z=-x*(-sin(x^-2)+cos(3*x-y/4+9.58))

Desarrollo de un evaluador de expresiones algebraicas. Versión 2.0

enero 22

2013

Más rápida, sencilla y funcional que la versión anterior.

Autor: Rafael Alberto Moreno Parra
http://darwin.50webs.com

En C++, C#, Visual Basic .NET, Java, PHP, JavaScript y Object Pascal

Contenido

Licencia de este libro 10
Licencia del software 10
Marcas registradas 10
Introducción 11
Sobre el Autor 11
¿Por qué una nueva versión del evaluador de expresiones? 12
Las pruebas de desempeño en calcular múltiples valores 16
Las pruebas de desempeño en el análisis de expresiones 18
¿Cómo hacer un evaluador de expresiones algebraicas? 20
¿Una sola clase o varias? 21
¿Qué métodos serán públicos? 21
El método TransformaExpresion() 22
Java 22
C# 22
Visual Basic .NET 22
C++ 22
Object Pascal 23
JavaScript 23
PHP 23
Validación de Sintaxis 25
Llamado a las diferentes pruebas. 25
Java 25
C# 25
Visual Basic .NET 25
C++ 26
Object Pascal 27
JavaScript 27
PHP 28
Validación de Sintaxis. Mensaje para cada prueba. 29
Java 29
C# 29
Visual Basic .NET 29
C++ 30
Object Pascal 30
JavaScript 31
PHP 31
Dos o más operadores estén seguidos 33
Java 33
C# 33
Visual Basic .NET 33
C++ 34
Object Pascal 34
JavaScript 35
PHP 35
Un operador seguido de un paréntesis que cierra 36
Java 36
C# 36
Visual Basic .NET 36
C++ 36

Object Pascal 36
JavaScript 37
PHP 37
Un paréntesis que abre seguido de un operador 38
Java 38
C# 38
Visual Basic .NET 38
C++ 38
Object Pascal 38
JavaScript 39
PHP 39
Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4)) 40
Java 40
C# 40
Visual Basic .NET 40
C++ 40
Object Pascal 40
JavaScript 41
PHP 41
Que haya paréntesis vacío. Ejemplo: 2-()*3 42
Java 42
C# 42
Visual Basic .NET 42
C++ 42
Object Pascal 42
JavaScript 42
PHP 42
Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4 44
Java 44
C# 44
Visual Basic .NET 44
C++ 44
Object Pascal 44
JavaScript 45
PHP 45
Un paréntesis que cierra seguido de un número o paréntesis que abre 46
Java 46
C# 46
Visual Basic .NET 46
C++ 46
Object Pascal 46
JavaScript 47
PHP 47
Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6) 48
Java 48
C# 48
Visual Basic .NET 48
C++ 48
Object Pascal 48
JavaScript 49
PHP 49
Doble punto en un número de tipo real. Ejemplo: 3-2..4+1 7-6.46.1+2 50

Java 50
C# 50
Visual Basic .NET 50
C++ 50
Object Pascal 50
JavaScript 51
PHP 51
Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1 52
Java 52
C# 52
Visual Basic .NET 52
C++ 52
Object Pascal 52
JavaScript 52
PHP 53
Una variable seguida de un punto. Ejemplo: 4-z.1+3 54
Java 54
C# 54
Visual Basic .NET 54
C++ 54
Object Pascal 54
JavaScript 54
PHP 55
Un punto seguido de una variable. Ejemplo: 7-2.p+1 56
Java 56
C# 56
Visual Basic .NET 56
C++ 56
Object Pascal 56
JavaScript 56
PHP 57
Un número antes de una variable. Ejemplo: 3x+1 58
Java 58
C# 58
Visual Basic .NET 58
C++ 58
Object Pascal 58
JavaScript 58
PHP 59
Un número después de una variable. Ejemplo: x21+4 60
Java 60
C# 60
Visual Basic .NET 60
C++ 60
Object Pascal 60
JavaScript 60
PHP 61
Chequea si hay 4 o más letras seguidas 62
Java 62
C# 62
Visual Basic .NET 62
C++ 62

Object Pascal 62
JavaScript 63
PHP 63
Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no 64
Java 64
C# 64
Visual Basic .NET 64
C++ 65
Object Pascal 65
JavaScript 66
PHP 66
Si detecta sólo dos letras seguidas es un error 68
Java 68
C# 68
Visual Basic .NET 68
C++ 68
Object Pascal 68
JavaScript 69
PHP 69
Antes de paréntesis que abre hay una letra 70
Java 70
C# 70
Visual Basic .NET 70
C++ 70
Object Pascal 70
JavaScript 71
PHP 71
Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x) 72
Java 72
C# 72
Visual Basic .NET 72
C++ 72
Object Pascal 72
JavaScript 72
PHP 72
Después de operador sigue un punto. Ejemplo: -.3+7 74
Java 74
C# 74
Visual Basic .NET 74
C++ 74
Object Pascal 74
JavaScript 74
PHP 75
Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4) 76
Java 76
C# 76
Visual Basic .NET 76
C++ 76
Object Pascal 76
JavaScript 76
PHP 76
Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6) 78

Java 78
C# 78
Visual Basic .NET 78
C++ 78
Object Pascal 78
JavaScript 78
PHP 78
Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2 80
Java 80
C# 80
Visual Basic .NET 80
C++ 80
Object Pascal 80
JavaScript 80
PHP 80
Punto seguido de operador. Ejemplo: 5.*9+1 82
Java 82
C# 82
Visual Basic .NET 82
C++ 82
Object Pascal 82
JavaScript 82
PHP 83
Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5 84
Java 84
C# 84
Visual Basic .NET 84
C++ 84
Object Pascal 84
JavaScript 84
PHP 84
El método Analizar_menos_unarios() 86
Java 86
C# 87
Visual Basic .NET 87
C++ 88
Object Pascal 88
JavaScript 89
PHP 89
El método "Analiza_Expresión" 91
Orden de llamado en Analiza_Expresion 93
Java 93
C# 93
Visual Basic .NET 93
C++ 93
Object Pascal 93
JavaScript 93
PHP 93
Partiendo la expresión y llