "a+(");

    /* Hace las pruebas de sintaxis */
    EsCorrecto[0] = BuenaSintaxis00(expresion);
    EsCorrecto[1] = BuenaSintaxis01(expresion);
    EsCorrecto[2] = BuenaSintaxis02(expresion);
    EsCorrecto[3] = BuenaSintaxis03(expresion);
    EsCorrecto[4] = BuenaSintaxis04(expresion);
    EsCorrecto[5] = BuenaSintaxis05(expresion);
    EsCorrecto[6] = BuenaSintaxis06(expresion);
    EsCorrecto[7] = BuenaSintaxis07(expresion);
    EsCorrecto[8] = BuenaSintaxis08(expresion);
    EsCorrecto[9] = BuenaSintaxis09(expresion);
    EsCorrecto[10] = BuenaSintaxis10(expresion);
    EsCorrecto[11] = BuenaSintaxis11(expresion);
    EsCorrecto[12] = BuenaSintaxis12(expresion);
    EsCorrecto[13] = BuenaSintaxis13(expresion);
    EsCorrecto[14] = BuenaSintaxis14(expresion);
    EsCorrecto[15] = BuenaSintaxis15(expresion);
    EsCorrecto[16] = BuenaSintaxis16(expresion);
    EsCorrecto[17] = BuenaSintaxis17(expresion);
    EsCorrecto[18] = BuenaSintaxis18(expresion);
    EsCorrecto[19] = BuenaSintaxis19(expresion);
    EsCorrecto[20] = BuenaSintaxis20(expresion);
    EsCorrecto[21] = BuenaSintaxis21(expresion);
    EsCorrecto[22] = BuenaSintaxis22(expresion);
    EsCorrecto[23] = BuenaSintaxis23(expresion);
    EsCorrecto[24] = BuenaSintaxis24(expresion);
    EsCorrecto[25] = BuenaSintaxis25(expresion);
    EsCorrecto[26] = BuenaSintaxis26(expresion);

    boolean Resultado = true;
    for (int cont = 0; cont < EsCorrecto.length && Resultado; cont++)
        if (EsCorrecto[cont] == false) Resultado = false;
    return Resultado;
}

```

```

/* Transforma la expresión para ser chequeada y analizada */
public String Transforma(String expresion) {
    /* Quita espacios, tabuladores y la vuelve a minúsculas */
    String nuevo = "";
    for (int num = 0; num < expresion.length(); num++) {
        char letra = expresion.charAt(num);
        if (letra >= 'A' && letra <= 'Z') letra += ' ';
        if (letra != ' ' && letra != '\t') nuevo += letra;
    }

    /* Cambia los )) por )+0) porque es requerido al crear las piezas */
    while (nuevo.indexOf("))") != -1) nuevo = nuevo.replace("))", ") +0)");

    return nuevo;
}

/* Muestra mensaje de error sintáctico */
public String MensajesErrorSintaxis(int codigoError) {
    return _mensajeError[codigoError];
}
}

```

```
package evaluador3;

public class Parte {
    public int Tipo; /* Acumulador, función, paréntesis que abre, paréntesis que cierra, operador, número,
variable */
    public int Funcion; /* 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 char Operador; /* + suma - resta * multiplicación / división ^ potencia */
    public double Numero; /* Número literal, por ejemplo: 3.141592 */
    public int UnaVariable; /* Variable algebraica */
    public int Acumulador; /* Usado cuando la expresión se convierte en piezas. Por ejemplo:
        3 + 2 / 5 se convierte así:
        |3| |+| |2| | / | |5|
        |3| |+| |A| A es un identificador de acumulador */

    public Parte(int tipo, int funcion, char operador, double numero, int unaVariable) {
        Tipo = tipo;
        Funcion = funcion;
        Operador = operador;
        Numero = numero;
        UnaVariable = unaVariable;
    }
}
```

```
package evaluador3;

public class Pieza {
    public double ValorPieza; /* Almacena el valor que genera la pieza al evaluarse */
    public int Funcion; /* 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 TipoA; /* La primera parte es un número o una variable o trae el valor de otra pieza */
    public double NumeroA; /* Es un número literal */
    public int VariableA; /* Es una variable */
    public int PiezaA; /* Trae el valor de otra pieza */
    public char Operador; /* + suma - resta * multiplicación / división ^ potencia */
    public int TipoB; /* La segunda parte es un número o una variable o trae el valor de otra pieza */
    public double NumeroB; /* Es un número literal */
    public int VariableB; /* Es una variable */
    public int PiezaB; /* Trae el valor de otra pieza */

    public Pieza(int funcion, int tipoA, double numeroA, int variableA, int piezaA, char operador, int tipoB,
double numeroB, int variableB, int piezaB) {
        Funcion = funcion;

        TipoA = tipoA;
        NumeroA = numeroA;
        VariableA = variableA;
        PiezaA = piezaA;

        Operador = operador;

        TipoB = tipoB;
        NumeroB = numeroB;
        VariableB = variableB;
        PiezaB = piezaB;
    }
}
```

```
/* Evaluador de expresiones. Versión 3.0
 * Autor: Rafael Alberto Moreno Parra
 * Fecha: 25 de abril de 2021
 *
 * Pasos para la evaluación de expresiones algebraicas
 * I. Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
 * [3.14] [+] [sen() [4] [/] [x] []] [*] [(] [7.2] [^] [3] [-] [1] [])]
 *
 * II. Toma las partes y las divide en piezas con la siguiente estructura:
 * acumula = funcion numero/variable/acumula operador numero/variable/acumula
 * Siguiendo el ejemplo anterior sería:
 * A = 7.2 ^ 3
 * B = A - 1
 * C = seno ( 4 / x )
 * D = C * B
 * E = 3.14 + D
 *
 * Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación
 *
 * */
package evaluador3;

import java.util.ArrayList;

public class Evaluador3 {
    /* Constantes de los diferentes tipos de datos que tendrán las piezas */
    private static final int ESFUNCION = 1;
    private static final int ESPARABRE = 2;
    private static final int ESPARCIERRA = 3;
    private static final int ESOPERADOR = 4;
    private static final int ESNUMERO = 5;
    private static final int ESVARIABLE = 6;
    private static final int ESACUMULA = 7;

    /* Listado de partes en que se divide la expresión
    Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
    [3.14] [+] [sen() [4] [/] [x] []] [*] [(] [7.2] [^] [3] [-] [1] [])]
    Cada parte puede tener un número, un operador, una función, un paréntesis que abre o un paréntesis que
    cierra */
    private ArrayList<Parte> Partes = new ArrayList<Parte>();

    /* ArrayListado de piezas que ejecutan
    Toma las partes y las divide en piezas con la siguiente estructura:
    acumula = funcion numero/variable/acumula operador numero/variable/acumula
    Siguiendo el ejemplo anterior sería:
    A = 7.2 ^ 3
    B = A - 1
    C = seno ( 4 / x )
    D = C * B
    E = 3.14 + D

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

    /* El arreglo unidimensional que lleva el valor de las variables */
    private double[] VariableAlgebra = new double[26];

    /* Uso del chequeo de sintaxis */
    public EvaluaSintaxis Sintaxis = new EvaluaSintaxis();

    /* Analiza la expresión */
    public Boolean Analizar(String expresionA) {
        String expresionB = Sintaxis.Transforma(expresionA);
        Boolean chequeo = Sintaxis.SintaxisCorrecta(expresionB);
        if (chequeo) {
            Partes.clear();
            Piezas.clear();
            CrearPartes(expresionB);
            CrearPiezas();
        }
        return chequeo;
    }

    /* Divide la expresión en partes */
```

```

private void CrearPartes(String expresion) {
    /* Debe analizarse con paréntesis */
    String NuevoA = "(" + expresion + ")";

    /* Reemplaza las funciones de tres letras por una letra mayúscula */
    String NuevoB = NuevoA.replace("sen", "A").replace("cos", "B").replace("tan", "C").replace("abs",
"D").replace("asn", "E").replace("acs", "F").replace("atn", "G").replace("log", "H").replace("cei",
"I").replace("exp", "J").replace("sqr", "K").replace("rcb", "L");

    /* Va de caracter en caracter */
    String Numero = "";
    for (int pos = 0; pos < NuevoB.length(); pos++) {
        char car = NuevoB.charAt(pos);
        /* Si es un número lo va acumulando en una cadena */
        if ((car >= '0' && car <= '9') || car == '.') {
            Numero += car;
        }
        /* Si es un operador entonces agrega número (si existía) */
        else if (car == '+' || car == '-' || car == '*' || car == '/' || car == '^') {
            if (Numero.length() > 0) {
                Partes.add(new Parte(ESNUMERO, -1, '0', CadenaAReal(Numero), 0));
                Numero = "";
            }
            Partes.add(new Parte(ESOPERADOR, -1, car, 0, 0));
        }
        /* Si es variable */
        else if (car >= 'a' && car <= 'z') {
            Partes.add(new Parte(ESVARIABLE, -1, '0', 0, car - 'a'));
        }
        /* Si es una función (seno, coseno, tangente, ...) */
        else if (car >= 'A' && car <= 'L') {
            Partes.add(new Parte(ESFUNCION, car - 'A', '0', 0, 0));
            pos++;
        }
        /* Si es un paréntesis que abre */
        else if (car == '(') {
            Partes.add(new Parte(ESPARABRE, -1, '0', 0, 0));
        }
        /* Si es un paréntesis que cierra */
        else {
            if (Numero.length() > 0) {
                Partes.add(new Parte(ESNUMERO, -1, '0', CadenaAReal(Numero), 0));
                Numero = "";
            }
            /* Si sólo había un número o variable dentro del paréntesis le agrega + 0 (por ejemplo: sen(x) o
3*(2) ) */
            if (Partes.get(Partes.size() - 2).Tipo == ESPARABRE || Partes.get(Partes.size() - 2).Tipo ==
ESFUNCION) {
                Partes.add(new Parte(ESOPERADOR, -1, '+', 0, 0));
                Partes.add(new Parte(ESNUMERO, -1, '0', 0, 0));
            }

            Partes.add(new Parte(ESPARCIERRA, -1, '0', 0, 0));
        }
    }
}

/* Convierte un número almacenado en una cadena a su valor real */
private double CadenaAReal(String Numero) {
    //Parte entera
    double parteEntera = 0;
    int cont;
    for (cont = 0; cont < Numero.length(); cont++) {
        if (Numero.charAt(cont) == '.') break;
        parteEntera = parteEntera * 10 + (Numero.charAt(cont) - '0');
    }

    //Parte decimal
    double parteDecimal = 0;
    double multiplica = 1;
    for (int num = cont + 1; num < Numero.length(); num++) {
        parteDecimal = parteDecimal * 10 + (Numero.charAt(num) - '0');
        multiplica *= 10;
    }

    double numero = parteEntera + parteDecimal / multiplica;
    return numero;
}

```



```

/* Ahora convierte las partes en las piezas finales de ejecución */
private void CrearPiezas() {
    int cont = Partes.size() - 1;
    do {
        Parte tmpParte = Partes.get(cont);
        if (tmpParte.Tipo == ESPARABRE || tmpParte.Tipo == ESFUNCION) {
            GenerarPiezasOperador('^', '^', cont); /* Evalúa las potencias */
            GenerarPiezasOperador('*', '/', cont); /* Luego evalúa multiplicar y dividir */
            GenerarPiezasOperador('+', '-', cont); /* Finalmente evalúa sumar y restar */

            if (tmpParte.Tipo == ESFUNCION) { /* Agrega la función a la última pieza */
                Piezas.get(Piezas.size() - 1).Funcion = tmpParte.Funcion;
            }

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

/* Genera las piezas buscando determinado operador */
private void GenerarPiezasOperador(char operA, char operB, int inicia) {
    int cont = inicia + 1;
    do {
        Parte tmpParte = Partes.get(cont);
        if (tmpParte.Tipo == ESOPERADOR && (tmpParte.Operador == operA || tmpParte.Operador == operB)) {
            Parte tmpParteIzq = Partes.get(cont - 1);
            Parte tmpParteDer = Partes.get(cont + 1);

            /* Crea Pieza */
            Piezas.add(new Pieza(-1,
                tmpParteIzq.Tipo, tmpParteIzq.Numero,
                tmpParteIzq.UnaVariable, tmpParteIzq.Acumulador,
                tmpParte.Operador,
                tmpParteDer.Tipo, tmpParteDer.Numero,
                tmpParteDer.UnaVariable, tmpParteDer.Acumulador));

            /* Elimina la parte del operador y la siguiente */
            Partes.remove(cont);
            Partes.remove(cont);

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

            /* Cambia la parte anterior por parte que acumula */
            tmpParteIzq.Tipo = ESACUMULA;
            tmpParteIzq.Acumulador = Piezas.size() - 1;
        }
        cont++;
    } while (Partes.get(cont).Tipo != ESPARCIERRA);
}

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

    for (int pos = 0; pos < Piezas.size(); pos++) {
        Pieza tmpPieza = Piezas.get(pos);
        double numA, numB;

        switch (tmpPieza.TipoA) {
            case ESNUMERO: numA = tmpPieza.NumeroA; break;
            case ESVARIABLE: numA = VariableAlgebra[tmpPieza.VariableA]; break;
            default: numA = Piezas.get(tmpPieza.PiezaA).ValorPieza; break;
        }

        switch (tmpPieza.TipoB) {
            case ESNUMERO: numB = tmpPieza.NumeroB; break;
            case ESVARIABLE: numB = VariableAlgebra[tmpPieza.VariableB]; break;
            default: numB = Piezas.get(tmpPieza.PiezaB).ValorPieza; break;
        }

        switch (tmpPieza.Operador) {
            case '*': resultado = numA * numB; break;
            case '/': resultado = numA / numB; break;
            case '+': resultado = numA + numB; break;
            case '-': resultado = numA - numB; break;
        }
    }
}

```


[illegible]

Como usar el evaluador

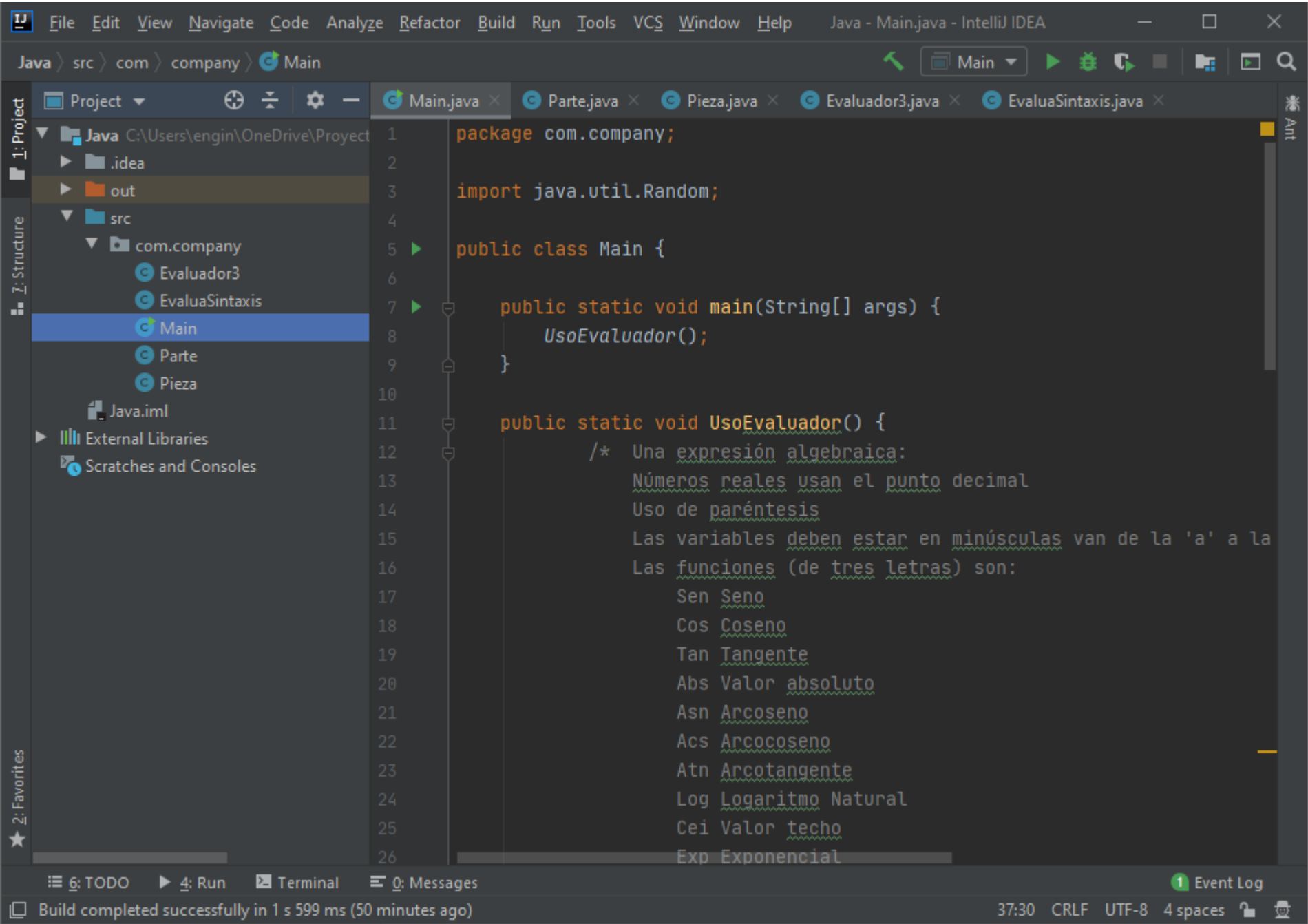


Ilustración 6: Las diferentes clases en el IDE IntelliJ IDEA

Main.java

```
package com.company;

import java.util.Random;

public class Main {

    public static void main(String[] args) {
        UsoEvaluador();
    }

    public static void UsoEvaluador() {
        /* Una expresión algebraica:
        Números reales usan el punto decimal
        Uso de paréntesis
        Las variables deben estar en minúsculas van de la 'a' a la 'z' excepto ñ
        Las funciones (de tres letras) son:
            Sen Seno
            Cos Coseno
            Tan Tangente
            Abs Valor absoluto
            Asn Arcoseno
            Acs Arcocoseno
            Atn Arcotangente
            Log Logaritmo Natural
            Cei Valor techo
            Exp Exponencial
            Sqr Raíz cuadrada
            Rcb Raíz Cúbica
        Los operadores son:
            + (suma)
            - (resta)
            * (multiplicación)
            / (división)
            ^ (potencia)
```

```

    No se acepta el "-" unario. Luego expresiones como: 4*-2 o (-5+3) o (-x^2) o (-x)^2 son inválidas.
    */
String expresion = "Cos(0.004 * x) - (Tan(1.78 / k + h) * SEN(k ^ x) + abs (k^3-h^2))";

//Instancia el evaluador
Evaluador3 evaluador = new Evaluador3();

//Analiza la expresión (valida sintaxis)
if (evaluador.Analizar(expresion)) {

    //Si no hay fallos de sintaxis, puede evaluar la expresión

    //Da valores a las variables que deben estar en minúsculas
    evaluador.DarValorVariable('k', 1.6);
    evaluador.DarValorVariable('x', -8.3);
    evaluador.DarValorVariable('h', 9.29);

    //Evalúa la expresión
    double resultado = evaluador.Evaluar();
    System.out.println(resultado);

    //Evalúa con ciclos
    Random azar = new Random();
    for (int num = 1; num <= 10; num++) {
        double valor = azar.nextDouble();
        evaluador.DarValorVariable('k', valor);
        resultado = evaluador.Evaluar();
        System.out.println(resultado);
    }
}
else {
    //Si se detectó un error de sintaxis
    for (int unError = 0; unError < evaluador.Sintaxis.EsCorrecto.length; unError++) {
        //Muestra que error de sintaxis se produjo
        if (evaluador.Sintaxis.EsCorrecto[unError] == false)
            System.out.println(evaluador.Sintaxis.MensajesErrorSintaxis(unError));
    }
}
}
}
}

```

The screenshot shows the 'Run' window in IntelliJ IDEA. The main pane displays the output of the program, which consists of ten lines of random double values generated by the 'Random' class. The values are: -81.2386498409949, -85.03324990969827, -85.34908156757764, -85.31832724952699, -86.06489291865987, -83.40397021991738, -85.08290990796868, -85.12575010366555, -85.98571466502013, and -85.98571466502013. The bottom status bar shows 'Build completed successfully in 1 s 781 ms (moments ago)'. The bottom right corner displays the time '37:30', encoding 'CRLF', 'UTF-8', and '4 spaces'.

Ilustración 7: Ejecución del código

JavaScript

```
class EvaluaSintaxis {
  constructor(){
    /* Mensajes de error de sintaxis */
    this._mensajeError = [
      '0. Caracteres no permitidos. Ejemplo: 3$5+2',
      '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)*',
    ];

    this.EsCorrecto = [];
  }

  /* Retorna si el caracter es un operador matemático */
  EsUnOperador(car) {
    return car === '+' || car === '-' || car === '*' || car === '/' || car === '^';
  }

  /* Retorna si el caracter es un número */
  EsUnNumero(car) {
    return car >= '0' && car <= '9';
  }

  /* Retorna si el caracter es una letra */
  EsUnaLetra(car) {
    return car >= 'a' && car <= 'z';
  }

  /* 0. Detecta si hay un caracter no válido */
  BuenaSintaxis00(expresion) {
    var Resultado = true;
    var permitidos = 'abcdefghijklmnopqrstuvwxyz0123456789.+*/^()';
    for (var pos = 0; pos < expresion.length && Resultado; pos++) {
      if (permitidos.indexOf(expresion[pos]) === -1)
        Resultado = false;
    }
    return Resultado;
  }

  /* 1. Un número seguido de una letra. Ejemplo: 2q-(*3) */
  BuenaSintaxis01(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
      var carA = expresion[pos];
      var carB = expresion[pos + 1];
      if (this.EsUnNumero(carA) && this.EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
  }
}
```

```

/* 2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6) */
BuenaSintaxis02(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (this.EsUnNumero(carA) && carB === '(') Resultado = false;
    }
    return Resultado;
}

/* 3. Doble punto seguido. Ejemplo: 3..1 */
BuenaSintaxis03(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (carA === '.' && carB === '.') Resultado = false;
    }
    return Resultado;
}

/* 4. Punto seguido de operador. Ejemplo: 3.*1 */
BuenaSintaxis04(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (carA === '.' && this.EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 5. Un punto y sigue una letra. Ejemplo: 3+5.w-8 */
BuenaSintaxis05(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (carA === '.' && this.EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3 */
BuenaSintaxis06(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (carA === '.' && carB === '(') Resultado = false;
    }
    return Resultado;
}

/* 7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3 */
BuenaSintaxis07(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (carA === '.' && carB === ')') Resultado = false;
    }
    return Resultado;
}

/* 8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7 */
BuenaSintaxis08(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (this.EsUnOperador(carA) && carB === '.') Resultado = false;
    }
    return Resultado;
}

/* 9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3 */
BuenaSintaxis09(expresion) {

```

```

var Resultado = true;
for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
    var carA = expresion[pos];
    var carB = expresion[pos + 1];
    if (this.EsUnOperador(carA) && this.EsUnOperador(carB)) Resultado = false;
}
return Resultado;
}

/* 10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7 */
BuenaSintaxis10(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (this.EsUnOperador(carA) && carB === ')') Resultado = false;
    }
    return Resultado;
}

/* 11. Una letra seguida de número. Ejemplo: 7-2a-6 */
BuenaSintaxis11(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (this.EsUnaLetra(carA) && this.EsUnNumero(carB)) Resultado = false;
    }
    return Resultado;
}

/* 12. Una letra seguida de punto. Ejemplo: 7-a.-6 */
BuenaSintaxis12(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (this.EsUnaLetra(carA) && carB === '.') Resultado = false;
    }
    return Resultado;
}

/* 13. Un paréntesis que abre seguido de punto. Ejemplo: 7-(.4-6) */
BuenaSintaxis13(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (carA === '(' && carB === '.') Resultado = false;
    }
    return Resultado;
}

/* 14. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3) */
BuenaSintaxis14(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (carA === '(' && this.EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 15. Un paréntesis que abre seguido de un paréntesis que cierra. Ejemplo: 7-()-6 */
BuenaSintaxis15(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (carA === '(' && carB === ')') Resultado = false;
    }
    return Resultado;
}

/* 16. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7 */
BuenaSintaxis16(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {

```



```

    var carA = expresion[pos];
    var carB = expresion[pos + 1];
    if (carA === ')') && this.EsUnNumero(carB)) Resultado = false;
}
return Resultado;
}

/* 17. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5). */
BuenaSintaxis17(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (carA === ')') && carB === '.') Resultado = false;
    }
    return Resultado;
}

/* 18. Un paréntesis que cierra y sigue una letra. Ejemplo: (3-5)t */
BuenaSintaxis18(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (carA === ')') && this.EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 19. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: (3-5)(4*5) */
BuenaSintaxis19(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (carA === ')') && carB === '(') Resultado = false;
    }
    return Resultado;
}

/* 20. Si hay dos letras seguidas (después de quitar las funciones), es un error */
BuenaSintaxis20(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (this.EsUnaLetra(carA) && this.EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 21. Los paréntesis estén desbalanceados. Ejemplo: 3-(2*4)) */
BuenaSintaxis21(expresion) {
    var parabre = 0; /* Contador de paréntesis que abre */
    var parcierra = 0; /* Contador de paréntesis que cierra */
    for (var pos = 0; pos < expresion.length; pos++) {
        var carA = expresion[pos];
        if (carA === '(') parabre++;
        if (carA === ')') parcierra++;
    }
    return parcierra == parabre;
}

/* 22. Doble punto en un número de tipo real. Ejemplo: 7-6.46.1+2 */
BuenaSintaxis22(expresion) {
    var Resultado = true;
    var totalpuntos = 0; /* Validar los puntos decimales de un número real */
    for (var pos = 0; pos < expresion.length && Resultado; pos++) {
        var carA = expresion[pos];
        if (this.EsUnOperador(carA)) totalpuntos = 0;
        if (carA === '.') totalpuntos++;
        if (totalpuntos > 1) Resultado = false;
    }
    return Resultado;
}

/* 23. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4'; */
BuenaSintaxis23(expresion) {
    var Resultado = true;

```



```

var parabre = 0; /* Contador de paréntesis que abre */
var parcierra = 0; /* Contador de paréntesis que cierra */
for (var pos = 0; pos < expresion.length && Resultado; pos++) {
    var carA = expresion[pos];
    if (carA === '(') parabre++;
    if (carA === ')') parcierra++;
    if (parcierra > parabre) Resultado = false;
}
return Resultado;
}

/* 24. Inicia con operador. Ejemplo: +3*5 */
BuenaSintaxis24(expresion) {
    var carA = expresion[0];
    return !this.EsUnOperador(carA);
}

/* 25. Finaliza con operador. Ejemplo: 3*5* */
BuenaSintaxis25(expresion) {
    var carA = expresion[expresion.length - 1];
    return !this.EsUnOperador(carA);
}

/* 26. Encuentra una letra seguida de paréntesis que abre. Ejemplo: 3-a(7)-5 */
BuenaSintaxis26(expresion) {
    var Resultado = true;
    for (var pos = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA = expresion[pos];
        var carB = expresion[pos + 1];
        if (this.EsUnaLetra(carA) && carB === '(') Resultado = false;
    }
    return Resultado;
}

SintaxisCorrecta(expresionA) {
    /* Reemplaza las funciones de tres letras por una letra */
    var expresionB = expresionA.replaceAll("sen(", "a+(").replaceAll("cos(", "a+(").replaceAll("tan(",
"a+(").replaceAll("abs(", "a+(").replaceAll("asn(", "a+(").replaceAll("acs(", "a+(").replaceAll("atn(",
"a+(").replaceAll("log(", "a+(").replaceAll("cei(", "a+(").replaceAll("exp(", "a+(").replaceAll("sqr(",
"a+(").replaceAll("rcb(", "a+(");

    /* Hace las pruebas de sintaxis */
    this.EsCorrecto[0] = this.BuenaSintaxis00(expresionB);
    this.EsCorrecto[1] = this.BuenaSintaxis01(expresionB);
    this.EsCorrecto[2] = this.BuenaSintaxis02(expresionB);
    this.EsCorrecto[3] = this.BuenaSintaxis03(expresionB);
    this.EsCorrecto[4] = this.BuenaSintaxis04(expresionB);
    this.EsCorrecto[5] = this.BuenaSintaxis05(expresionB);
    this.EsCorrecto[6] = this.BuenaSintaxis06(expresionB);
    this.EsCorrecto[7] = this.BuenaSintaxis07(expresionB);
    this.EsCorrecto[8] = this.BuenaSintaxis08(expresionB);
    this.EsCorrecto[9] = this.BuenaSintaxis09(expresionB);
    this.EsCorrecto[10] = this.BuenaSintaxis10(expresionB);
    this.EsCorrecto[11] = this.BuenaSintaxis11(expresionB);
    this.EsCorrecto[12] = this.BuenaSintaxis12(expresionB);
    this.EsCorrecto[13] = this.BuenaSintaxis13(expresionB);
    this.EsCorrecto[14] = this.BuenaSintaxis14(expresionB);
    this.EsCorrecto[15] = this.BuenaSintaxis15(expresionB);
    this.EsCorrecto[16] = this.BuenaSintaxis16(expresionB);
    this.EsCorrecto[17] = this.BuenaSintaxis17(expresionB);
    this.EsCorrecto[18] = this.BuenaSintaxis18(expresionB);
    this.EsCorrecto[19] = this.BuenaSintaxis19(expresionB);
    this.EsCorrecto[20] = this.BuenaSintaxis20(expresionB);
    this.EsCorrecto[21] = this.BuenaSintaxis21(expresionB);
    this.EsCorrecto[22] = this.BuenaSintaxis22(expresionB);
    this.EsCorrecto[23] = this.BuenaSintaxis23(expresionB);
    this.EsCorrecto[24] = this.BuenaSintaxis24(expresionB);
    this.EsCorrecto[25] = this.BuenaSintaxis25(expresionB);
    this.EsCorrecto[26] = this.BuenaSintaxis26(expresionB);

    var Resultado = true;
    for (var cont = 0; cont < this.EsCorrecto.length && Resultado; cont++)
        if (this.EsCorrecto[cont] === false) Resultado = false;
    return Resultado;
}

/* Transforma la expresión para ser chequeada y analizada */
Transforma(expresion) {
    /* Quita espacios, tabuladores y la vuelve a minúsculas */

```

```

var nuevo = "";
for (var num = 0; num < expresion.length; num++) {
    var letra = expresion[num];
    if (letra >= 'A' && letra <= 'Z') letra = String.fromCharCode(letra.charCodeAt(0) + '
'.charCodeAt(0));
    if (letra != ' ' && letra != '\n') nuevo += letra;
}

/* Cambia los )) por )+0) porque es requerido al crear las piezas */
while (nuevo.indexOf("))") != -1) nuevo = nuevo.replaceAll("))", ") +0)");

return nuevo;
}

/* Muestra mensaje de error sintáctico */
MensajesErrorSintaxis(CodigoError) {
    return this._mensajeError[CodigoError];
}
}

```

```
/* ***** */
class Parte {
  constructor(tipo, funcion, operador, numero, unaVariable) {
    this.Tipo = tipo; /* Acumulador, función, paréntesis que abre, paréntesis que cierra, operador, número,
variable */
    this.Funcion = funcion; /* 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 */
    this.Operador = operador; /* + suma - resta * multiplicación / división ^ potencia */
    this.Numero = numero; /* Número literal, por ejemplo: 3.141592 */
    this.UnaVariable = unaVariable; /* Variable algebraica */
    this.Acumulador = 0; /* Usado cuando la expresión se convierte en piezas. Por ejemplo:
      3 + 2 / 5   se convierte así:
      |3| |+| |2| | / | |5|
      |3| |+| |A|   A es un identificador de acumulador */
  }
}
```

```
/* ***** */
class Pieza {
  constructor(funcion, tipoA, numeroA, variableA, piezaA, operador, tipoB, numeroB, variableB, piezaB) {
    this.ValorPieza = 0; /* Almacena el valor que genera la pieza al evaluarse */
    this.Funcion = funcion; /* 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 */
    this.TipoA = tipoA; /* La primera parte es un número o una variable o trae el valor de otra pieza */
    this.NumeroA = numeroA; /* Es un número literal */
    this.VariableA = variableA; /* Es una variable */
    this.PiezaA = piezaA; /* Trae el valor de otra pieza */

    this.Operador = operador; /* + suma - resta * multiplicación / división ^ potencia */

    this.TipoB = tipoB; /* La segunda parte es un número o una variable o trae el valor de otra pieza */
    this.NumeroB = numeroB; /* Es un número literal */
    this.VariableB = variableB; /* Es una variable */
    this.PiezaB = piezaB; /* Trae el valor de otra pieza */
  }
}
```

```
/* Evaluador de expresiones. Versión 3.0
 * Autor: Rafael Alberto Moreno Parra
 * Fecha: 25 de abril de 2021
 *
 * Pasos para la evaluación de expresiones algebraicas
 * I. Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
 * [3.14] [+] [sen() [4] [/] [x] []) [*] [(] [7.2] [^] [3] [-] [1] [])]
 *
 * II. Toma las partes y las divide en piezas con la siguiente estructura:
 * acumula = funcion numero/variable/acumula operador numero/variable/acumula
 * Siguiendo el ejemplo anterior sería:
 * A = 7.2 ^ 3
 * B = A - 1
 * C = seno ( 4 / x )
 * D = C * B
 * E = 3.14 + D
 *
 * Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación
 *
 */

class Evaluador3 {
  constructor() {
    /* Constantes de los diferentes tipos de datos que tendrán las piezas */
    this.ESFUNCION = 1;
    this.ESPARABRE = 2;
    this.ESPARCIERRA = 3;
    this.ESOPERADOR = 4;
    this.ESNUMERO = 5;
    this.ESVARIABLE = 6;
    this.ESACUMULA = 7;

    /* Listado de partes en que se divide la expresión
      Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
      |3.14| |+| |sen(| |4| | / | |x| |)| |*| |( | |7.2| |^| |3| |-| |1| |)|
      Cada parte puede tener un número, un operador, una función, un paréntesis que abre o un paréntesis
que cierra */
    this.Partes = [];

    /* Listado de piezas que ejecutan
      Toma las partes y las divide en piezas con la siguiente estructura:
      acumula = funcion numero/variable/acumula operador numero/variable/acumula
      Siguiendo el ejemplo anterior sería:
      A = 7.2^3
      B = A - 1
      C = seno ( 4 / x )
      D = C * B
      E = 3.14 + D

      Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación */
    this.Piezas = [];

    /* El arreglo unidimensional que lleva el valor de las variables */
    this.VariableAlgebra = [];

    /* Uso del chequeo de sintaxis */
    this.Sintaxis = new EvaluaSintaxis();
  }

  /* Analiza la expresión */
  Analizar(expresionA) {
    var expresionB = this.Sintaxis.Transforma(expresionA);
    var chequeo = this.Sintaxis.SintaxisCorrecta(expresionB);
    if (chequeo === true) {
      this.Partes.length = 0;
      this.Piezas.length = 0;
      this.CrearPartes(expresionB);
      this.CrearPiezas();
    }
    return chequeo;
  }

  /* Divide la expresión en partes */
  CrearPartes(expresion) {
    /* Debe analizarse con paréntesis */
    var NuevoA = "(" + expresion + ")";
```

```

/* Reemplaza las funciones de tres letras por una letra mayúscula */
var NuevoB = NuevoA.replaceAll("sen", "A").replaceAll("cos", "B").replaceAll("tan",
"C").replaceAll("abs", "D").replaceAll("asn", "E").replaceAll("acs", "F").replaceAll("atn",
"G").replaceAll("log", "H").replaceAll("cei", "I").replaceAll("exp", "J").replaceAll("sqr",
"K").replaceAll("rcb", "L");

/* Va de caracter en caracter */
var Numero = "";
for (var pos = 0; pos < NuevoB.length; pos++) {
    var car = NuevoB[pos];
    /* Si es un número lo va acumulando en una cadena */
    if ((car >= '0' && car <= '9') || car == '.') {
        Numero += car;
    }
    /* Si es un operador entonces agrega número (si existía) */
    else if (car == '+' || car == '-' || car == '*' || car == '/' || car == '^') {
        if (Numero.length > 0) {
            this.Partes.push(new Parte(this.ESNUMERO, -1, '0', this.CadenaAReal(Numero), 0));
            Numero = "";
        }
        this.Partes.push(new Parte(this.ESOPERADOR, -1, car, 0, 0));
    }
    /* Si es variable */
    else if (car >= 'a' && car <= 'z') {
        this.Partes.push(new Parte(this.ESVARIABLE, -1, '0', 0, car.charCodeAt(0) - 'a'.charCodeAt(0)));
    }
    /* Si es una función (seno, coseno, tangente, ...) */
    else if (car >= 'A' && car <= 'L') {
        this.Partes.push(new Parte(this.ESFUNCION, car.charCodeAt(0) - 'A'.charCodeAt(0), '0', 0, 0));
        pos++;
    }
    /* Si es un paréntesis que abre */
    else if (car == '(') {
        this.Partes.push(new Parte(this.ESPARABRE, -1, '0', 0, 0));
    }
    /* Si es un paréntesis que cierra */
    else {
        if (Numero.length > 0) {
            this.Partes.push(new Parte(this.ESNUMERO, -1, '0', this.CadenaAReal(Numero), 0));
            Numero = "";
        }
        /* Si sólo había un número o variable dentro del paréntesis le agrega + 0 (por ejemplo: sen(x) o
3*(2) ) */
        if (this.Partes[this.Partes.length - 2].Tipo == this.ESPARABRE || this.Partes[this.Partes.length -
2].Tipo == this.ESFUNCION) {
            this.Partes.push(new Parte(this.ESOPERADOR, -1, '+', 0, 0));
            this.Partes.push(new Parte(this.ESNUMERO, -1, '0', 0, 0));
        }

        this.Partes.push(new Parte(this.ESPARCIERRA, -1, '0', 0, 0));
    }
}
}

/* Convierte un número almacenado en una cadena a su valor real */
CadenaAReal(Numero) {
    //Parte entera
    var parteEntera = 0;
    var cont = 0;
    for (cont = 0; cont < Numero.length; cont++) {
        if (Numero[cont].charCodeAt(0) === '.'.charCodeAt(0)) break;
        parteEntera = parteEntera * 10 + (Numero[cont].charCodeAt(0) - '0'.charCodeAt(0));
    }

    //Parte decimal
    var parteDecimal = 0;
    var multiplica = 1;
    for (var num = cont + 1; num < Numero.length; num++) {
        parteDecimal = parteDecimal * 10 + (Numero[num].charCodeAt(0) - '0'.charCodeAt(0));
        multiplica *= 10;
    }

    var numero = parteEntera + parteDecimal / multiplica;
    return numero;
}

/* Ahora convierte las partes en las piezas finales de ejecución */
CrearPiezas() {

```

```

var cont = this.Partes.length - 1;
do {
    var tmpParte = this.Partes[cont];
    if (tmpParte.Tipo === this.ESPARABRE || tmpParte.Tipo === this.ESFUNCION) {
        this.GenerarPiezasOperador('^', '^', cont); /* Evalúa las potencias */
        this.GenerarPiezasOperador('*', '/', cont); /* Luego evalúa multiplicar y dividir */
        this.GenerarPiezasOperador('+', '-', cont); /* Finalmente evalúa sumar y restar */

        if (tmpParte.Tipo === this.ESFUNCION) { /* Agrega la función a la última pieza */
            this.Piezas[this.Piezas.length - 1].Funcion = tmpParte.Funcion;
        }

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

GenerarPiezasOperador(operA, operB, ini) {
    var cont = ini + 1;
    do {
        var tmpParte = this.Partes[cont];
        if (tmpParte.Tipo === this.ESOPERADOR && (tmpParte.Operador === operA || tmpParte.Operador === operB))
        {
            var tmpParteIzq = this.Partes[cont - 1];
            var tmpParteDer = this.Partes[cont + 1];
            /* Crea Pieza */
            this.Piezas.push(new Pieza(-1,
                tmpParteIzq.Tipo, tmpParteIzq.Numero,
                tmpParteIzq.UnaVariable, tmpParteIzq.Acumulador,
                tmpParte.Operador,
                tmpParteDer.Tipo, tmpParteDer.Numero,
                tmpParteDer.UnaVariable, tmpParteDer.Acumulador));

            /* Elimina la parte del operador y la siguiente */
            this.Partes.splice(cont, 1);
            this.Partes.splice(cont, 1);

            /* Retorna el contador en uno para tomar la siguiente operación */
            cont -= 1;

            /* Cambia la parte anterior por parte que acumula */
            tmpParteIzq.Tipo = this.ESACUMULA;
            tmpParteIzq.Acumulador = this.Piezas.length-1;
        }
        cont++;
    } while (this.Partes[cont].Tipo !== this.ESPARCIERRA);
}

/* Evalúa la expresión convertida en piezas */
Evaluar() {
    var numA, numB, resultado = 0;
    for (var pos = 0; pos < this.Piezas.length; pos++) {
        var tmpPieza = this.Piezas[pos];

        switch (tmpPieza.TipoA) {
            case this.ESNUMERO: numA = tmpPieza.NumeroA; break;
            case this.ESVARIABLE: numA = this.VariableAlgebra[tmpPieza.VariableA]; break;
            default: numA = this.Piezas[tmpPieza.PiezaA].ValorPieza; break;
        }

        switch (tmpPieza.TipoB) {
            case this.ESNUMERO: numB = tmpPieza.NumeroB; break;
            case this.ESVARIABLE: numB = this.VariableAlgebra[tmpPieza.VariableB]; break;
            default: numB = this.Piezas[tmpPieza.PiezaB].ValorPieza; break;
        }

        switch (tmpPieza.Operador) {
            case '*': resultado = numA * numB; break;
            case '/': resultado = numA / numB; break;
            case '+': resultado = numA + numB; break;
            case '-': resultado = numA - numB; break;
            default: resultado = Math.pow(numA, numB); break;
        }
        if (isNaN(resultado) || !isFinite(resultado)) return resultado;

        switch (tmpPieza.Funcion) {

```

[illegible]


```
<!DOCTYPE HTML><html><body>
<script type="text/javascript" src="evaluador3.js"> </script>
<script type="text/javascript">
UsoEvaluador();

function UsoEvaluador(){
  /* Una expresión algebraica:
  Números reales usan el punto decimal
  Uso de paréntesis
  Las variables deben estar en minúsculas van de la 'a' a la 'z' excepto ñ
  Las funciones (de tres letras) son:
    Sen  Seno
    Cos  Coseno
    Tan  Tangente
    Abs  Valor absoluto
    Asn  Arcoseno
    Acs  Arcocoseno
    Atn  Arcotangente
    Log  Logaritmo Natural
    Cei  Valor techo
    Exp  Exponencial
    Sqr  Raíz cuadrada
    Rcb  Raíz Cúbica
  Los operadores son:
    + (suma)
    - (resta)
    * (multiplicación)
    / (división)
    ^ (potencia)
  No se acepta el "-" unario. Luego expresiones como: 4*-2 o (-5+3) o (-x^2) o (-x)^2 son inválidas.
  */
  var expresion = "Cos(0.004 * x) - (Tan(1.78 / k + h) * SEN(k ^ x) + abs (k^3-h^2))";

  //Instancia el evaluador
  var evaluador = new Evaluador3();

  //Analiza la expresión (valida sintaxis)
  if (evaluador.Analizar(expresion)===true){
    //Si no hay fallos de sintaxis, puede evaluar la expresión

    //Da valores a las variables que deben estar en minúsculas
    evaluador.DarValorVariable('k', 1.6);
    evaluador.DarValorVariable('x', -8.3);
    evaluador.DarValorVariable('h', 9.29);

    //Evalúa la expresión
    var resultado = evaluador.Evaluar();
    document.write(resultado + "<br>");

    //Evalúa con ciclos
    for (var num = 1; num <= 10; num++) {
      var valor = Math.random();
      evaluador.DarValorVariable('k', valor);
      resultado = evaluador.Evaluar();
      document.write(resultado + "<br>");
    }
  }
  else {
    for(var unError = 0; unError < evaluador.Sintaxis.EsCorrecto.length; unError++){
      if (evaluador.Sintaxis.EsCorrecto[unError] === false) {
        document.write(evaluador.Sintaxis.MensajesErrorSintaxis(unError)+"<br>");
      }
    }
  }
}
</script>
</body></html>
```

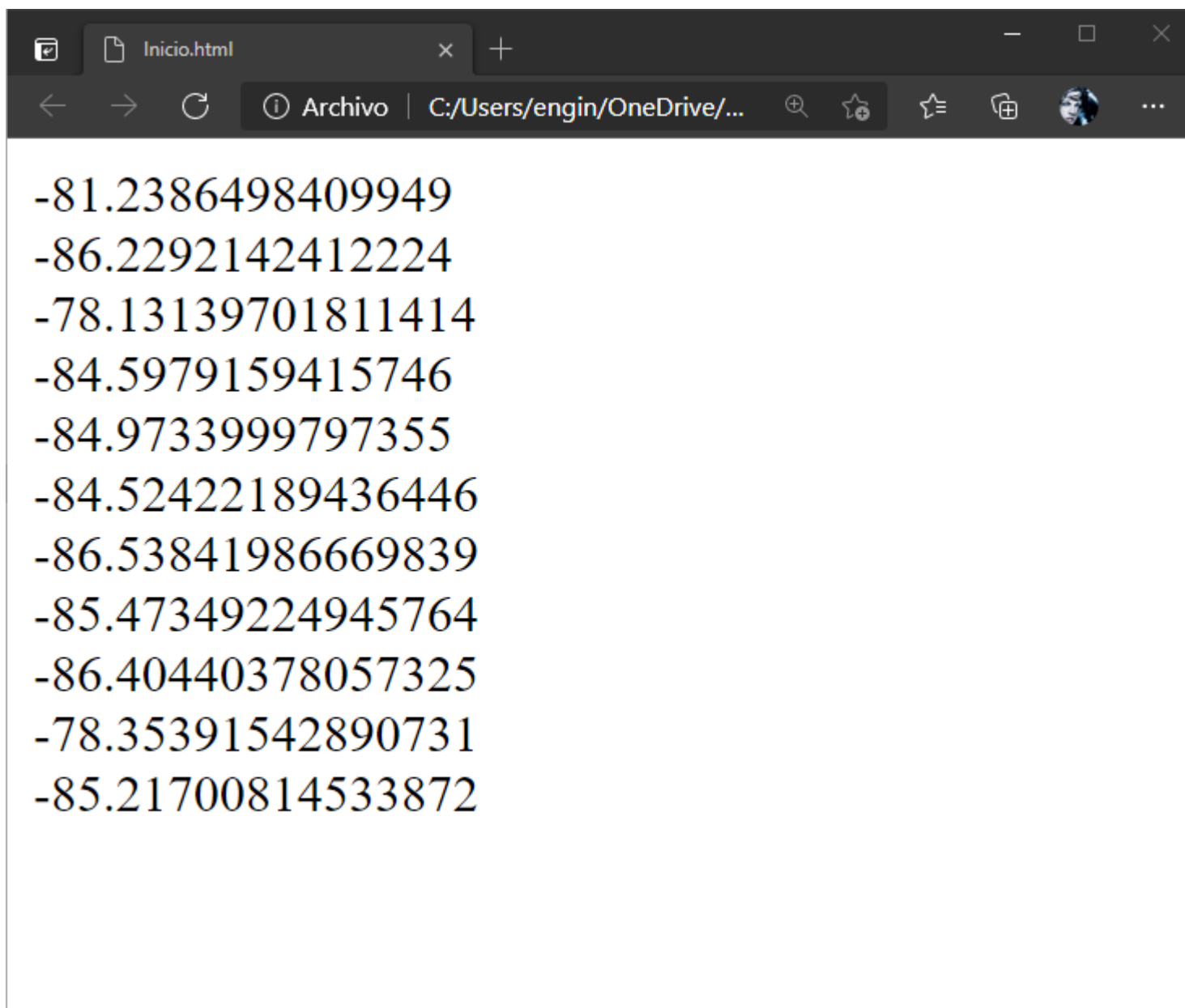


Ilustración 8: Ejecución del programa escrito en JavaScript

PHP

```
<?php
class EvaluaSintaxis {
    /* Mensajes de error de sintaxis */
    var $_mensajeError = [
        "0. Caracteres no permitidos. Ejemplo: 3$5+2",
        "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)"
    ];

    var $EsCorrecto = array();

    /* Retorna si el caracteres un operador matemático */
    function EsUnOperador($car) {
        return $car === '+' || $car === '-' || $car === '*' || $car === '/' || $car === '^';
    }

    /* Retorna si el caracter es un número */
    function EsUnNumero($car) {
        return $car >= '0' && $car <= '9';
    }

    /* Retorna si el caracter es una letra */
    function EsUnaLetra($car) {
        return $car >= 'a' && $car <= 'z';
    }

    /* 0. Detecta si hay un caracter no válido */
    function BuenaSintaxis00($expresion) {
        $Resultado = true;
        $permitidos = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
        for ($pos = 0; $pos < strlen($expresion) && $Resultado; $pos++)
            if (strpos($permitidos, $expresion[$pos]) === false)
                $Resultado = false;
        return $Resultado;
    }

    /* 1. Un número seguido de una letra. Ejemplo: 2q-(*3) */
    function BuenaSintaxis01($expresion) {
        $Resultado = true;
        for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
            $carA = $expresion[$pos];
            $carB = $expresion[$pos+1];
            if ($this->EsUnNumero($carA) && $this->EsUnaLetra($carB)) $Resultado = false;
        }
        return $Resultado;
    }

    /* 2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6) */
    function BuenaSintaxis02($expresion) {
```

```

$Resultado = true;
for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
    $carA = $expresion[$pos];
    $carB = $expresion[$pos+1];
    if ($this->EsUnNumero($carA) && $carB === '(') $Resultado = false;
}
return $Resultado;
}

/* 3. Doble punto seguido. Ejemplo: 3..1 */
function BuenaSintaxis03($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($carA === '.' && $carB === '.') $Resultado = false;
    }
    return $Resultado;
}

/* 4. Punto seguido de operador. Ejemplo: 3.*1 */
function BuenaSintaxis04($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($carA === '.' && $this->EsUnOperador($carB)) $Resultado = false;
    }
    return $Resultado;
}

/* 5. Un punto y sigue una letra. Ejemplo: 3+5.w-8 */
function BuenaSintaxis05($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($carA === '.' && $this->EsUnaLetra($carB)) $Resultado = false;
    }
    return $Resultado;
}

/* 6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3 */
function BuenaSintaxis06($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($carA === '.' && $carB === '(') $Resultado = false;
    }
    return $Resultado;
}

/* 7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3 */
function BuenaSintaxis07($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($carA === '.' && $carB === ')') $Resultado = false;
    }
    return $Resultado;
}

/* 8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7 */
function BuenaSintaxis08($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($this->EsUnOperador($carA) && $carB === '.') $Resultado = false;
    }
    return $Resultado;
}

/* 9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3 */
function BuenaSintaxis09($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {

```

```

    $carA = $expresion[$pos];
    $carB = $expresion[$pos+1];
    if ($this->EsUnOperador($carA) && $this->EsUnOperador($carB)) $Resultado = false;
}
return $Resultado;
}

/* 10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7 */
function BuenaSintaxis10($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($this->EsUnOperador($carA) && $carB === ')') $Resultado = false;
    }
    return $Resultado;
}

/* 11. Una letra seguida de número. Ejemplo: 7-2a-6 */
function BuenaSintaxis11($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($this->EsUnaLetra($carA) && $this->EsUnNumero($carB)) $Resultado = false;
    }
    return $Resultado;
}

/* 12. Una letra seguida de punto. Ejemplo: 7-a.-6 */
function BuenaSintaxis12($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($this->EsUnaLetra($carA) && $carB === '.') $Resultado = false;
    }
    return $Resultado;
}

/* 13. Un paréntesis que abre seguido de punto. Ejemplo: 7-(.4-6) */
function BuenaSintaxis13($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($carA === '(' && $carB === '.') $Resultado = false;
    }
    return $Resultado;
}

/* 14. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3) */
function BuenaSintaxis14($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($carA === '(' && $this->EsUnOperador($carB)) $Resultado = false;
    }
    return $Resultado;
}

/* 15. Un paréntesis que abre seguido de un paréntesis que cierra. Ejemplo: 7-()-6 */
function BuenaSintaxis15($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($carA === '(' && $carB === ')') $Resultado = false;
    }
    return $Resultado;
}

/* 16. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7 */
function BuenaSintaxis16($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];

```

```

    if ($carA === ')') && $this->EsUnNumero($carB)) $Resultado = false;
}
return $Resultado;
}

/* 17. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5). */
function BuenaSintaxis17($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($carA === ')') && $carB === '.') $Resultado = false;
    }
    return $Resultado;
}

/* 18. Un paréntesis que cierra y sigue una letra. Ejemplo: (3-5)t */
function BuenaSintaxis18($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($carA === ')') && $this->EsUnaLetra($carB)) $Resultado = false;
    }
    return $Resultado;
}

/* 19. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: (3-5)(4*5) */
function BuenaSintaxis19($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($carA === ')') && $carB === '(') $Resultado = false;
    }
    return $Resultado;
}

/* 20. Si hay dos letras seguidas (después de quitar las funciones), es un error */
function BuenaSintaxis20($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($this->EsUnaLetra($carA) && $this->EsUnaLetra($carB)) $Resultado = false;
    }
    return $Resultado;
}

/* 21. Los paréntesis estén desbalanceados. Ejemplo: 3-(2*4)) */
function BuenaSintaxis21($expresion) {
    $parabre = 0; /* Contador de paréntesis que abre */
    $parcierra = 0; /* Contador de paréntesis que cierra */
    for ($pos = 0; $pos < strlen($expresion); $pos++) {
        switch ($expresion[$pos]) {
            case '(': $parabre++; break;
            case ')': $parcierra++; break;
        }
    }
    return $parcierra == $parabre;
}

/* 22. Doble punto en un número de tipo real. Ejemplo: 7-6.46.1+2 */
function BuenaSintaxis22($expresion) {
    $Resultado = true;
    $totalpuntos = 0; /* Validar los puntos decimales de un número real */
    for ($pos = 0; $pos < strlen($expresion) && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        if ($this->EsUnOperador($carA)) $totalpuntos = 0;
        if ($carA === '.') $totalpuntos++;
        if ($totalpuntos > 1) $Resultado = false;
    }
    return $Resultado;
}

/* 23. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4"; */
function BuenaSintaxis23($expresion) {
    $Resultado = true;
    $parabre = 0; /* Contador de paréntesis que abre */

```



```

$parcierra = 0; /* Contador de paréntesis que cierra */
for ($pos = 0; $pos < strlen($expresion) && $Resultado; $pos++) {
    switch ($expresion[$pos]) {
        case '(': $parabre++; break;
        case ')': $parcierra++; break;
    }
    if ($parcierra > $parabre) $Resultado = false;
}
return $Resultado;
}

/* 24. Inicia con operador. Ejemplo: +3*5 */
function BuenaSintaxis24($expresion) {
    return !$this->EsUnOperador($expresion[0]);
}

/* 25. Finaliza con operador. Ejemplo: 3*5* */
function BuenaSintaxis25($expresion) {
    return !$this->EsUnOperador($expresion[strlen($expresion) - 1]);
}

/* 26. Encuentra una letra seguida de paréntesis que abre. Ejemplo: 3-a(7)-5 */
function BuenaSintaxis26($expresion) {
    $Resultado = true;
    for ($pos = 0; $pos < strlen($expresion) - 1 && $Resultado; $pos++) {
        $carA = $expresion[$pos];
        $carB = $expresion[$pos+1];
        if ($this->EsUnaLetra($carA) && $carB === '(') $Resultado = false;
    }
    return $Resultado;
}

function SintaxisCorrecta($ecuacion) {
    /* Reemplaza las funciones de tres letras por una letra mayúscula */
    $expresion = $ecuacion;
    $expresion = str_replace("sen(", "a+", $expresion);
    $expresion = str_replace("cos(", "a+", $expresion);
    $expresion = str_replace("tan(", "a+", $expresion);
    $expresion = str_replace("abs(", "a+", $expresion);
    $expresion = str_replace("asn(", "a+", $expresion);
    $expresion = str_replace("acs(", "a+", $expresion);
    $expresion = str_replace("atn(", "a+", $expresion);
    $expresion = str_replace("log(", "a+", $expresion);
    $expresion = str_replace("cei(", "a+", $expresion);
    $expresion = str_replace("exp(", "a+", $expresion);
    $expresion = str_replace("sqr(", "a+", $expresion);
    $expresion = str_replace("rcb(", "a+", $expresion);

    /* Hace las pruebas de sintaxis */
    $this->EsCorrecto[0] = $this->BuenaSintaxis00($expresion);
    $this->EsCorrecto[1] = $this->BuenaSintaxis01($expresion);
    $this->EsCorrecto[2] = $this->BuenaSintaxis02($expresion);
    $this->EsCorrecto[3] = $this->BuenaSintaxis03($expresion);
    $this->EsCorrecto[4] = $this->BuenaSintaxis04($expresion);
    $this->EsCorrecto[5] = $this->BuenaSintaxis05($expresion);
    $this->EsCorrecto[6] = $this->BuenaSintaxis06($expresion);
    $this->EsCorrecto[7] = $this->BuenaSintaxis07($expresion);
    $this->EsCorrecto[8] = $this->BuenaSintaxis08($expresion);
    $this->EsCorrecto[9] = $this->BuenaSintaxis09($expresion);
    $this->EsCorrecto[10] = $this->BuenaSintaxis10($expresion);
    $this->EsCorrecto[11] = $this->BuenaSintaxis11($expresion);
    $this->EsCorrecto[12] = $this->BuenaSintaxis12($expresion);
    $this->EsCorrecto[13] = $this->BuenaSintaxis13($expresion);
    $this->EsCorrecto[14] = $this->BuenaSintaxis14($expresion);
    $this->EsCorrecto[15] = $this->BuenaSintaxis15($expresion);
    $this->EsCorrecto[16] = $this->BuenaSintaxis16($expresion);
    $this->EsCorrecto[17] = $this->BuenaSintaxis17($expresion);
    $this->EsCorrecto[18] = $this->BuenaSintaxis18($expresion);
    $this->EsCorrecto[19] = $this->BuenaSintaxis19($expresion);
    $this->EsCorrecto[20] = $this->BuenaSintaxis20($expresion);
    $this->EsCorrecto[21] = $this->BuenaSintaxis21($expresion);
    $this->EsCorrecto[22] = $this->BuenaSintaxis22($expresion);
    $this->EsCorrecto[23] = $this->BuenaSintaxis23($expresion);
    $this->EsCorrecto[24] = $this->BuenaSintaxis24($expresion);
    $this->EsCorrecto[25] = $this->BuenaSintaxis25($expresion);
    $this->EsCorrecto[26] = $this->BuenaSintaxis26($expresion);

    $Resultado = true;
    for ($cont = 0; $cont < 27 && $Resultado; $cont++)

```



```

    if ($this->EsCorrecto[$cont] === false) $Resultado = false;
    return $Resultado;
}

/* Transforma la expresión para ser chequeada y analizada */
function Transforma($expresion) {
    /* Quita espacios, tabuladores y la vuelve a minúsculas */
    $nuevo = "";
    for ($num = 0; $num < strlen($expresion); $num++) {
        $letra = $expresion[$num];
        if ($letra >= 'A' && $letra <= 'Z') $letra = chr(ord($letra) + ord(' '));
        if ($letra != ' ' && $letra != '\t') $nuevo .= $letra;
    }

    /* Cambia los )) por )+0) porque es requerido al crear las piezas */
    while (strpos($nuevo, ")))") $nuevo = str_replace(")))", ") +0)", $nuevo);

    return $nuevo;
}

/* Muestra mensaje de error sintáctico */
function MensajesErrorSintaxis($codigoError) {
    return $this->_mensajeError[$codigoError];
}
}

```

```
class Parte {
    public $Tipo; /* Acumulador, función, paréntesis que abre, paréntesis que cierra, operador, número,
variable */
    public $Funcion; /* 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 $Operador; /* + suma - resta * multiplicación / división ^ potencia */
    public $Numero; /* Número literal, por ejemplo: 3.141592 */
    public $UnaVariable; /* Variable algebraica */
    public $Acumulador; /* Usado cuando la expresión se convierte en piezas. Por ejemplo:
    3 + 2 / 5 se convierte así:
    |3| |+| |2| | / | |5|
    |3| |+| |A| A es un identificador de acumulador */

    function __construct($tipo, $funcion, $operador, $numero, $unaVariable) {
        $this->Tipo = $tipo;
        $this->Funcion = $funcion;
        $this->Operador = $operador;
        $this->Numero = $numero;
        $this->UnaVariable = $unaVariable;
        $this->Acumulador = 0;
    }
}
```

```
class Pieza {
    public $ValorPieza; /* Almacena el valor que genera la pieza al evaluarse */
    public $Funcion; /* 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 $TipoA; /* La primera parte es un número o una variable o trae el valor de otra pieza */
    public $NumeroA; /* Es un número literal */
    public $VariableA; /* Es una variable */
    public $PiezaA; /* Trae el valor de otra pieza */
    public $Operador; /* + suma - resta * multiplicación / división ^ potencia */
    public $TipoB; /* La segunda parte es un número o una variable o trae el valor de otra pieza */
    public $NumeroB; /* Es un número literal */
    public $VariableB; /* Es una variable */
    public $PiezaB; /* Trae el valor de otra pieza */

    function __construct($funcion, $tipoA, $numeroA, $variableA, $piezaA, $operador, $tipoB, $numeroB,
    $variableB, $piezaB) {
        $this->Funcion = $funcion;

        $this->TipoA = $tipoA;
        $this->NumeroA = $numeroA;
        $this->VariableA = $variableA;
        $this->PiezaA = $piezaA;

        $this->Operador = $operador;

        $this->TipoB = $tipoB;
        $this->NumeroB = $numeroB;
        $this->VariableB = $variableB;
        $this->PiezaB = $piezaB;

        $this->ValorPieza = 0;
    }
}
```

```
<?php
/* Evaluador de expresiones. Versión 3.0
 * Autor: Rafael Alberto Moreno Parra
 * Fecha: 25 de marzo de 2021
 *
 * Pasos para la evaluación de expresiones algebraicas
 * I. Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
 * [3.14] [+] [sen() [4] [/] [x] []] [*] [(] [7.2] [^] [3] [-] [1] [])
 *
 * II. Toma las partes y las divide en piezas con la siguiente estructura:
 * acumula = funcion numero/variable/acumula operador numero/variable/acumula
 * Siguiendo el ejemplo anterior sería:
 * A = 7.2 ^ 3
 * B = A - 1
 * C = seno ( 4 / x )
 * D = C * B
 * E = 3.14 + D
 *
 * Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación
 *
 * */
require_once("EvaluaSintaxis.php");

class Evaluador3 {
    /* Constantes de los diferentes tipos de datos que tendrán las piezas */
    var $ESFUNCION = 1;
    var $ESPARABRE = 2;
    var $ESPARCIERRA = 3;
    var $ESOPERADOR = 4;
    var $ESNUMERO = 5;
    var $ESVARIABLE = 6;
    var $ESACUMULA = 7;

    /* Listado de partes en que se divide la expresión
     Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
     [3.14] [+] [sen() [4] [/] [x] []] [*] [(] [7.2] [^] [3] [-] [1] [])
     Cada parte puede tener un número, un operador, una función, un paréntesis que abre o un paréntesis que
     cierra */
    var $Partes = array();

    /* Listado de piezas que ejecutan
     Toma las partes y las divide en piezas con la siguiente estructura:
     acumula = funcion numero/variable/acumula operador numero/variable/acumula
     Siguiendo el ejemplo anterior sería:
     A = 7.2 ^ 3
     B = A - 1
     C = seno ( 4 / x )
     D = C * B
     E = 3.14 + D

     Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación */
    var $Piezas = array();

    /* El arreglo unidimensional que lleva el valor de las variables */
    var $VariableAlgebra = array();

    /* Uso del chequeo de sintaxis */
    var $Sintaxis;

    /* Analiza la expresión */
    public function Analizar($expresionA) {
        $this->Sintaxis = new EvaluaSintaxis();
        $expresionB = $this->Sintaxis->Transforma($expresionA);
        $chequeo = $this->Sintaxis->SintaxisCorrecta($expresionB);
        if ($chequeo) {
            unset($this->Partes);
            unset($this->Piezas);
            $this->CrearPartes($expresionB);
            $this->CrearPiezas();
        }
        return $chequeo;
    }

    /* Divide la expresión en partes */
    function CrearPartes($expresion) {
        /* Debe analizarse con paréntesis */
    }
```

```

$NuevoA = "(" . $expresion . ")";

/* Reemplaza las funciones de tres letras por una letra mayúscula */
$NuevoB = str_replace("sen", "A", $NuevoA);
$NuevoB = str_replace("cos", "B", $NuevoB);
$NuevoB = str_replace("tan", "C", $NuevoB);
$NuevoB = str_replace("abs", "D", $NuevoB);
$NuevoB = str_replace("asn", "E", $NuevoB);
$NuevoB = str_replace("acs", "F", $NuevoB);
$NuevoB = str_replace("atn", "G", $NuevoB);
$NuevoB = str_replace("log", "H", $NuevoB);
$NuevoB = str_replace("cei", "I", $NuevoB);
$NuevoB = str_replace("exp", "J", $NuevoB);
$NuevoB = str_replace("sqr", "K", $NuevoB);
$NuevoB = str_replace("rcb", "L", $NuevoB);

/* Va de caracter en caracter */
$Numero = "";
for ($pos = 0; $pos < strlen($NuevoB); $pos++) {
    $car = $NuevoB[$pos];
    /* Si es un número lo va acumulando en una cadena */
    if (($car >= '0' && $car <= '9') || $car === '.') {
        $Numero = $Numero . $car;
    }
    /* Si es un operador entonces agrega número (si existía) */
    else if ($car === '+' || $car === '-' || $car === '*' || $car === '/' || $car === '^') {
        if (strlen($Numero) > 0) {
            $objeto = new Parte($this->ESNUMERO, -1, '0', $this->CadenaAReal($Numero), 0);
            $this->Partes[] = $objeto;
            $Numero = "";
        }
        $objeto = new Parte($this->ESOPERADOR, -1, $car, 0, 0);
        $this->Partes[] = $objeto;
    }
    /* Si es variable */
    else if ($car >= 'a' && $car <= 'z') {
        $objeto = new Parte($this->ESVARIABLE, -1, '0', 0, ord($car) - ord('a'));
        $this->Partes[] = $objeto;
    }
    /* Si es una función (seno, coseno, tangente, ...) */
    else if ($car >= 'A' && $car <= 'L') {
        $objeto = new Parte($this->ESFUNCION, ord($car) - ord('A'), '0', 0, 0);
        $this->Partes[] = $objeto;
        $pos++;
    }
    /* Si es un paréntesis que abre */
    else if ($car === '(') {
        $objeto = new Parte($this->ESPARABRE, -1, '0', 0, 0);
        $this->Partes[] = $objeto;
    }
    /* Si es un paréntesis que cierra */
    else {
        if (strlen($Numero) > 0) {
            $objeto = new Parte($this->ESNUMERO, -1, '0', $this->CadenaAReal($Numero), 0);
            $this->Partes[] = $objeto;
            $Numero = "";
        }
        /* Si sólo había un número o variable dentro del paréntesis le agrega + 0 (por ejemplo: sen(x) o
3*(2) ) */
        if ($this->Partes[sizeof($this->Partes) - 2]->Tipo == $this->ESPARABRE || $this->
Partes[sizeof($this->Partes) - 2]->Tipo == $this->ESFUNCION) {
            $objeto = new Parte($this->ESOPERADOR, -1, '+', 0, 0);
            $this->Partes[] = $objeto;
            $objeto = new Parte($this->ESNUMERO, -1, '0', 0, 0);
            $this->Partes[] = $objeto;
        }

        $objeto = new Parte($this->ESPARCIERRA, -1, '0', 0, 0);
        $this->Partes[] = $objeto;
    }
}
}

/* Convierte un número almacenado en una cadena a su valor real */
function CadenaAReal($Numero) {
    //Parte entera
    $parteEntera = 0;
    $cont = 0;
    for ($cont = 0; $cont < strlen($Numero); $cont++) {

```

```

    if ($Numero[$cont] === '.') break;
    $parteEntera = $parteEntera * 10 + (ord($Numero[$cont]) - ord('0'));
}

//Parte decimal
$parteDecimal = 0;
$multiplica = 1;
for ($num = $cont + 1; $num < strlen($Numero); $num++) {
    $parteDecimal = $parteDecimal * 10 + (ord($Numero[$num]) - ord('0'));
    $multiplica *= 10;
}

$numero = $parteEntera + $parteDecimal / $multiplica;
return $numero;
}

/* Ahora convierte las partes en las piezas finales de ejecución */
function CrearPiezas() {
    $cont = sizeof($this->Partes) - 1;
    do {
        $tmpParte = $this->Partes[$cont];
        if ($tmpParte->Tipo == $this->ESPARABRE || $tmpParte->Tipo == $this->ESFUNCION) {
            $this->GenerarPiezasOperador('^', '^', $cont); /* Evalúa las potencias */
            $this->GenerarPiezasOperador('*', '/', $cont); /* Luego evalúa multiplicar y dividir */
            $this->GenerarPiezasOperador('+', '-', $cont); /* Finalmente evalúa sumar y restar */

            if ($tmpParte->Tipo == $this->ESFUNCION) { /* Agrega la función a la última pieza */
                $this->Piezas[sizeof($this->Piezas) - 1]->Funcion = $tmpParte->Funcion;
            }

            /* Quita el paréntesis/función que abre y el que cierra, dejando el centro */
            unset($this->Partes[$cont]);
            $this->Partes = array_values($this->Partes);
            unset($this->Partes[$cont+1]);
            $this->Partes = array_values($this->Partes);
        }
        $cont--;
    } while ($cont >= 0);
}

/* Genera las piezas buscando determinado operador */
function GenerarPiezasOperador($operA, $operB, $inicia) {
    $cont = $inicia + 1;
    do {
        $tmpParte = $this->Partes[$cont];
        if ($tmpParte->Tipo == $this->ESOPERADOR && ($tmpParte->Operador == $operA || $tmpParte->Operador ==
$operB)) {
            $tmpParteIzq = $this->Partes[$cont - 1];
            $tmpParteDer = $this->Partes[$cont + 1];

            /* Crea Pieza */
            $objeto = new Pieza(-1,
                $tmpParteIzq->Tipo, $tmpParteIzq->Numero,
                $tmpParteIzq->UnaVariable, $tmpParteIzq->Acumulador,
                $tmpParte->Operador,
                $tmpParteDer->Tipo, $tmpParteDer->Numero,
                $tmpParteDer->UnaVariable, $tmpParteDer->Acumulador);
            $this->Piezas[] = $objeto;

            /* Elimina la parte del operador y la siguiente */
            unset($this->Partes[$cont]);
            $this->Partes = array_values($this->Partes);
            unset($this->Partes[$cont]);
            $this->Partes = array_values($this->Partes);

            /* Cambia la parte anterior por parte que acumula */
            $tmpParteIzq->Tipo = $this->ESACUMULA;
            $tmpParteIzq->Acumulador = sizeof($this->Piezas) - 1;

            /* Retorna el contador en uno para tomar la siguiente operación */
            $cont--;
        }
        $cont++;
    } while ($this->Partes[$cont]->Tipo != $this->ESPARCIERRA);
}

/* Evalúa la expresión convertida en piezas */
public function Evaluar() {
    $resultado = 0;

```

```
for ($pos = 0; $pos < sizeof($this->Piezas); $pos++) {
    $tmpPieza = $this->Piezas[$pos];
    $numA=0;
    $numB=0;

    switch ($tmpPieza->TipoA) {
        case $this->ESNUMERO: $numA = $tmpPieza->NumeroA; break;
        case $this->ESVARIABLE: $numA = $this->VariableAlgebra[$tmpPieza->VariableA]; break;
        default: $numA = $this->Piezas[$tmpPieza->PiezaA]->ValorPieza; break;
    }

    switch ($tmpPieza->TipoB) {
        case $this->ESNUMERO: $numB = $tmpPieza->NumeroB; break;
        case $this->ESVARIABLE: $numB = $this->VariableAlgebra[$tmpPieza->VariableB]; break;
        default: $numB = $this->Piezas[$tmpPieza->PiezaB]->ValorPieza; break;
    }

    switch ($tmpPieza->Operador) {
        case '*': $resultado = $numA * $numB; break;
        case '/': if ($numB == 0) return NAN; $resultado = $numA / $numB; break;
        case '+': $resultado = $numA + $numB; break;
        case '-': $resultado = $numA - $numB; break;
        default: $resultado = pow($numA, $numB); break;
    }

    if (is_nan($resultado) || is_infinite($resultado)) return $resultado;

    switch ($tmpPieza->Funcion) {
        case 0: $resultado = sin($resultado); break;
        case 1: $resultado = cos($resultado); break;
        case 2: $resultado = tan($resultado); break;
        case 3: $resultado = abs($resultado); break;
        case 4: $resultado = asin($resultado); break;
        case 5: $resultado = acos($resultado); break;
        case 6: $resultado = atan($resultado); break;
        case 7: $resultado = log($resultado); break;
        case 8: $resultado = ceil($resultado); break;
        case 9: $resultado = exp($resultado); break;
        case 10: $resultado = sqrt($resultado); break;
        case 11: $resultado = pow($resultado, 0.333333333333333333333333333333); break;
    }

    if (is_nan($resultado) || is_infinite($resultado)) return $resultado;

    $tmpPieza->ValorPieza = $resultado;
}
return $resultado;
}

/* Da valor a las variables que tendrá la expresión algebraica */
public function DarValorVariable($varAlgebra, $valor) {
    $this->VariableAlgebra[ord($varAlgebra) - ord('a')] = $valor;
}
```

```
<?php
require_once("EvaluaSintaxis.php");
require_once("Evaluador3.php");

UsoEvaluador();

function UsoEvaluador() {
/* Una expresión algebraica:
Números reales usan el punto decimal
Uso de paréntesis
Las variables deben estar en minúsculas van de la 'a' a la 'z' excepto ñ
Las funciones (de tres letras) son:
Sen Seno
Cos Coseno
Tan Tangente
Abs Valor absoluto
Asn Arcoseno
Acs Arcocoseno
Atn Arcotangente
Log Logaritmo Natural
Cei Valor techo
Exp Exponencial
Sqr Raíz cuadrada
Rcb Raíz Cúbica
Los operadores son:
+ (suma)
- (resta)
* (multiplicación)
/ (división)
^ (potencia)
No se acepta el "-" unario. Luego expresiones como: 4*-2 o (-5+3) o (-x^2) o (-x)^2 son inválidas.
*/
$expresion = "Cos(0.004 * x) - (Tan(1.78 / k + h) * SEN(k ^ x) + abs (k^3-h^2))";

//Instancia el evaluador
$evaluador = new Evaluador3();

//Analiza la expresión (valida sintaxis)
if ($evaluador->Analizar($expresion)) {

    //Si no hay fallos de sintaxis, puede evaluar la expresión

    //Da valores a las variables que deben estar en minúsculas
    $evaluador->DarValorVariable('k', 1.6);
    $evaluador->DarValorVariable('x', -8.3);
    $evaluador->DarValorVariable('h', 9.29);

    //Evalúa la expresión
    $resultado = $evaluador->Evaluar();
    echo $resultado . "<br>";

    //Evalúa con ciclos
    for ($num = 1; $num <= 10; $num++) {
        $valor = mt_rand() / mt_getrandmax();
        $evaluador->DarValorVariable('k', $valor);
        $resultado = $evaluador->Evaluar();
        echo $resultado . "<br>";
    }
}
else {
    //Si se detectó un error de sintaxis
    for ($unError = 0; $unError < count($evaluador->Sintaxis->EsCorrecto); $unError++) {
        //Muestra que error de sintaxis se produjo
        if ($evaluador->Sintaxis->EsCorrecto[$unError] == false)
            echo $evaluador->Sintaxis->MensajesErrorSintaxis($unError) . "<br>";
    }
}
}
```

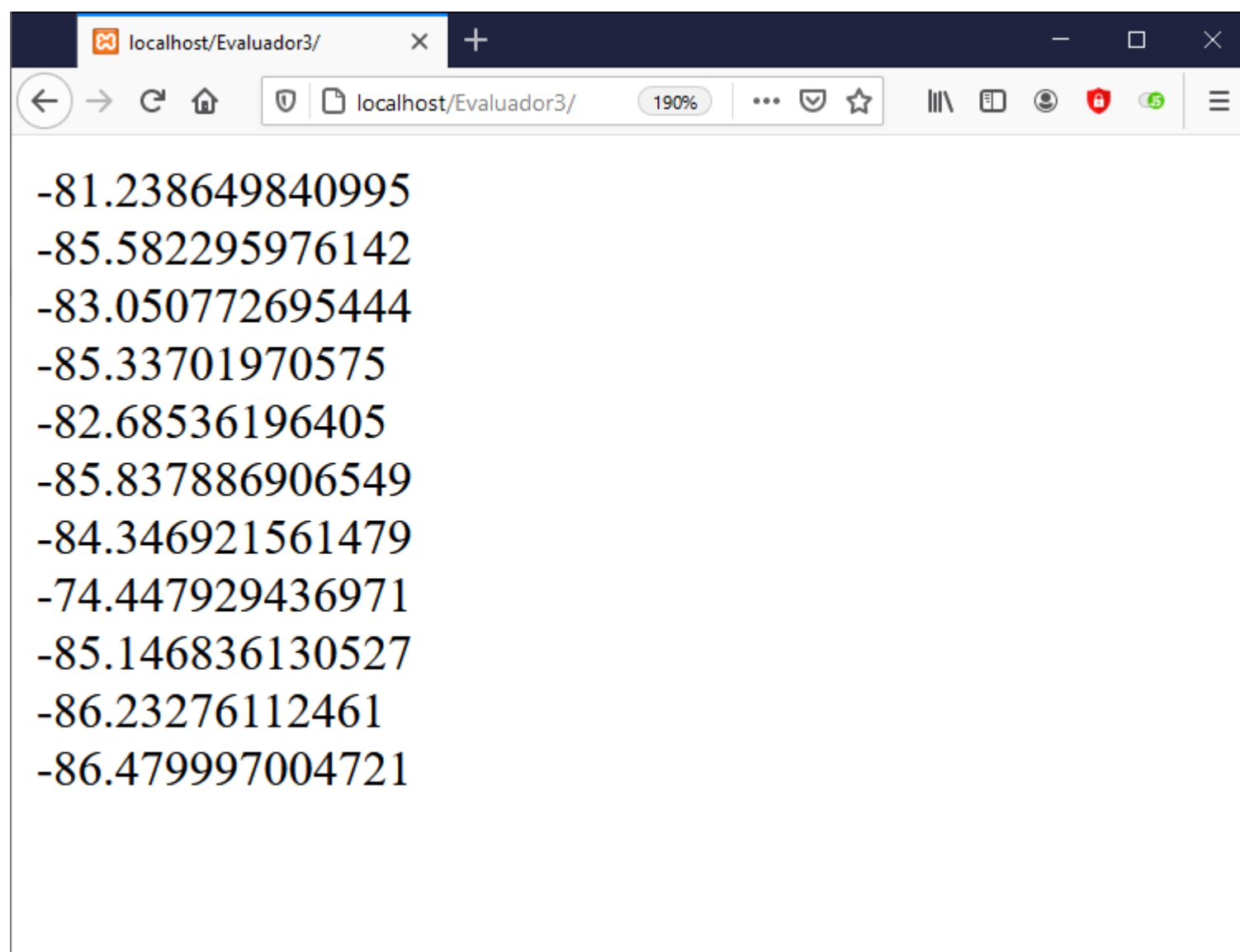



Ilustración 9: Ejecución del programa en PHP

Python

```
class EvaluaSintaxis:
    # Mensajes de error de sintaxis
    _mensajeError = [
        "0. Caracteres no permitidos. Ejemplo: 3$5+2",
        "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)"
    ]

EsCorrecto = [None] * 27;

# Retorna si el caracter es un operador matemático
def EsUnOperador(self, car):
    return car == '+' or car == '-' or car == '*' or car == '/' or car == '^'

# Retorna si el caracter es un número */
def EsUnNumero(self, car):
    return car >= '0' and car <= '9'

# Retorna si el caracter es una letra */
def EsUnaLetra(self, car):
    return car >= 'a' and car <= 'z'

# 0. Detecta si hay un caracter no válido
def BuenaSintaxis00(self, expresion):
    Resultado = True
    permitidos = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()"
    pos = 0
    while pos < len(expresion) and Resultado == True:
        if permitidos.find(expresion[pos]) == -1:
            Resultado = False
        pos = pos + 1
    return Resultado

# 1. Un número seguido de una letra. Ejemplo: 2q-(*3)
def BuenaSintaxis01(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if self.EsUnNumero(carA) and self.EsUnaLetra(carB):
            Resultado = False
        pos = pos + 1
    return Resultado

# 2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
def BuenaSintaxis02(self, expresion):
    Resultado = True
    pos = 0
```

```

while pos < len(expresion)-1 and Resultado == True:
    carA = expresion[pos]
    carB = expresion[pos + 1]
    if self.EsUnNumero(carA) and carB == '(':
        Resultado = False
    pos = pos + 1
return Resultado

# 3. Doble punto seguido. Ejemplo: 3..1
def BuenaSintaxis03(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos+1]
        if carA == '.' and carB == '.':
            Resultado = False
        pos = pos + 1
    return Resultado

# 4. Punto seguido de operador. Ejemplo: 3.*1
def BuenaSintaxis04(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if carA == '.' and self.EsUnOperador(carB):
            Resultado = False
        pos = pos + 1
    return Resultado

# 5. Un punto y sigue una letra. Ejemplo: 3+5.w-8
def BuenaSintaxis05(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if carA == '.' and self.EsUnaLetra(carB):
            Resultado = False
        pos = pos + 1
    return Resultado

# 6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3
def BuenaSintaxis06(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if carA == '.' and carB == '(':
            Resultado = False
        pos = pos + 1
    return Resultado;

# 7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3
def BuenaSintaxis07(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if carA == '.' and carB == ')':
            Resultado = False
        pos = pos + 1
    return Resultado

# 8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7
def BuenaSintaxis08(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if self.EsUnOperador(carA) and carB == '.':
            Resultado = False
        pos = pos + 1
    return Resultado

```

```

# 9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3
def BuenaSintaxis09(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if self.EsUnOperador(carA) and self.EsUnOperador(carB):
            Resultado = False
        pos = pos + 1
    return Resultado

# 10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
def BuenaSintaxis10(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if self.EsUnOperador(carA) and carB == ')':
            Resultado = False
        pos = pos + 1
    return Resultado

# 11. Una letra seguida de número. Ejemplo: 7-2a-6
def BuenaSintaxis11(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if self.EsUnaLetra(carA) and self.EsUnNumero(carB):
            Resultado = False
        pos = pos + 1
    return Resultado

# 12. Una letra seguida de punto. Ejemplo: 7-a.-6
def BuenaSintaxis12(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if self.EsUnaLetra(carA) and carB == '.':
            Resultado = False
        pos = pos + 1
    return Resultado

# 13. Un paréntesis que abre seguido de punto. Ejemplo: 7-(.4-6)
def BuenaSintaxis13(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if carA == '(' and carB == '.':
            Resultado = False
        pos = pos + 1
    return Resultado

# 14. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
def BuenaSintaxis14(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if carA == '(' and self.EsUnOperador(carB):
            Resultado = False
        pos = pos + 1
    return Resultado

# 15. Un paréntesis que abre seguido de un paréntesis que cierra. Ejemplo: 7-()-6
def BuenaSintaxis15(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]

```

```

        carB = expresion[pos + 1]
        if carA == '(' and carB == ')':
            Resultado = False
        pos = pos + 1
    return Resultado

# 16. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7
def BuenaSintaxis16(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if carA == ')' and self.EsUnNumero(carB):
            Resultado = False
        pos = pos + 1
    return Resultado

# 17. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5).
def BuenaSintaxis17(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos];
        carB = expresion[pos + 1];
        if carA == ')' and carB == '.':
            Resultado = False
        pos = pos + 1
    return Resultado

# 18. Un paréntesis que cierra y sigue una letra. Ejemplo: (3-5)t
def BuenaSintaxis18(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if carA == ')' and self.EsUnaLetra(carB):
            Resultado = False
        pos = pos + 1
    return Resultado

# 19. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: (3-5)(4*5)
def BuenaSintaxis19(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if carA == ')' and carB == '(':
            Resultado = False
        pos = pos + 1
    return Resultado;

# 20. Si hay dos letras seguidas (después de quitar las funciones), es un error
def BuenaSintaxis20(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos]
        carB = expresion[pos + 1]
        if self.EsUnaLetra(carA) and self.EsUnaLetra(carB):
            Resultado = False
        pos = pos + 1
    return Resultado

# 21. Los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
def BuenaSintaxis21(self, expresion):
    par abre = 0 # Contador de paréntesis que abre
    par cierra = 0 # Contador de paréntesis que cierra
    pos = 0
    while pos < len(expresion):
        carA = expresion[pos]
        if carA == '(': par abre += 1
        if carA == ')': par cierra += 1
        pos = pos + 1
    return par cierra == par abre;

# 22. Doble punto en un número de tipo real. Ejemplo: 7-6.46.1+2

```

```

def BuenaSintaxis22(self, expresion):
    Resultado = True
    totalpuntos = 0 # Validar los puntos decimales de un número real
    pos = 0
    while pos < len(expresion) and Resultado == True:
        carA = expresion[pos]
        if self.EsUnOperador(carA): totalpuntos = 0;
        if carA == '.': totalpuntos += 1
        if totalpuntos > 1: Resultado = False
        pos = pos + 1
    return Resultado

# 23. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4";
def BuenaSintaxis23(self, expresion):
    Resultado = True
    parabre = 0 # Contador de paréntesis que abre
    parcierra = 0 # Contador de paréntesis que cierra
    pos = 0
    while pos < len(expresion) and Resultado == True:
        carA = expresion[pos]
        if carA == '(': parabre += 1
        if carA == ')': parcierra += 1
        if parcierra > parabre:
            Resultado = False
        pos = pos + 1
    return Resultado

# 24. Inicia con operador. Ejemplo: +3*5
def BuenaSintaxis24(self, expresion):
    carA = expresion[0]
    return not self.EsUnOperador(carA)

# 25. Finaliza con operador. Ejemplo: 3*5*
def BuenaSintaxis25(self, expresion):
    carA = expresion[len(expresion) - 1]
    return not self.EsUnOperador(carA)

# 26. Encuentra una letra seguida de paréntesis que abre. Ejemplo: 3-a(7)-5
def BuenaSintaxis26(self, expresion):
    Resultado = True
    pos = 0
    while pos < len(expresion)-1 and Resultado == True:
        carA = expresion[pos];
        carB = expresion[pos + 1];
        if self.EsUnaLetra(carA) and carB == '(':
            Resultado = False
        pos = pos + 1
    return Resultado

def SintaxisCorrecta(self, ecuacion):
    # Reemplaza las funciones de tres letras por una letra mayúscula
    expresion = ecuacion.replace("sen(", "a+(").replace("cos(", "a+(").replace("tan(",
"a+(").replace("abs(", "a+(").replace("asn(", "a+(").replace("acs(", "a+(").replace("atn(",
"a+(").replace("log(", "a+(").replace("cei(", "a+(").replace("exp(", "a+(").replace("sqr(",
"a+(").replace("rcb(", "a+(");

# Hacer las pruebas de sintaxis
self.EsCorrecto[0] = self.BuenaSintaxis00(expresion);
self.EsCorrecto[1] = self.BuenaSintaxis01(expresion);
self.EsCorrecto[2] = self.BuenaSintaxis02(expresion);
self.EsCorrecto[3] = self.BuenaSintaxis03(expresion);
self.EsCorrecto[4] = self.BuenaSintaxis04(expresion);
self.EsCorrecto[5] = self.BuenaSintaxis05(expresion);
self.EsCorrecto[6] = self.BuenaSintaxis06(expresion);
self.EsCorrecto[7] = self.BuenaSintaxis07(expresion);
self.EsCorrecto[8] = self.BuenaSintaxis08(expresion);
self.EsCorrecto[9] = self.BuenaSintaxis09(expresion);
self.EsCorrecto[10] = self.BuenaSintaxis10(expresion);
self.EsCorrecto[11] = self.BuenaSintaxis11(expresion);
self.EsCorrecto[12] = self.BuenaSintaxis12(expresion);
self.EsCorrecto[13] = self.BuenaSintaxis13(expresion);
self.EsCorrecto[14] = self.BuenaSintaxis14(expresion);
self.EsCorrecto[15] = self.BuenaSintaxis15(expresion);
self.EsCorrecto[16] = self.BuenaSintaxis16(expresion);
self.EsCorrecto[17] = self.BuenaSintaxis17(expresion);
self.EsCorrecto[18] = self.BuenaSintaxis18(expresion);
self.EsCorrecto[19] = self.BuenaSintaxis19(expresion);
self.EsCorrecto[20] = self.BuenaSintaxis20(expresion);
self.EsCorrecto[21] = self.BuenaSintaxis21(expresion);

```



```

self.EsCorrecto[22] = self.BuenaSintaxis22(expresion);
self.EsCorrecto[23] = self.BuenaSintaxis23(expresion);
self.EsCorrecto[24] = self.BuenaSintaxis24(expresion);
self.EsCorrecto[25] = self.BuenaSintaxis25(expresion);
self.EsCorrecto[26] = self.BuenaSintaxis26(expresion);

Resultado = True
cont = 0
while cont < len(self.EsCorrecto) and Resultado == True:
    if self.EsCorrecto[cont] == False:
        Resultado = False;
        cont = cont + 1
return Resultado

#Transforma la expresión para ser chequeada y analizada
def Transforma(self, expresion):
    #Quita espacios, tabuladores y la vuelve a minúsculas
    nuevo = ""
    for num in range(0, len(expresion), 1):
        letra = expresion[num]
        if letra >= 'A' and letra <= 'Z':
            letra = chr(ord(letra) + ord(' '));
        if letra != ' ' and letra != '\n':
            nuevo = nuevo + letra;

    #Cambia los )) por )+0) porque es requerido al crear las piezas.
    while (nuevo.find("))") != -1):
        nuevo = nuevo.replace("))", ") +0)");

    return nuevo

# Muestra mensaje de error sintáctico
def MensajesErrorSintaxis(self, CodigoError):
    return self._mensajeError[CodigoError]

```



```
class Parte:
    Tipo = 0 # Acumulador, función, paréntesis que abre, paréntesis que cierra, operador, número, variable
    Funcion = 0 # 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
    Operador = '?' # + - * / ^
    Numero = 0 # Número literal, por ejemplo: 3.141592
    UnaVariable = 0 # Variable algebraica
    Acumulador = 0 # Usado cuando la expresión se convierte en piezas. Por ejemplo:
        # 3 + 2 / 5 se convierte así:
        #|3| |+| |2| | / | |5|
        #|3| |+| |A| A es un identificador de acumulador

    def __init__(self, tipo, funcion, operador, numero, variable):
        self.Tipo = tipo
        self.Funcion = funcion
        self.Operador = operador
        self.Numero = numero
        self.UnaVariable = variable
        self.Acumulador = 0
```

```
class Pieza:
    ValorPieza = 0 # Almacena el valor que genera la pieza al evaluarse
    Funcion = 0 # 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
    TipoA = 0 # La primera parte es un número o una variable o trae el valor de otra pieza
    NumeroA = 0 # Es un número literal
    VariableA = 0 # Es una variable
    PiezaA = 0 # Trae el valor de otra pieza
    Operador = '?' # + suma - resta * multiplicación / división ^ potencia
    TipoB = 0 # La segunda parte es un número o una variable o trae el valor de otra pieza
    NumeroB = 0 # Es un número literal
    VariableB = 0 # Es una variable
    PiezaB = 0 # Trae el valor de otra pieza

    def __init__(self, funcion, tipoA, numA, varA, piezaA, operador, tipoB, numB, varB, piezaB):
        self.Funcion = funcion

        self.TipoA = tipoA
        self.NumeroA = numA
        self.VariableA = varA
        self.PiezaA = piezaA

        self.Operador = operador

        self.TipoB = tipoB
        self.NumeroB = numB
        self.VariableB = varB
        self.PiezaB = piezaB
```

```
class Evaluador3:
    # Autor: Rafael Alberto Moreno Parra. 02 de abril de 2021
    # Constantes de los diferentes tipos de datos que tendrán las piezas
    ESFUNCION = 1
    ESPARABRE = 2
    ESPARCIERRA = 3
    ESOPERADOR = 4
    ESNUMERO = 5
    ESVARIABLE = 6
    ESACUMULA = 7

    """ Listado de partes en que se divide la expresión
    Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
    |3.14| |+| |sen(| |4| | / | |x| |)| |*| |( | |7.2| |^| |3| |-| |1| |)|
    Cada parte puede tener un número, un operador, una función, un paréntesis que abre o un paréntesis
que cierra """
    Partes = [];

    """ Listado de piezas que ejecutan
    Toma las partes y las divide en piezas con la siguiente estructura:
    acumula = funcion  numero/variable/acumula  operador  numero/variable/acumula
    Siguiendo el ejemplo anterior sería:
    A = 7.2 ^ 3
    B = A - 1
    C = seno ( 4 / x )
    D = C * B
    E = 3.14 + D

    Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación """
    Piezas = [];

    # El arreglo unidimensional que lleva el valor de las variables
    VariableAlgebra = [None] * 26;

# Uso del chequeo de sintaxis
Sintaxis = EvaluaSintaxis.EvaluaSintaxis();

# Analiza la expresión
def Analizar(self, expresionA):
    expresionB = self.Sintaxis.Transforma(expresionA);
    chequeo = self.Sintaxis.SintaxisCorrecta(expresionB);
    if (chequeo):
        self.Partes.clear();
        self.Piezas.clear();
        self.CrearPartes(expresionB);
        self.CrearPiezas();
    return chequeo

# Divide la expresión en partes
def CrearPartes(self, expresion):
    # Debe analizarse con paréntesis
    NuevoA = "(" + expresion + ")"

    # Reemplaza las funciones de tres letras por una letra mayúscula
    NuevoB = NuevoA.replace("sen", "A").replace("cos", "B").replace("tan", "C").replace("abs",
"D").replace("asn", "E").replace("acs", "F").replace("atn", "G").replace("log", "H").replace("cei",
"I").replace("exp", "J").replace("sqr", "K").replace("rcb", "L")
    Numero = ""

    pos = 0
    while (pos < len(NuevoB)):
        car = NuevoB[pos]

        if car >= '0' and car <= '9' or car == '.':
            Numero += car
        elif car == "+" or car == "-" or car == "*" or car == "/" or car == "^":
            if len(Numero) > 0:
                self.Partes.append(Parte(self.ESNUMERO, -1, "0", self.CadenaAReal(Numero), 0))
                Numero = ""
            self.Partes.append(Parte(self.ESOPERADOR, -1, car, 0, 0))
        elif car >= 'a' and car <= 'z':
            self.Partes.append(Parte(self.ESVARIABLE, -1, "0", 0, ord(car) - ord('a')))
        elif car >= 'A' and car <= 'L':
            self.Partes.append(Parte(self.ESFUNCION, ord(car) - ord('A'), "0", 0, 0))
            pos += 1
        elif car == '(':
```

```

        self.Partes.append(Parte(self.ESPARABRE, -1, "0", 0, 0))
    else:
        if len(Numero) > 0:
            self.Partes.append(Parte(self.ESNUMERO, -1, "0", self.CadenaAReal(Numero), 0))
            Numero = ""

            if self.Partes[len(self.Partes) - 2].Tipo == self.ESPARABRE or
self.Partes[len(self.Partes) - 2].Tipo == self.ESFUNCION:
                self.Partes.append(Parte(self.ESOPERADOR, -1, "+", 0, 0))
                self.Partes.append(Parte(self.ESNUMERO, -1, "0", 0, 0))

            self.Partes.append(Parte(self.ESPARCIERRA, -1, "0", 0, 0))
            pos = pos + 1

# Convierte número en cadena a real
def CadenaAReal(self, Numero):
    parteEntera = 0
    cont = 0

    for cont in range(0, len(Numero), 1):
        if Numero[cont] == '.':
            break
        parteEntera = (parteEntera * 10) + (ord(Numero[cont]) - ord('0'))

    parteDecimal = 0
    multiplica = 1

    for num in range(cont + 1, len(Numero), 1):
        parteDecimal = (parteDecimal * 10) + (ord(Numero[num]) - ord('0'))
        multiplica = multiplica * 10

    numeroB = parteEntera + parteDecimal / multiplica
    return numeroB

# Ahora convierte las partes en las piezas finales de ejecución
def CrearPiezas(self):
    cont = len(self.Partes)-1

    while True:
        tmpParte = self.Partes[cont];
        if tmpParte.Tipo == self.ESPARABRE or tmpParte.Tipo == self.ESFUNCION:
            self.GenerarPiezasOperador('^', '^', cont); # Evalúa las potencias
            self.GenerarPiezasOperador('*', '/', cont); # Luego evalúa multiplicar y dividir
            self.GenerarPiezasOperador('+', '-', cont); # Finalmente evalúa sumar y restar

            if tmpParte.Tipo == self.ESFUNCION: # Agrega la función a la última pieza
                self.Piezas[len(self.Piezas) - 1].Funcion = tmpParte.Funcion;

            # Quita el paréntesis/función que abre y el que cierra, dejando el centro
            self.Partes.pop(cont)
            self.Partes.pop(cont + 1)

        cont = cont - 1
        if cont < 0:
            break

def GenerarPiezasOperador(self, operA, operB, ini):
    cont = ini + 1

    while True:
        tmpParte = self.Partes[cont];
        if tmpParte.Tipo == self.ESOPERADOR and (tmpParte.Operador == operA or tmpParte.Operador ==
operB):

            tmpParteIzq = self.Partes[cont - 1];
            tmpParteDer = self.Partes[cont + 1];

            # Crea Pieza
            self.Piezas.append(Pieza(-1,
                                    tmpParteIzq.Tipo, tmpParteIzq.Numero,
                                    tmpParteIzq.UnaVariable, tmpParteIzq.Acumulador,
                                    tmpParte.Operador,
                                    tmpParteDer.Tipo, tmpParteDer.Numero,
                                    tmpParteDer.UnaVariable, tmpParteDer.Acumulador))

            # Elimina la parte del operador y la siguiente
            self.Partes.pop(cont)
            self.Partes.pop(cont)

            # Retorna el contador en uno para tomar la siguiente operación

```

```

        cont = cont - 1

        # Cambia la parte anterior por parte que acumula
        tmpParteIzq.Tipo = self.ESACUMULA
        tmpParteIzq.Acumulador = len(self.Piezas) - 1

    cont = cont + 1

    if self.Partes[cont].Tipo == self.ESPARCIERRA:
        break

# Evalúa la expresión convertida en piezas
def Evaluar(self):
    resultado = 0
    for pos in range(0, len(self.Piezas), 1):
        tmpPieza = self.Piezas[pos];

        if tmpPieza.TipoA == self.ESNUMERO:
            numA = tmpPieza.NumeroA
        elif tmpPieza.TipoA == self.ESVARIABLE:
            numA = self.VariableAlgebra[tmpPieza.VariableA]
        else:
            numA = self.Piezas[tmpPieza.PiezaA].ValorPieza

        if tmpPieza.TipoB == self.ESNUMERO:
            numB = tmpPieza.NumeroB
        elif tmpPieza.TipoB == self.ESVARIABLE:
            numB = self.VariableAlgebra[tmpPieza.VariableB]
        else:
            numB = self.Piezas[tmpPieza.PiezaB].ValorPieza

        if tmpPieza.Operador == '*':
            resultado = numA * numB
        elif tmpPieza.Operador == '/':
            try:
                resultado = numA / numB
            except ZeroDivisionError:
                return float('NaN')
        elif tmpPieza.Operador == '+':
            resultado = numA + numB
        elif tmpPieza.Operador == '-':
            resultado = numA - numB
        else:
            try:
                resultado = math.pow(numA, numB)
            except:
                return float('NaN')

        if tmpPieza.Funcion == 0:
            resultado = math.sin(resultado)
        elif tmpPieza.Funcion == 1:
            resultado = math.cos(resultado)
        elif tmpPieza.Funcion == 2:
            resultado = math.tan(resultado)
        elif tmpPieza.Funcion == 3:
            resultado = math.fabs(resultado)
        elif tmpPieza.Funcion == 4:
            try:
                resultado = math.asin(resultado)
            except:
                return float('NaN')
        elif tmpPieza.Funcion == 5:
            try:
                resultado = math.acos(resultado)
            except:
                return float('NaN')
        elif tmpPieza.Funcion == 6:
            resultado = math.atan(resultado)
        elif tmpPieza.Funcion == 7:
            try:
                resultado = math.log(resultado)
            except:
                return float('NaN')
        elif tmpPieza.Funcion == 8:
            resultado = math.ceil(resultado)
        elif tmpPieza.Funcion == 9:
            try:
                resultado = math.exp(resultado)
            except:

```

```

        return float('NaN')
    elif tmpPieza.Funcion == 10:
        try:
            resultado = math.sqrt(resultado)
        except:
            return float('NaN')
    elif tmpPieza.Funcion == 11:
        resultado = math.pow(resultado, 0.333333333333333333333333)

    if math.isnan(resultado) or math.isinf(resultado):
        return resultado;

    tmpPieza.ValorPieza = resultado;

    return resultado;

# Da valor a las variables que tendrá la expresión algebraica
def DarValorVariable(self, varAlgebra, valor):
    self.VariableAlgebra[ord(varAlgebra) - ord('a')] = valor;

```

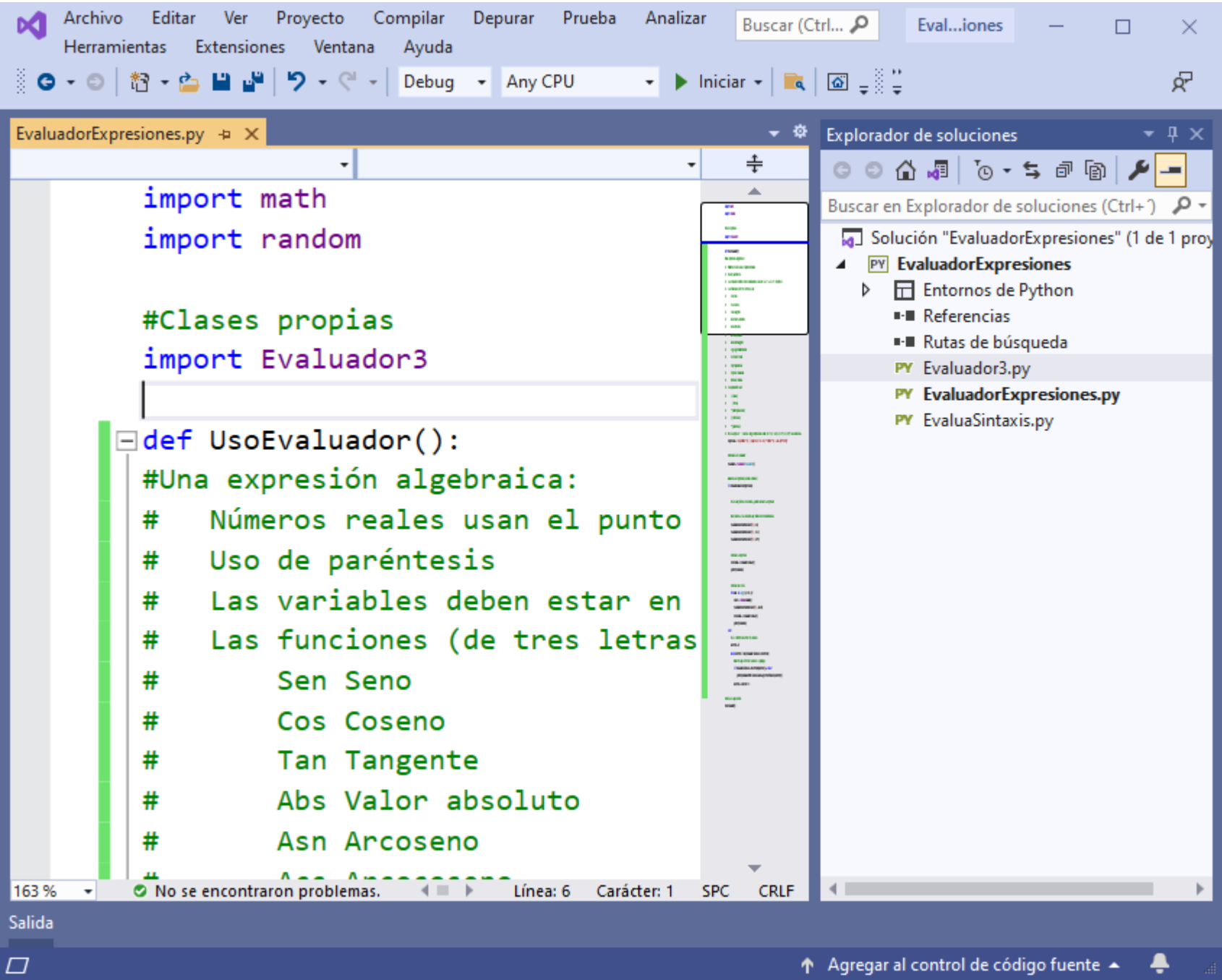


Ilustración 10: Proyecto en Visual Studio 2019

Program.py

```
import math
import random

#Clases propias
import Evaluador3

def UsoEvaluador()
#Una expresión algebraica:
#  Números reales usan el punto decimal
#  Uso de paréntesis
#  Las variables deben estar en minúsculas van de la 'a' a la 'z' excepto ñ
#  Las funciones (de tres letras) son:
#      Sen Seno
#      Cos Coseno
#      Tan Tangente
#      Abs Valor absoluto
#      Asn Arcoseno
#      Acs Arcocoseno
#      Atn Arcotangente
#      Log Logaritmo Natural
#      Cei Valor techo
#      Exp Exponencial
#      Sqr Raíz cuadrada
#      Rcb Raíz Cúbica
#  Los operadores son:
#      + (suma)
#      - (resta)
#      * (multiplicación)
#      / (división)
#      ^ (potencia)
#  No se acepta el "-" unario. Luego expresiones como: 4*-2 o (-5+3) o (-x^2) o (-x)^2 son inválidas.
expresion = "Cos(0.004 * x) - (Tan(1.78 / k + h) * SEN(k ^ x) + abs (k^3-h^2))";

#Instancia el evaluador
evaluador = new Evaluador3()
```



```

#Analiza la expresión (valida sintaxis)
if evaluador.Analizar(expresion)):

    #Si no hay fallos de sintaxis, puede evaluar la expresión

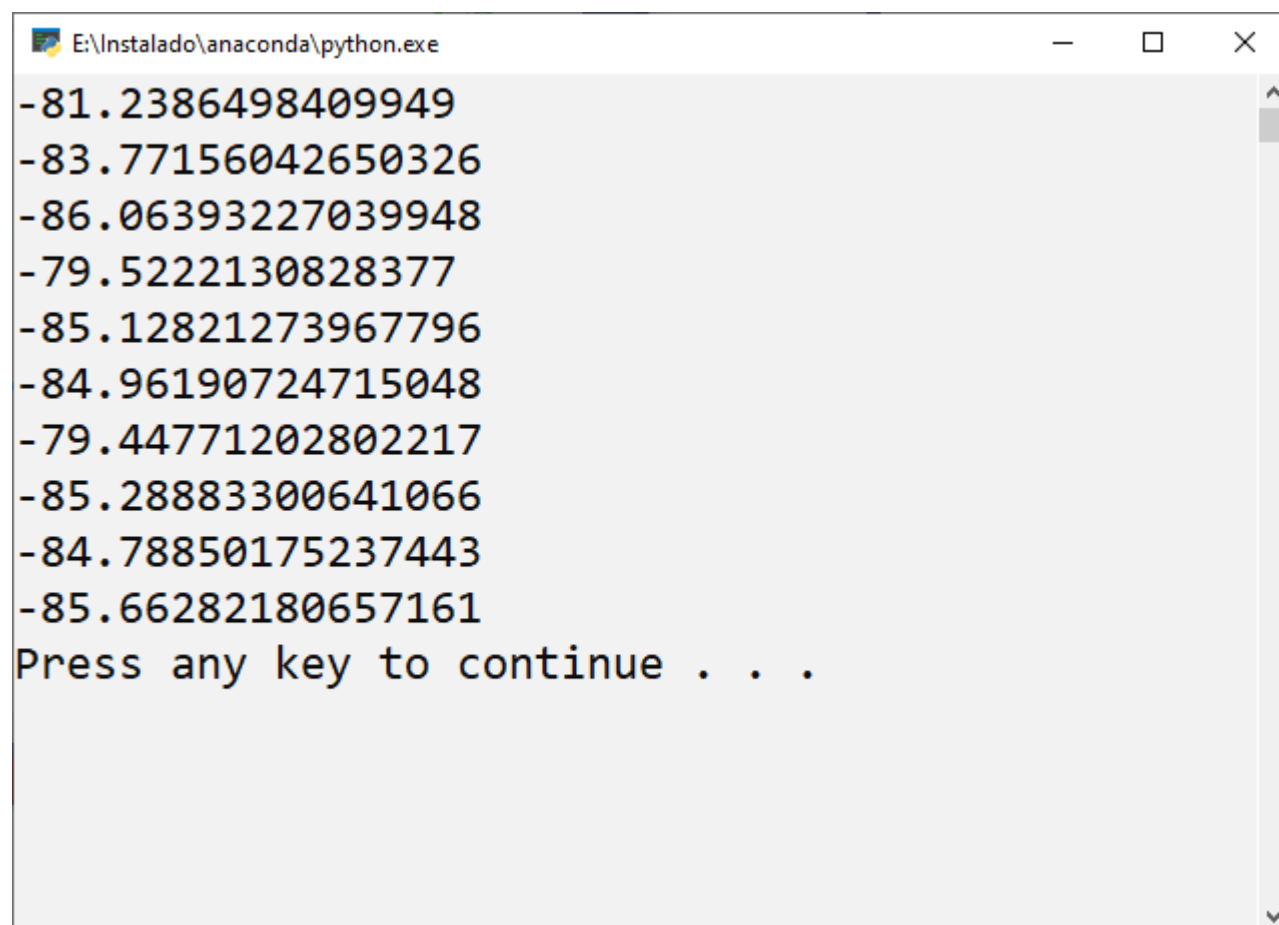
    #Da valores a las variables que deben estar en minúsculas
    evaluador.DarValorVariable('k', 1.6)
    evaluador.DarValorVariable('x', -8.3)
    evaluador.DarValorVariable('h', 9.29)

    #Evalúa la expresión
    resultado = evaluador.Evaluar()
    print($resultado)

    #Evalúa con ciclos
    for num in range(1, 10, 1):
        valor = random.random()
        evaluador.DarValorVariable('k', valor)
        resultado = $evaluador.Evaluar()
        print(resultado)
else:
    #Si se detectó un error de sintaxis
    unError = 0
    while unError < len(evaluador.Sintaxis.EsCorrecto):
        #Muestra que error de sintaxis se produjo
        if evaluador.Sintaxis.EsCorrecto[unError] == False:
            print(evaluador.Sintaxis.MensajesErrorSintaxis(unError))
            unError = unError + 1

#Inicia la aplicación
UsoEvaluador()

```



```

E:\Instalado\anaconda\python.exe
-81.2386498409949
-83.77156042650326
-86.06393227039948
-79.5222130828377
-85.12821273967796
-84.96190724715048
-79.44771202802217
-85.28883300641066
-84.78850175237443
-85.66282180657161
Press any key to continue . . .

```

Ilustración 11: Ejecución del programa en Python

TypeScript

```
class EvaluaSintaxis {
  _mensajeError: Array<string>;
  EsCorrecto: Array<boolean>;

  constructor(){
    /* Mensajes de error de sintaxis */
    this._mensajeError = [
      '0. Caracteres no permitidos. Ejemplo: 3$5+2',
      '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)*',
    ];

    this.EsCorrecto = new Array();
  }

  /* Retorna si el caracter es un operador matemático */
  EsUnOperador(car: string): boolean {
    return car === '+' || car === '-' || car === '*' || car === '/' || car === '^';
  }

  /* Retorna si el caracter es un número */
  EsUnNumero(car: string): boolean {
    return car >= '0' && car <= '9';
  }

  /* Retorna si el caracter es una letra */
  EsUnaLetra(car: string): boolean {
    return car >= 'a' && car <= 'z';
  }

  /* 0. Detecta si hay un caracter no válido */
  BuenaSintaxis00(expresion: string): boolean {
    var Resultado: boolean = true;
    var permitidos: string = 'abcdefghijklmnopqrstuvwxyz0123456789.+*/^()';
    for (var pos: number = 0; pos < expresion.length && Resultado; pos++) {
      if (permitidos.indexOf(expresion[pos] || '') === -1)
        Resultado = false;
    }
    return Resultado;
  }

  /* 1. Un número seguido de una letra. Ejemplo: 2q-(*3) */
  BuenaSintaxis01(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
      var carA: string = expresion[pos]!;
      var carB: string = expresion[pos + 1]!;
      if (this.EsUnNumero(carA) && this.EsUnaLetra(carB)) Resultado = false;
    }
  }
}
```

```

    return Resultado;
}

/* 2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6) */
BuenaSintaxis02(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (this.EsUnNumero(carA) && carB === '(') Resultado = false;
    }
    return Resultado;
}

/* 3. Doble punto seguido. Ejemplo: 3..1 */
BuenaSintaxis03(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === '.' && carB === '.') Resultado = false;
    }
    return Resultado;
}

/* 4. Punto seguido de operador. Ejemplo: 3.*1 */
BuenaSintaxis04(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === '.' && this.EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 5. Un punto y sigue una letra. Ejemplo: 3+5.w-8 */
BuenaSintaxis05(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === '.' && this.EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3 */
BuenaSintaxis06(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === '.' && carB === '(') Resultado = false;
    }
    return Resultado;
}

/* 7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3 */
BuenaSintaxis07(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === '.' && carB === ')') Resultado = false;
    }
    return Resultado;
}

/* 8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7 */
BuenaSintaxis08(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (this.EsUnOperador(carA) && carB === '.') Resultado = false;
    }
    return Resultado;
}

```

```

/* 9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3 */
BuenaSintaxis09(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (this.EsUnOperador(carA) && this.EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7 */
BuenaSintaxis10(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (this.EsUnOperador(carA) && carB === ')') Resultado = false;
    }
    return Resultado;
}

/* 11. Una letra seguida de número. Ejemplo: 7-2a-6 */
BuenaSintaxis11(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (this.EsUnaLetra(carA) && this.EsUnNumero(carB)) Resultado = false;
    }
    return Resultado;
}

/* 12. Una letra seguida de punto. Ejemplo: 7-a.-6 */
BuenaSintaxis12(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (this.EsUnaLetra(carA) && carB === '.') Resultado = false;
    }
    return Resultado;
}

/* 13. Un paréntesis que abre seguido de punto. Ejemplo: 7-(.4-6) */
BuenaSintaxis13(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === '(' && carB === '.') Resultado = false;
    }
    return Resultado;
}

/* 14. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3) */
BuenaSintaxis14(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === '(' && this.EsUnOperador(carB)) Resultado = false;
    }
    return Resultado;
}

/* 15. Un paréntesis que abre seguido de un paréntesis que cierra. Ejemplo: 7-()-6 */
BuenaSintaxis15(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === '(' && carB === ')') Resultado = false;
    }
    return Resultado;
}

/* 16. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7 */

```

```

BuenaSintaxis16(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === ')') && this.EsUnNumero(carB)) Resultado = false;
    }
    return Resultado;
}

/* 17. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5). */
BuenaSintaxis17(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === ')') && carB === '.') Resultado = false;
    }
    return Resultado;
}

/* 18. Un paréntesis que cierra y sigue una letra. Ejemplo: (3-5)t */
BuenaSintaxis18(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === ')') && this.EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 19. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: (3-5)(4*5) */
BuenaSintaxis19(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (carA === ')') && carB === '(') Resultado = false;
    }
    return Resultado;
}

/* 20. Si hay dos letras seguidas (después de quitar las funciones), es un error */
BuenaSintaxis20(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (this.EsUnaLetra(carA) && this.EsUnaLetra(carB)) Resultado = false;
    }
    return Resultado;
}

/* 21. Los paréntesis estén desbalanceados. Ejemplo: 3-(2*4)) */
BuenaSintaxis21(expresion: string): boolean {
    var parabre: number = 0; /* Contador de paréntesis que abre */
    var parcierra: number = 0; /* Contador de paréntesis que cierra */
    for (var pos: number = 0; pos < expresion.length; pos++) {
        var carA: string = expresion[pos]!;
        if (carA === '(') parabre++;
        if (carA === ')') parcierra++;
    }
    return parcierra == parabre;
}

/* 22. Doble punto en un número de tipo real. Ejemplo: 7-6.46.1+2 */
BuenaSintaxis22(expresion: string): boolean {
    var Resultado: boolean = true;
    var totalpuntos: number = 0; /* Validar los puntos decimales de un número real */
    for (var pos: number = 0; pos < expresion.length && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        if (this.EsUnOperador(carA)) totalpuntos = 0;
        if (carA === '.') totalpuntos++;
        if (totalpuntos > 1) Resultado = false;
    }
    return Resultado;
}

```



```

/* 23. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4'; */
BuenaSintaxis23(expresion: string): boolean {
    var Resultado: boolean = true;
    var parabre: number = 0; /* Contador de paréntesis que abre */
    var parcierra: number = 0; /* Contador de paréntesis que cierra */
    for (var pos: number = 0; pos < expresion.length && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        if (carA === '(') parabre++;
        if (carA === ')') parcierra++;
        if (parcierra > parabre) Resultado = false;
    }
    return Resultado;
}

/* 24. Inicia con operador. Ejemplo: +3*5 */
BuenaSintaxis24(expresion: string): boolean {
    return !this.EsUnOperador(expresion[0]!);
}

/* 25. Finaliza con operador. Ejemplo: 3*5* */
BuenaSintaxis25(expresion: string): boolean {
    return !this.EsUnOperador(expresion[expresion.length - 1]!);
}

/* 26. Encuentra una letra seguida de paréntesis que abre. Ejemplo: 3-a(7)-5 */
BuenaSintaxis26(expresion: string): boolean {
    var Resultado: boolean = true;
    for (var pos: number = 0; pos < expresion.length - 1 && Resultado; pos++) {
        var carA: string = expresion[pos]!;
        var carB: string = expresion[pos + 1]!;
        if (this.EsUnaLetra(carA) && carB === '(') Resultado = false;
    }
    return Resultado;
}

SintaxisCorrecta(ecuacion: string): boolean {
    /* Reemplaza las funciones de tres letras por una letra */
    var expresion: string = ecuacion.replace(/sen\(/gi, "a+(").replace(/cos\(/gi, "a+(").replace(/tan\(/gi,
"a+(").replace(/abs\(/gi, "a+(").replace(/asn\(/gi, "a+(").replace(/acs\(/gi, "a+(").replace(/atn\(/gi,
"a+(").replace(/log\(/gi, "a+(").replace(/cei\(/gi, "a+(").replace(/exp\(/gi, "a+(").replace(/sqr\(/gi,
"a+(").replace(/rcb\(/gi, "a+(");

    /* Hace las pruebas de sintaxis */
    this.EsCorrecto[0] = this.BuenaSintaxis00(expresion);
    this.EsCorrecto[1] = this.BuenaSintaxis01(expresion);
    this.EsCorrecto[2] = this.BuenaSintaxis02(expresion);
    this.EsCorrecto[3] = this.BuenaSintaxis03(expresion);
    this.EsCorrecto[4] = this.BuenaSintaxis04(expresion);
    this.EsCorrecto[5] = this.BuenaSintaxis05(expresion);
    this.EsCorrecto[6] = this.BuenaSintaxis06(expresion);
    this.EsCorrecto[7] = this.BuenaSintaxis07(expresion);
    this.EsCorrecto[8] = this.BuenaSintaxis08(expresion);
    this.EsCorrecto[9] = this.BuenaSintaxis09(expresion);
    this.EsCorrecto[10] = this.BuenaSintaxis10(expresion);
    this.EsCorrecto[11] = this.BuenaSintaxis11(expresion);
    this.EsCorrecto[12] = this.BuenaSintaxis12(expresion);
    this.EsCorrecto[13] = this.BuenaSintaxis13(expresion);
    this.EsCorrecto[14] = this.BuenaSintaxis14(expresion);
    this.EsCorrecto[15] = this.BuenaSintaxis15(expresion);
    this.EsCorrecto[16] = this.BuenaSintaxis16(expresion);
    this.EsCorrecto[17] = this.BuenaSintaxis17(expresion);
    this.EsCorrecto[18] = this.BuenaSintaxis18(expresion);
    this.EsCorrecto[19] = this.BuenaSintaxis19(expresion);
    this.EsCorrecto[20] = this.BuenaSintaxis20(expresion);
    this.EsCorrecto[21] = this.BuenaSintaxis21(expresion);
    this.EsCorrecto[22] = this.BuenaSintaxis22(expresion);
    this.EsCorrecto[23] = this.BuenaSintaxis23(expresion);
    this.EsCorrecto[24] = this.BuenaSintaxis24(expresion);
    this.EsCorrecto[25] = this.BuenaSintaxis25(expresion);
    this.EsCorrecto[26] = this.BuenaSintaxis26(expresion);

    var Resultado: boolean = true;
    for (var cont: number = 0; cont < this.EsCorrecto.length && Resultado; cont++)
        if (this.EsCorrecto[cont] === false) Resultado = false;
    return Resultado;
}

/* Transforma la expresión para ser chequeada y analizada */
Transforma(expresion: string): string {

```



```

/* Quita espacios, tabuladores y la vuelve a minúsculas */
var nuevo: string = "";
for (var num: number = 0; num < expresion.length; num++) {
    var letra: string = expresion[num];
    if (letra >= 'A' && letra <= 'Z') String.fromCharCode(letra.charCodeAt(0) + ' '.charCodeAt(0));
    if (letra != ' ' && letra != '\n') nuevo += letra;
}

/* Cambia los )) por )+0) porque es requerido al crear las piezas */
var Nuevo: string = nuevo.replace(/\\)\)/gi, ") + 0)");

return Nuevo;
}

/* Muestra mensaje de error sintáctico */
MensajesErrorSintaxis(CodigoError: number): string {
    return this._mensajeError[CodigoError]!;
}
}

```

```
class Parte {
  Tipo: number;
  Funcion: number;
  Operador: string;
  Numero: number;
  UnaVariable: number;
  Acumulador: number;

  constructor(tipo: number, funcion: number, operador: string, numero: number, unaVariable: number) {
    this.Tipo = tipo; /* Acumulador, función, paréntesis que abre, paréntesis que cierra, operador, número,
variable */
    this.Funcion = funcion; /* 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 */
    this.Operador = operador; /* + suma - resta * multiplicación / división ^ potencia */
    this.Numero = numero; /* Número literal, por ejemplo: 3.141592 */
    this.UnaVariable = unaVariable; /* Variable algebraica */
    this.Acumulador = 0; /* Usado cuando la expresión se convierte en piezas. Por ejemplo:
      3 + 2 / 5 se convierte así:
      |3| |+| |2| | / | |5|
      |3| |+| |A| A es un identificador de acumulador */
  }
}
```

```
class Pieza {
  ValorPieza: number; /* Almacena el valor que genera la pieza al evaluarse */
  Funcion: number; /* 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 */
  TipoA: number; /* La primera parte es un número o una variable o trae el valor de otra pieza */
  NumeroA: number; /* Es un número literal */
  VariableA: number; /* Es una variable */
  PiezaA: number; /* Trae el valor de otra pieza */
  Operador: string; /* + suma - resta * multiplicación / división ^ potencia */
  TipoB: number; /* La segunda parte es un número o una variable o trae el valor de otra pieza */
  NumeroB: number; /* Es un número literal */
  VariableB: number; /* Es una variable */
  PiezaB: number; /* Trae el valor de otra pieza */

  constructor(funcion:number, tipoA: number, numeroA: number, variableA: number, piezaA: number, operador:
string, tipoB: number, numeroB: number, variableB: number, piezaB: number) {
    this.ValorPieza = 0;

    this.Funcion = funcion;

    this.TipoA = tipoA;
    this.NumeroA = numeroA;
    this.VariableA = variableA;
    this.PiezaA = piezaA;

    this.Operador = operador;

    this.TipoB = tipoB;
    this.NumeroB = numeroB;
    this.VariableB = variableB;
    this.PiezaB = piezaB;
  }
}
```

```
class Evaluador3 {
  /* Constantes de los diferentes tipos de datos que tendrán las piezas */
  ESFUNCION: number;
  ESPARABRE: number;
  ESPARCIERRA: number;
  ESOPERADOR: number;
  ESNUMERO: number;
  ESVARIABLE: number;
  ESACUMULA: number;

  /* Listado de partes en que se divide la expresión
    Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
    |3.14| |+| |sen(| |4| | / | |x| |)| |*| |( | |7.2| | ^ | |3| |-| |1| |)|
    Cada parte puede tener un número, un operador, una función, un paréntesis que abre o un paréntesis
    que cierra */
  Partes: Array<Parte>;

  /* Listado de piezas que ejecutan
    Toma las partes y las divide en piezas con la siguiente estructura:
    acumula = funcion numero/variable/acumula operador numero/variable/acumula
    Siguiendo el ejemplo anterior sería:
    A = 7.2^3
    B = A - 1
    C = seno ( 4 / x )
    D = C * B
    E = 3.14 + D

    Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación */
  Piezas: Array<Pieza>;

  /* El arreglo unidimensional que lleva el valor de las variables */
  VariableAlgebra: Array<number>;

  /* Uso del chequeo de sintaxis */
  Sintaxis: EvaluaSintaxis;

  constructor() {
    /* Constantes de los diferentes tipos de datos que tendrán las piezas */
    this.ESFUNCION = 1;
    this.ESPARABRE = 2;
    this.ESPARCIERRA = 3;
    this.ESOPERADOR = 4;
    this.ESNUMERO = 5;
    this.ESVARIABLE = 6;
    this.ESACUMULA = 7;

    /* Inicializa la lista de partes */
    this.Partes = new Array();

    /* Inicializa la lista de piezas */
    this.Piezas = new Array();

    /* El arreglo unidimensional que lleva el valor de las variables */
    this.VariableAlgebra = new Array();

    /* Uso del chequeo de sintaxis */
    this.Sintaxis = new EvaluaSintaxis();
  }

  /* Analiza la expresión */
  Analizar(expresionA: string): boolean {
    var expresionB: string = this.Sintaxis.Transforma(expresionA);
    var chequeo: boolean = this.Sintaxis.SintaxisCorrecta(expresionB);
    if (chequeo === true) {
      this.Partes.length = 0;
      this.Piezas.length = 0;
      this.CrearPartes(expresionB);
      this.CrearPiezas();
    }
    return chequeo;
  }

  /* Divide la expresión en partes */
  CrearPartes(expresion: string): void {
    /* Debe analizarse con paréntesis */
    var NuevoA: string = "(" + expresion + ")";
```

```

/* Reemplaza las funciones de tres letras por una letra mayúscula */
var NuevoB: string = NuevoA.replace(/sen/gi, "A").replace(/cos/gi, "B").replace(/tan/gi,
"C").replace(/abs/gi, "D").replace(/asn/gi, "E").replace(/acs/gi, "F").replace(/atn/gi,
"G").replace(/log/gi, "H").replace(/cei/gi, "I").replace(/exp/gi, "J").replace(/sqr/gi,
"K").replace(/rcb/gi, "L");

/* Va de caracter en caracter */
var Numero: string = "";
for (var pos: number = 0; pos < NuevoB.length; pos++) {
    var car: string = NuevoB[pos]!;
    /* Si es un número lo va acumulando en una cadena */
    if ((car >= '0' && car <= '9') || car == '.') {
        Numero += car;
    }
    /* Si es un operador entonces agrega número (si existía) */
    else if (car == '+' || car == '-' || car == '*' || car == '/' || car == '^') {
        if (Numero.length > 0) {
            this.Partes.push(new Parte(this.ESNUMERO, -1, '0', this.CadenaAReal(Numero), 0));
            Numero = "";
        }
        this.Partes.push(new Parte(this.ESOPERADOR, -1, car, 0, 0));
    }
    /* Si es variable */
    else if (car >= 'a' && car <= 'z') {
        this.Partes.push(new Parte(this.ESVARIABLE, -1, '0', 0, car.charCodeAt(0) - 'a'.charCodeAt(0)));
    }
    /* Si es una función (seno, coseno, tangente, ...) */
    else if (car >= 'A' && car <= 'L') {
        this.Partes.push(new Parte(this.ESFUNCION, car.charCodeAt(0) - 'A'.charCodeAt(0), '0', 0, 0));
        pos++;
    }
    /* Si es un paréntesis que abre */
    else if (car == '(') {
        this.Partes.push(new Parte(this.ESPARABRE, -1, '0', 0, 0));
    }
    /* Si es un paréntesis que cierra */
    else {
        if (Numero.length > 0) {
            this.Partes.push(new Parte(this.ESNUMERO, -1, '0', this.CadenaAReal(Numero), 0));
            Numero = "";
        }
        /* Si sólo había un número o variable dentro del paréntesis le agrega + 0 (por ejemplo: sen(x) o
3*(2) ) */
        if (this.Partes[this.Partes.length - 2]!.Tipo == this.ESPARABRE || this.Partes[this.Partes.length -
2]!.Tipo == this.ESFUNCION) {
            this.Partes.push(new Parte(this.ESOPERADOR, -1, '+', 0, 0));
            this.Partes.push(new Parte(this.ESNUMERO, -1, '0', 0, 0));
        }

        this.Partes.push(new Parte(this.ESPARCIERRA, -1, '0', 0, 0));
    }
}
}

/* Convierte un número almacenado en una cadena a su valor real */
CadenaAReal(Numero: string): number {
    //Parte entera
    var parteEntera: number = 0;
    var cont: number = 0;
    for (cont = 0; cont < Numero.length; cont++) {
        if (Numero[cont]!.charCodeAt(0) === '.'.charCodeAt(0)) break;
        parteEntera = parteEntera * 10 + (Numero[cont]!.charCodeAt(0) - '0'.charCodeAt(0));
    }

    //Parte decimal
    var parteDecimal: number = 0;
    var multiplica: number = 1;
    for (var num: number = cont + 1; num < Numero.length; num++) {
        parteDecimal = parteDecimal * 10 + (Numero[num]!.charCodeAt(0) - '0'.charCodeAt(0));
        multiplica *= 10;
    }

    var numero: number = parteEntera + parteDecimal / multiplica;
    return numero;
}

/* Ahora convierte las partes en las piezas finales de ejecución */
CrearPiezas(): void {

```

```

var cont: number = this.Partes.length - 1;
do {
    var tmpParte: Parte = this.Partes[cont]!;
    if (tmpParte.Tipo === this.ESPARABRE || tmpParte.Tipo === this.ESFUNCION) {
        this.GenerarPiezasOperador('^', '^', cont); /* Evalúa las potencias */
        this.GenerarPiezasOperador('*', '/', cont); /* Luego evalúa multiplicar y dividir */
        this.GenerarPiezasOperador('+', '-', cont); /* Finalmente evalúa sumar y restar */

        if (tmpParte.Tipo === this.ESFUNCION) { /* Agrega la función a la última pieza */
            this.Piezas[this.Piezas.length - 1]!.Funcion = tmpParte.Funcion;
        }

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

GenerarPiezasOperador(operA: string, operB: string, ini: number): void {
    var cont: number = ini + 1;
    do {
        var tmpParte: Parte = this.Partes[cont]!;
        if (tmpParte.Tipo === this.ESOPERADOR && (tmpParte.Operador === operA || tmpParte.Operador === operB)) {

            var tmpParteIzq: Parte = this.Partes[cont - 1]!;
            var tmpParteDer: Parte = this.Partes[cont + 1]!;
            /* Crea Pieza */
            this.Piezas.push(new Pieza(-1,
                tmpParteIzq.Tipo, tmpParteIzq.Numero,
                tmpParteIzq.UnaVariable, tmpParteIzq.Acumulador,
                tmpParte.Operador,
                tmpParteDer.Tipo, tmpParteDer.Numero,
                tmpParteDer.UnaVariable, tmpParteDer.Acumulador));

            /* Elimina la parte del operador y la siguiente */
            this.Partes.splice(cont, 1);
            this.Partes.splice(cont, 1);

            /* Retorna el contador en uno para tomar la siguiente operación */
            cont -= 1;

            /* Cambia la parte anterior por parte que acumula */
            tmpParteIzq.Tipo = this.ESACUMULA;
            tmpParteIzq.Acumulador = this.Piezas.length - 1;
        }
        cont++;
    } while (this.Partes[cont]!.Tipo !== this.ESPARCIERRA);
}

/* Evalúa la expresión convertida en piezas */
Evaluar(): number {
    var numA: number, numB: number, resultado: number = 0;
    for (var pos: number = 0; pos < this.Piezas.length; pos++) {
        var tmpPieza: Pieza = this.Piezas[pos]!;

        switch (tmpPieza.TipoA) {
            case this.ESNUMERO: numA = tmpPieza.NumeroA; break;
            case this.ESVARIABLE: numA = this.VariableAlgebra[tmpPieza.VariableA]!; break;
            default: numA = this.Piezas[tmpPieza.PiezaA]!.ValorPieza; break;
        }

        switch (tmpPieza.TipoB) {
            case this.ESNUMERO: numB = tmpPieza.NumeroB; break;
            case this.ESVARIABLE: numB = this.VariableAlgebra[tmpPieza.VariableB]!; break;
            default: numB = this.Piezas[tmpPieza.PiezaB]!.ValorPieza; break;
        }

        switch (tmpPieza.Operador) {
            case '*': resultado = numA * numB; break;
            case '/': resultado = numA / numB; break;
            case '+': resultado = numA + numB; break;
            case '-': resultado = numA - numB; break;
            default: resultado = Math.pow(numA, numB); break;
        }
    }
    if (isNaN(resultado) || !isFinite(resultado)) return resultado;

    switch (tmpPieza.Funcion) {

```

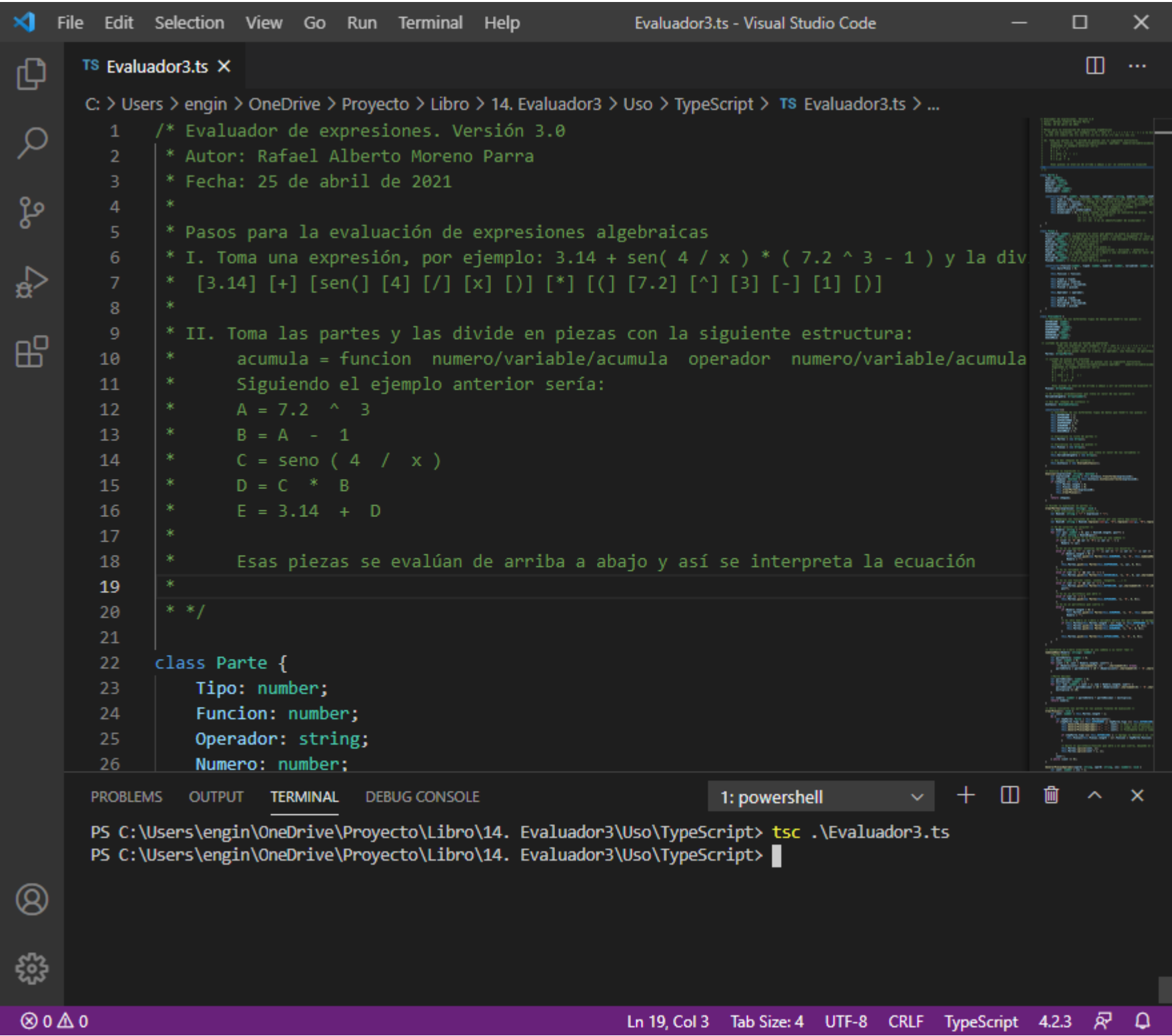



Ilustración 12: Usando Visual Studio Code como editor.

Inicio.html

```

<!DOCTYPE HTML><html><body>
<script type="text/javascript" src="Evaluador3.js"> </script>
<script type="text/javascript">
UsoEvaluador();

function UsoEvaluador() {
    /* Una expresión algebraica:
    Números reales usan el punto decimal
    Uso de paréntesis
    Las variables deben estar en minúsculas van de la 'a' a la 'z' excepto ñ
    Las funciones (de tres letras) son:
    Sen  Seno
    Cos  Coseno
    Tan  Tangente
    Abs  Valor absoluto
    Asn  Arcoseno
    Acs  Arcocoseno
    Atn  Arcotangente
    Log  Logaritmo Natural
    Cei  Valor techo
    Exp  Exponencial
    Sqr  Raíz cuadrada
    Rcb  Raíz Cúbica
    Los operadores son:
    + (suma)
    - (resta)
    * (multiplicación)
    / (división)
    ^ (potencia)

```



```

    No se acepta el "-" unario. Luego expresiones como: 4*-2 o (-5+3) o (-x^2) o (-x)^2 son inválidas.
*/
var expresion = "Cos(0.004 * x) - (Tan(1.78 / k + h) * SEN(k ^ x) + abs (k^3-h^2))";

//Instancia el evaluador
var evaluador = new Evaluador3();

//Analiza la expresión (valida sintaxis)
if (evaluador.Analizar(expresion)===true){
    //Si no hay fallos de sintaxis, puede evaluar la expresión

    //Da valores a las variables que deben estar en minúsculas
    evaluador.DarValorVariable('k', 1.6);
    evaluador.DarValorVariable('x', -8.3);
    evaluador.DarValorVariable('h', 9.29);

    //Evalúa la expresión
    var resultado = evaluador.Evaluar();
    document.write(resultado + "<br>");

    //Evalúa con ciclos
    for (var num = 1; num <= 10; num++) {
        var valor = Math.random();
        evaluador.DarValorVariable('k', valor);
        resultado = evaluador.Evaluar();
        document.write(resultado + "<br>");
    }
}
else {
    for(var unError = 0; unError < evaluador.Sintaxis.EsCorrecto.length; unError++){
        if (evaluador.Sintaxis.EsCorrecto[unError] === false) {
            document.write(evaluador.Sintaxis.MensajesErrorSintaxis(unError)+"<br>");
        }
    }
}
}
</script>
</body></html>
```

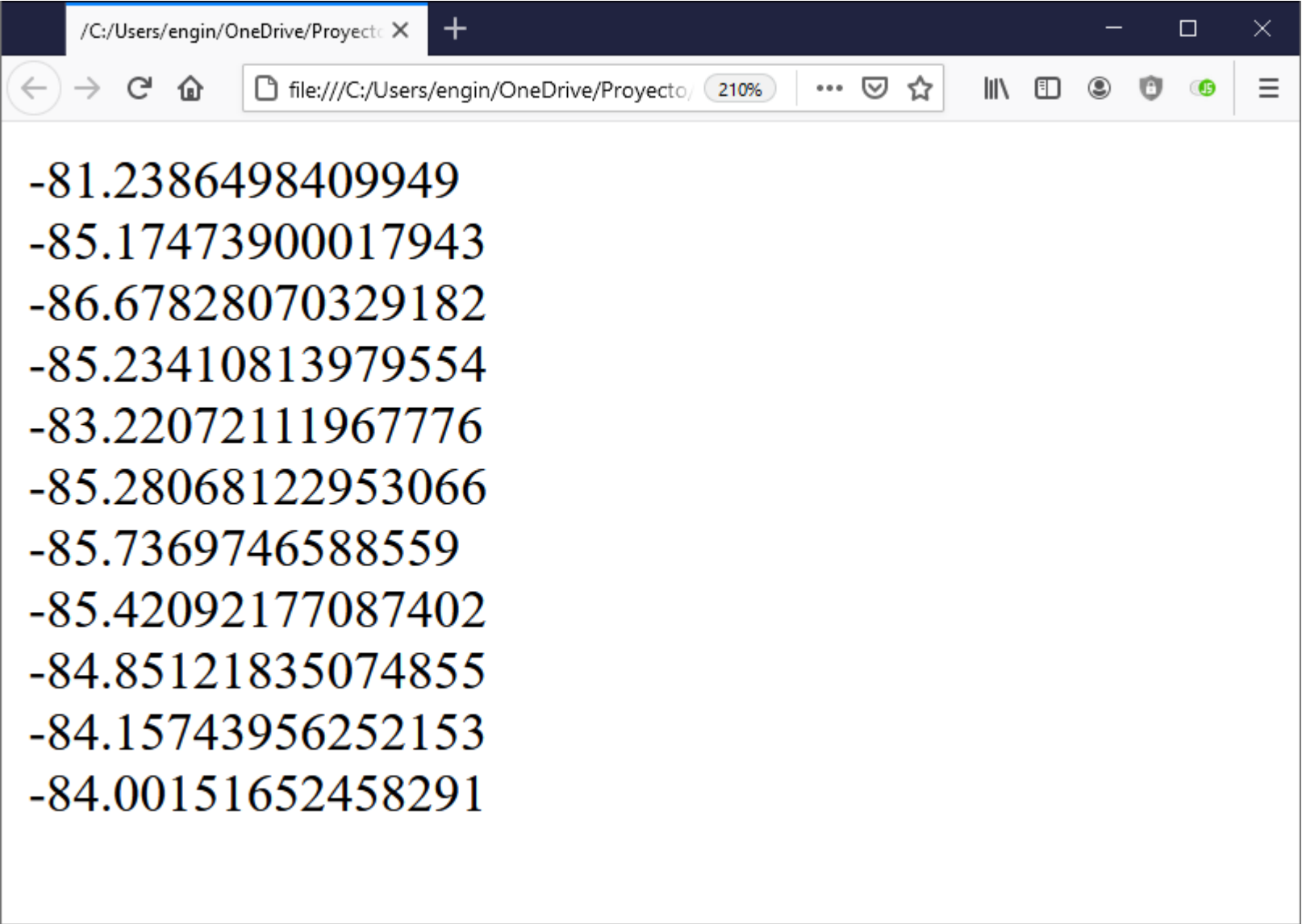


Ilustración 13: Ejecución dentro del navegador

Visual Basic .NET

```
Public Class EvaluaSintaxis
    'Mensajes de error de sintaxis
    Private ReadOnly _mensajeError As String() = {
        "0. Caracteres no permitidos. Ejemplo: 3$5+2",
        "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)*"}

    Public EsCorrecto As Boolean() = New Boolean(26) {}

    'Retorna si el caracter es un operador matemático
    Private Function EsUnOperador(ByVal car As Char) As Boolean
        Return car = "+"c OrElse car = "-"c OrElse car = "*"c OrElse car = "/"c OrElse car = "^"c
    End Function

    'Retorna si el caracter es un número
    Private Function EsUnNumero(ByVal car As Char) As Boolean
        Return car >= "0"c AndAlso car <= "9"c
    End Function

    'Retorna si el caracter es una letra
    Private Function EsUnaLetra(ByVal car As Char) As Boolean
        Return car >= "a"c AndAlso car <= "z"c
    End Function

    '0. Detecta si hay un caracter no válido
    Private Function BuenaSintaxis00(ByVal expresion As String) As Boolean
        Dim Resultado As Boolean = True
        Const permitidos As String = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^() "
        Dim pos As Integer = 0

        While pos < expresion.Length AndAlso Resultado
            If permitidos.IndexOf(expresion(pos)) = -1 Then Resultado = False
            pos += 1
        End While

        Return Resultado
    End Function

    '1. Un número seguido de una letra. Ejemplo: 2q-(*3)
    Private Function BuenaSintaxis01(ByVal expresion As String) As Boolean
        Dim Resultado As Boolean = True
        Dim pos As Integer = 0

        While pos < expresion.Length - 1 AndAlso Resultado
            Dim carA As Char = expresion(pos)
            Dim carB As Char = expresion(pos + 1)
            If EsUnNumero(carA) AndAlso EsUnaLetra(carB) Then Resultado = False
            pos += 1
        End While
    End Function
End Class
```

```

Return Resultado
End Function

'2. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
Private Function BuenaSintaxis02(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If EsUnNumero(carA) AndAlso carB = "(" Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'3. Doble punto seguido. Ejemplo: 3..1
Private Function BuenaSintaxis03(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If carA = "."c AndAlso carB = "."c Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'4. Punto seguido de operador. Ejemplo: 3.*1
Private Function BuenaSintaxis04(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If carA = "."c AndAlso EsUnOperador(carB) Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'5. Un punto y sigue una letra. Ejemplo: 3+5.w-8
Private Function BuenaSintaxis05(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If carA = "."c AndAlso EsUnaLetra(carB) Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'6. Punto seguido de paréntesis que abre. Ejemplo: 2-5.(4+1)*3
Private Function BuenaSintaxis06(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If carA = "."c AndAlso carB = "("c Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

```

```

'7. Punto seguido de paréntesis que cierra. Ejemplo: 2-(5.)*3
Private Function BuenaSintaxis07(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If carA = "."c AndAlso carB = ")"c Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'8. Un operador seguido de un punto. Ejemplo: 2-(4+.1)-7
Private Function BuenaSintaxis08(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If EsUnOperador(carA) AndAlso carB = "."c Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'9. Dos operadores estén seguidos. Ejemplo: 2++4, 5-*3
Private Function BuenaSintaxis09(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If EsUnOperador(carA) AndAlso EsUnOperador(carB) Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'10. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
Private Function BuenaSintaxis10(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If EsUnOperador(carA) AndAlso carB = ")"c Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'11. Una letra seguida de número. Ejemplo: 7-2a-6
Private Function BuenaSintaxis11(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If EsUnaLetra(carA) AndAlso EsUnNumero(carB) Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'12. Una letra seguida de punto. Ejemplo: 7-a.-6 */
Private Function BuenaSintaxis12(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

```

```

While pos < expresion.Length - 1 AndAlso Resultado
    Dim carA As Char = expresion(pos)
    Dim carB As Char = expresion(pos + 1)
    If EsUnaLetra(carA) AndAlso carB = "."c Then Resultado = False
    pos += 1
End While

Return Resultado
End Function

```

```

'13. Un paréntesis que abre seguido de punto. Ejemplo: 7-(.4-6)
Private Function BuenaSintaxis13(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If carA = "("c AndAlso carB = "."c Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

```

```

'14. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
Private Function BuenaSintaxis14(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If carA = "("c AndAlso EsUnOperador(carB) Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

```

```

'15. Un paréntesis que abre seguido de un paréntesis que cierra. Ejemplo: 7-()-6
Private Function BuenaSintaxis15(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If carA = "("c AndAlso carB = ")"c Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

```

```

'16. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7
Private Function BuenaSintaxis16(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If carA = ")"c AndAlso EsUnNumero(carB) Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

```

```

'17. Un paréntesis que cierra y sigue un punto. Ejemplo: (3-5).
Private Function BuenaSintaxis17(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)

```



```

    If carA = ")"c AndAlso carB = "."c Then Resultado = False
    pos += 1
End While

Return Resultado
End Function

'18. Un paréntesis que cierra y sigue una letra. Ejemplo: (3-5)t
Private Function BuenaSintaxis18(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If carA = ")"c AndAlso EsUnaLetra(carB) Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'19. Un paréntesis que cierra y sigue un paréntesis que abre. Ejemplo: (3-5)(4*5)
Private Function BuenaSintaxis19(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If carA = ")"c AndAlso carB = "("c Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'20. Si hay dos letras seguidas (después de quitar las funciones), es un error
Private Function BuenaSintaxis20(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim pos As Integer = 0

    While pos < expresion.Length - 1 AndAlso Resultado
        Dim carA As Char = expresion(pos)
        Dim carB As Char = expresion(pos + 1)
        If EsUnaLetra(carA) AndAlso EsUnaLetra(carB) Then Resultado = False
        pos += 1
    End While

    Return Resultado
End Function

'21. Los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
Private Function BuenaSintaxis21(ByVal expresion As String) As Boolean
    Dim parabre As Integer = 0 'Contador de paréntesis que abre
    Dim parcierra As Integer = 0 'Contador de paréntesis que cierra

    For pos As Integer = 0 To expresion.Length - 1

        Select Case expresion(pos)
            Case "("c
                parabre += 1
            Case ")"c
                parcierra += 1
        End Select
    Next

    Return parcierra = parabre
End Function

'22. Doble punto en un número de tipo real. Ejemplo: 7-6.46.1+2
Private Function BuenaSintaxis22(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim totalpuntos As Integer = 0 'Validar los puntos decimales de un número real
    Dim pos As Integer = 0

    While pos < expresion.Length AndAlso Resultado
        Dim carA As Char = expresion(pos)
        If EsUnOperador(carA) Then totalpuntos = 0
    End While

```

```

    If carA = "."c Then totalpuntos += 1
    If totalpuntos > 1 Then Resultado = False
    pos += 1
End While

Return Resultado
End Function

```

'23. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4

```

Private Function BuenaSintaxis23(ByVal expresion As String) As Boolean
    Dim Resultado As Boolean = True
    Dim parabre As Integer = 0 'Contador de paréntesis que abre
    Dim parcierra As Integer = 0 'Contador de paréntesis que cierra
    Dim pos As Integer = 0

```

```

While pos < expresion.Length AndAlso Resultado

```

```

    Select Case expresion(pos)

```

```

        Case "("c

```

```

            parabre += 1

```

```

        Case ")"c

```

```

            parcierra += 1

```

```

    End Select

```

```

    If parcierra > parabre Then Resultado = False

```

```

    pos += 1

```

```

End While

```

```

Return Resultado

```

```

End Function

```

'24. Inicia con operador. Ejemplo: +3*5

```

Private Function BuenaSintaxis24(ByVal expresion As String) As Boolean

```

```

    Dim carA As Char = expresion(0)

```

```

    Return Not EsUnOperador(carA)

```

```

End Function

```

'25. Finaliza con operador. Ejemplo: 3*5*

```

Private Function BuenaSintaxis25(ByVal expresion As String) As Boolean

```

```

    Dim carA As Char = expresion(expresion.Length - 1)

```

```

    Return Not EsUnOperador(carA)

```

```

End Function

```

'26. Encuentra una letra seguida de paréntesis que abre. Ejemplo: 3-a(7)-5

```

Private Function BuenaSintaxis26(ByVal expresion As String) As Boolean

```

```

    Dim Resultado As Boolean = True

```

```

    Dim pos As Integer = 0

```

```

While pos < expresion.Length - 1 AndAlso Resultado

```

```

    Dim carA As Char = expresion(pos)

```

```

    Dim carB As Char = expresion(pos + 1)

```

```

    If EsUnaLetra(carA) AndAlso carB = "("c Then Resultado = False

```

```

    pos += 1

```

```

End While

```

```

Return Resultado

```

```

End Function

```

```

Public Function SintaxisCorrecta(ByVal ecuacion As String) As Boolean

```

```

    'Reemplaza las funciones de tres letras por una variable que suma

```

```

    Dim expresion As String = ecuacion.Replace("sen(", "a+").Replace("cos(", "a+").Replace("tan(",
"a+").Replace("abs(", "a+").Replace("asn(", "a+").Replace("acs(", "a+").Replace("atn(",
"a+").Replace("log(", "a+").Replace("cei(", "a+").Replace("exp(", "a+").Replace("sqr(",
"a+").Replace("rcb(", "a+")

```

```

    'Hace las pruebas de sintaxis

```

```

    EsCorrecto(0) = BuenaSintaxis00(expresion)

```

```

    EsCorrecto(1) = BuenaSintaxis01(expresion)

```

```

    EsCorrecto(2) = BuenaSintaxis02(expresion)

```

```

    EsCorrecto(3) = BuenaSintaxis03(expresion)

```

```

    EsCorrecto(4) = BuenaSintaxis04(expresion)

```

```

    EsCorrecto(5) = BuenaSintaxis05(expresion)

```

```

    EsCorrecto(6) = BuenaSintaxis06(expresion)

```

```

    EsCorrecto(7) = BuenaSintaxis07(expresion)

```

```

    EsCorrecto(8) = BuenaSintaxis08(expresion)

```

```

    EsCorrecto(9) = BuenaSintaxis09(expresion)

```

```

    EsCorrecto(10) = BuenaSintaxis10(expresion)

```

```

    EsCorrecto(11) = BuenaSintaxis11(expresion)

```

```

    EsCorrecto(12) = BuenaSintaxis12(expresion)

```



```

EsCorrecto(13) = BuenaSintaxis13(expresion)
EsCorrecto(14) = BuenaSintaxis14(expresion)
EsCorrecto(15) = BuenaSintaxis15(expresion)
EsCorrecto(16) = BuenaSintaxis16(expresion)
EsCorrecto(17) = BuenaSintaxis17(expresion)
EsCorrecto(18) = BuenaSintaxis18(expresion)
EsCorrecto(19) = BuenaSintaxis19(expresion)
EsCorrecto(20) = BuenaSintaxis20(expresion)
EsCorrecto(21) = BuenaSintaxis21(expresion)
EsCorrecto(22) = BuenaSintaxis22(expresion)
EsCorrecto(23) = BuenaSintaxis23(expresion)
EsCorrecto(24) = BuenaSintaxis24(expresion)
EsCorrecto(25) = BuenaSintaxis25(expresion)
EsCorrecto(26) = BuenaSintaxis26(expresion)
Dim Resultado As Boolean = True
Dim cont As Integer = 0

While cont < EsCorrecto.Length AndAlso Resultado
    If EsCorrecto(cont) = False Then Resultado = False
    cont += 1
End While

Return Resultado
End Function

'Transforma la expresión para ser chequeada y analizada */
Public Function Transforma(expresion As String) As String
    'Quita espacios, tabuladores y la vuelve a minúsculas */
    Dim nuevo As String = ""
    For num As Integer = 0 To expresion.Length - 1 Step 1
        Dim letra As Char = expresion(num)
        If letra >= "A" And letra <= "Z" Then letra = Chr(Asc(letra) + Asc(" "c))
        If letra <> " " And letra <> " " Then nuevo += letra.ToString()
    Next

    'Cambia los )) por )+0) porque es requerido al crear las piezas.
    While nuevo.IndexOf("))") <> -1
        nuevo = nuevo.Replace("))", ") +0)")
    End While

    Return nuevo
End Function

'Muestra mensaje de error sintáctico
Public Function MensajesErrorSintaxis(ByVal codigoError As Integer) As String
    Return _mensajeError(codigoError)
End Function
End Class

```

```
Public Class Parte
    Public Tipo As Integer 'Acumulador, función, paréntesis que abre, paréntesis que cierra, operador,
    número, variable
    Public Funcion As Integer '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 Operador As Char '+ suma - resta * multiplicación / división ^ potencia
    Public Numero As Double 'Número literal, por ejemplo: 3.141592
    Public Variable As Integer 'Variable algebraica
    Public Acumulador As Integer 'Usado cuando la expresión se convierte en piezas. Por ejemplo:
    '3 + 2 / 5 se convierte así:
    '|3| |+| |2| | / | |5|
    '|3| |+| |A| A es un identificador de acumulador

    Public Sub New(ByVal tipo As Integer, ByVal funcion As Integer, ByVal operador As Char, ByVal numero As
    Double, ByVal variable As Integer)
        Me.Tipo = tipo
        Me.Funcion = funcion
        Me.Operador = operador
        Me.Numero = numero
        Me.Variable = variable
    End Sub
End Class
```

```
Public Class Pieza
    Public ValorPieza As Double 'Almacena el valor que genera la pieza al evaluarse
    Public Funcion As Integer '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 TipoA As Integer 'La primera parte es un número o una variable o trae el valor de otra pieza
    Public NumeroA As Double 'Es un número literal
    Public VariableA As Integer 'Es una variable
    Public PiezaA As Integer 'Trae el valor de otra pieza */
    Public Operador As Char '+ suma - resta * multiplicación / división ^ potencia
    Public TipoB As Integer 'La segunda parte es un número o una variable o trae el valor de otra pieza
    Public NumeroB As Double 'Es un número literal
    Public VariableB As Integer 'Es una variable
    Public PiezaB As Integer 'Trae el valor de otra pieza

    Public Sub New(ByVal Funcion As Integer, ByVal TipoA As Integer, ByVal NumeroA As Double, ByVal VariableA
As Integer, ByVal PiezaA As Integer, ByVal Operador As Char, ByVal TipoB As Integer, ByVal NumeroB As
Double, ByVal VariableB As Integer, ByVal PiezaB As Integer)
        Me.Funcion = Funcion
        Me.TipoA = TipoA
        Me.NumeroA = NumeroA
        Me.VariableA = VariableA
        Me.PiezaA = PiezaA
        Me.Operador = Operador
        Me.TipoB = TipoB
        Me.NumeroB = NumeroB
        Me.VariableB = VariableB
        Me.PiezaB = PiezaB
    End Sub
End Class
```

```
' Evaluador de expresiones. Versión 3.0
' Autor: Rafael Alberto Moreno Parra
' Fecha: 26 de abril de 2021
'
' Pasos para la evaluación de expresiones algebraicas
' I. Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
' |3.14| |+| |sen(| |4| | / | |x| |)| |*| |( | |7.2| | ^ | |3| |-| |1| |)|
'
' II. Toma las partes y las divide en piezas con la siguiente estructura:
'   acumula = funcion numero/variable/acumula operador numero/variable/acumula
'   Siguiendo el ejemplo anterior sería:
'   A = 7.2 ^ 3
'   B = A - 1
'   C = seno ( 4 / x )
'   D = C * B
'   E = 3.14 + D
'
'   Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación

Public Class Evaluador3
    'Constantes de los diferentes tipos de datos que tendrán las piezas */
    Private Const ESFUNCION As Integer = 1
    Private Const ESPARABRE As Integer = 2
    Private Const ESPARCIERRA As Integer = 3
    Private Const ESOPERADOR As Integer = 4
    Private Const ESNUMERO As Integer = 5
    Private Const ESVARIABLE As Integer = 6
    Private Const ESACUMULA As Integer = 7

    ' Listado de partes en que se divide la expresión
    '   Toma una expresión, por ejemplo: 3.14 + sen( 4 / x ) * ( 7.2 ^ 3 - 1 ) y la divide en partes así:
    '   |3.14| |+| |sen(| |4| | / | |x| |)| |*| |( | |7.2| | ^ | |3| |-| |1| |)|
    '   Cada parte puede tener un número, un operador, una función, un paréntesis que abre o un paréntesis
    '   que cierra
    Private Partes As List(Of Parte) = New List(Of Parte)()

    ' Listado de piezas que ejecutan
    '   Toma las partes y las divide en piezas con la siguiente estructura:
    '   acumula = funcion numero/variable/acumula operador numero/variable/acumula
    '   Siguiendo el ejemplo anterior sería:
    '   A = 7.2 ^ 3
    '   B = A - 1
    '   C = seno ( 4 / x )
    '   D = C * B
    '   E = 3.14 + D

    '   Esas piezas se evalúan de arriba a abajo y así se interpreta la ecuación
    Private Piezas As List(Of Pieza) = New List(Of Pieza)()

    'El arreglo unidimensional que lleva el valor de las variables
    Private VariableAlgebra As Double() = New Double(25) {}

    'Uso del chequeo de sintaxis
    Public Sintaxis As EvaluaSintaxis = New EvaluaSintaxis()

    'Analiza la expresión
    Public Function Analizar(ByVal expresionA As String) As Boolean
        Dim expresionB As String = Sintaxis.Transforma(expresionA)
        Dim chequeo As Boolean = Sintaxis.SintaxisCorrecta(expresionB)
        If chequeo Then
            Partes.Clear()
            Piezas.Clear()
            CrearPartes(expresionB)
            CrearPiezas()
        End If
        Return chequeo
    End Function

    Private Sub CrearPartes(ByVal expresion As String)
        'Debe analizarse con paréntesis
        Dim NuevoA As String = "(" & expresion & ")"

        'Reemplaza las funciones de tres letras por una letra mayúscula */
        Dim NuevoB As String = NuevoA.Replace("sen", "A").Replace("cos", "B").Replace("tan",
"C").Replace("abs", "D").Replace("asn", "E").Replace("acs", "F").Replace("atn", "G").Replace("log",
"H").Replace("cei", "I").Replace("exp", "J").Replace("sqr", "K").Replace("rcb", "L")
```

```

'Va de caracter en caracter
Dim Numero As String = ""
For pos As Integer = 0 To NuevoB.Length - 1
    Dim car As Char = NuevoB(pos)

    'Si es un número lo va acumulando en una cadena
    If (car >= "0"c AndAlso car <= "9"c) OrElse car = "."c Then
        Numero += car.ToString()
    'Si es un operador entonces agrega número (si existía)
    ElseIf car = "+"c OrElse car = "-"c OrElse car = "*"c OrElse car = "/"c OrElse car = "^"c Then
        If Numero.Length > 0 Then
            Partes.Add(New Parte(ESNUMERO, -1, "0"c, CadenaAReal(Numero), 0))
            Numero = ""
        End If
        Partes.Add(New Parte(ESOPERADOR, -1, car, 0, 0))
    'Si es variable */
    ElseIf car >= "a"c AndAlso car <= "z"c Then
        Partes.Add(New Parte(ESVARIABLE, -1, "0"c, 0, Asc(car) - Asc("a"c)))
    'Si es una función (seno, coseno, tangente, ...)
    ElseIf car >= "A"c AndAlso car <= "L"c Then
        Partes.Add(New Parte(ESFUNCION, Asc(car) - Asc("A"c), "0"c, 0, 0))
        pos += 1
    'Si es un paréntesis que abre
    ElseIf car = "("c Then
        Partes.Add(New Parte(ESPARABRE, -1, "0"c, 0, 0))
    'Si es un paréntesis que cierra
    Else
        If Numero.Length > 0 Then
            Partes.Add(New Parte(ESNUMERO, -1, "0"c, CadenaAReal(Numero), 0))
            Numero = ""
        End If
        '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 OrElse Partes(Partes.Count - 2).Tipo = ESFUNCION Then
            Partes.Add(New Parte(ESOPERADOR, -1, "+"c, 0, 0))
            Partes.Add(New Parte(ESNUMERO, -1, "0"c, 0, 0))
        End If

        Partes.Add(New Parte(ESPARCIERRA, -1, "0"c, 0, 0))
    End If
Next
End Sub

'Convierte un número almacenado en una cadena a su valor real */
Private Function CadenaAReal(ByVal Numero As String) As Double
    'Parte entera
    Dim parteEntera As Double = 0
    Dim cont As Integer

    For cont = 0 To Numero.Length - 1
        If Numero(cont) = "."c Then Exit For
        parteEntera = (parteEntera * 10) + (Asc(Numero(cont)) - Asc("0"c))
    Next

    'Parte decimal
    Dim parteDecimal As Double = 0
    Dim multiplica As Double = 1

    For num As Integer = cont + 1 To Numero.Length - 1
        parteDecimal = (parteDecimal * 10) + (Asc(Numero(num)) - Asc("0"c))
        multiplica = multiplica * 10
    Next

    Dim numeroB As Double = parteEntera + parteDecimal / multiplica
    Return numeroB
End Function

'Ahora convierte las partes en las piezas finales de ejecución
Private Sub CrearPiezas()
    Dim tmpParte As Parte
    Dim cont As Integer = Partes.Count - 1

    Do
        tmpParte = Partes(cont)

        If tmpParte.Tipo = ESPARABRE OrElse tmpParte.Tipo = ESFUNCION Then
            GenerarPiezasOperador("^"c, "^"c, cont) 'Evalúa las potencias
            GenerarPiezasOperador("*"c, "/"c, cont) 'Luego evalúa multiplicar y dividir

```

```

GenerarPiezasOperador("+"c, "-"c, cont) 'Finalmente evalúa sumar y restar

If tmpParte.Tipo = ESFUNCION Then 'Agrega la función a la última pieza
    Piezas(Piezas.Count - 1).Funcion = tmpParte.Funcion
End If

'Quita el paréntesis/función que abre y el que cierra, dejando el centro
Partes.RemoveAt(cont)
Partes.RemoveAt(cont + 1)
End If

cont -= 1
Loop While cont >= 0
End Sub

'Genera las piezas buscando determinado operador
Private Sub GenerarPiezasOperador(ByVal operA As Char, ByVal operB As Char, ByVal ini As Integer)
    Dim tmpParte, tmpParteIzq, tmpParteDer As Parte
    Dim cont As Integer = ini + 1

    Do
        tmpParte = Partes(cont)

        If tmpParte.Tipo = ESOPERADOR AndAlso (tmpParte.Operator = operA OrElse tmpParte.Operator = operB)
Then
            tmpParteIzq = Partes(cont - 1)
            tmpParteDer = Partes(cont + 1)

            'Crea Pieza
            Piezas.Add(New Pieza(-1, tmpParteIzq.Tipo, tmpParteIzq.Numero, tmpParteIzq.Variable,
tmpParteIzq.Acumulador, tmpParte.Operator, tmpParteDer.Tipo, tmpParteDer.Numero, tmpParteDer.Variable,
tmpParteDer.Acumulador))

            'Elimina la parte del operador y la siguiente
            Partes.RemoveAt(cont)
            Partes.RemoveAt(cont)

            'Retorna el contador en uno para tomar la siguiente operación
            cont -= 1

            'Cambia la parte anterior por parte que acumula
            tmpParteIzq.Tipo = ESACUMULA
            tmpParteIzq.Acumulador = Piezas.Count - 1
        End If

        cont += 1
    Loop While Partes(cont).Tipo <> ESPARCIERRA
End Sub

'Evalúa la expresión convertida en piezas */
Public Function Evaluar() As Double
    Dim numA, numB As Double, resultado As Double = 0
    Dim tmpPieza As Pieza

    For pos As Integer = 0 To Piezas.Count - 1
        tmpPieza = Piezas(pos)

        Select Case tmpPieza.TipoA
            Case ESNUMERO
                numA = tmpPieza.NumeroA
            Case ESVARIABLE
                numA = VariableAlgebra(tmpPieza.VariableA)
            Case Else
                numA = Piezas(tmpPieza.PiezaA).ValorPieza
        End Select

        Select Case tmpPieza.TipoB
            Case ESNUMERO
                numB = tmpPieza.NumeroB
            Case ESVARIABLE
                numB = VariableAlgebra(tmpPieza.VariableB)
            Case Else
                numB = Piezas(tmpPieza.PiezaB).ValorPieza
        End Select

        Select Case tmpPieza.Operator
            Case "*"c
                resultado = numA * numB
            Case "/"c

```

```

        resultado = numA / numB
    Case "+"c
        resultado = numA + numB
    Case "-"c
        resultado = numA - numB
    Case Else
        resultado = Math.Pow(numA, numB)
End Select

If Double.IsNaN(resultado) Or Double.IsInfinity(resultado) Then Return resultado

Select Case tmpPieza.Funcion
    Case 0
        resultado = Math.Sin(resultado)
    Case 1
        resultado = Math.Cos(resultado)
    Case 2
        resultado = Math.Tan(resultado)
    Case 3
        resultado = Math.Abs(resultado)
    Case 4
        resultado = Math.Asin(resultado)
    Case 5
        resultado = Math.Acos(resultado)
    Case 6
        resultado = Math.Atan(resultado)
    Case 7
        resultado = Math.Log(resultado)
    Case 8
        resultado = Math.Ceiling(resultado)
    Case 9
        resultado = Math.Exp(resultado)
    Case 10
        resultado = Math.Sqrt(resultado)
    Case 11
        resultado = Math.Pow(resultado, 0.3333333333333331)
End Select

If Double.IsNaN(resultado) Or Double.IsInfinity(resultado) Then Return resultado
tmpPieza.ValorPieza = resultado
Next

Return resultado
End Function

'Da valor a las variables que tendrá la expresión algebraica
Public Sub DarValorVariable(ByVal varAlgebra As Char, ByVal valor As Double)
    VariableAlgebra(Asc(varAlgebra) - Asc("a"c)) = valor
End Sub

End Class

```

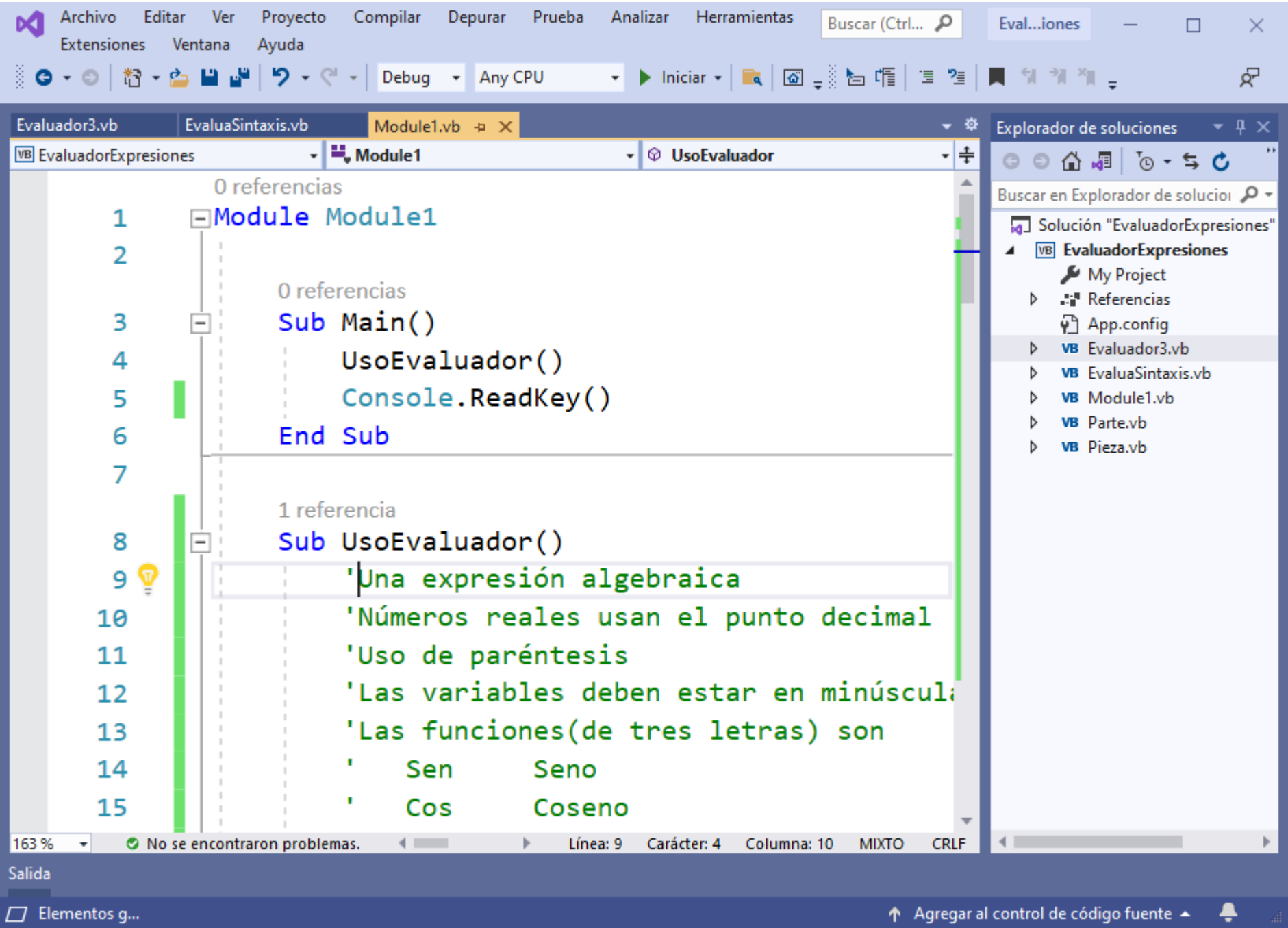



Ilustración 14: En Visual Studio 2019.

Module1.vb

```
Module Module1

Sub Main()
    UsoEvaluador()
    Console.ReadKey()
End Sub

Sub UsoEvaluador()
    'Una expresión algebraica
    'Números reales usan el punto decimal
    'Uso de paréntesis
    'Las variables deben estar en minúsculas van de la 'a' a la 'z' excepto ñ
    'Las funciones(de tres letras) son
    ' Sen      Seno
    ' Cos      Coseno
    ' Tan      Tangente
    ' Abs      Valor absoluto
    ' Asn      Arcoseno
    ' Acs      Arcocoseno
    ' Atn      Arcotangente
    ' Log      Logaritmo Natural
    ' Cei      Valor techo
    ' Exp      Exponencial
    ' Sqr      Raíz cuadrada
    ' Rcb      Raíz Cúbica
    'Los operadores son:
    ' + (suma)
    ' - (resta)
    ' * (multiplicación)
    ' / (división)
    ' ^ (potencia)
    'No se acepta el "-" unario. Luego expresiones como:  4*-2 o (-5+3) o (-x^2) o (-x)^2 son inválidas.
    Dim expresion As String = "Cos(0.004 * x) - (Tan(1.78 / k + h) * SEN(k ^ x) + abs (k^3-h^2))"

    'Instancia el evaluador
```



```

Dim evaluador As Evaluador3 = New Evaluador3()

'Analiza la expresión (valida sintaxis)
If evaluador.Analizar(expresion) Then
    'Si no hay fallos de sintaxis, puede evaluar la expresión

    'Da valores a las variables que deben estar en minúsculas
    evaluador.DarValorVariable("k", 1.6)
    evaluador.DarValorVariable("x", -8.3)
    evaluador.DarValorVariable("h", 9.29)

    'Evalúa la expresión
    Dim resultado As Double = evaluador.Evaluar()
    Console.WriteLine(resultado)

    'Evalúa con ciclos
    Dim azar As New Random()
    For num As Integer = 1 To 10 Step 1
        Dim valor As Double = azar.NextDouble()
        evaluador.DarValorVariable("k", valor)
        resultado = evaluador.Evaluar()
        Console.WriteLine(resultado)
    Next
Else
    For unError As Integer = 0 To evaluador.Sintaxis.EsCorrecto.Length - 1 Step 1
        If evaluador.Sintaxis.EsCorrecto(unError) = False Then
            Console.WriteLine(evaluador.Sintaxis.MensajesErrorSintaxis(unError))
        End If
    Next
End If
End Sub
End Module

```

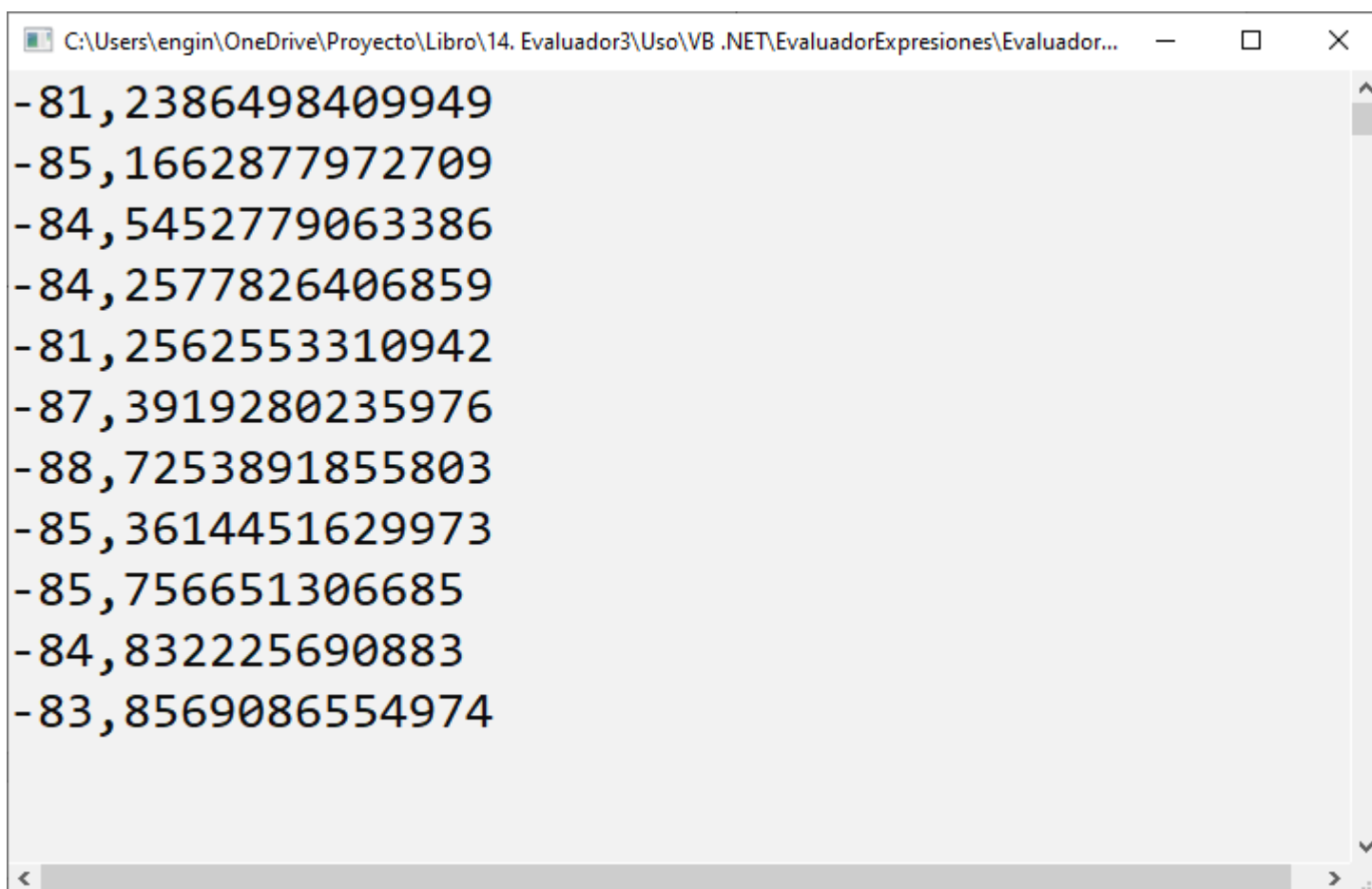


Ilustración 15: Ejecución del programa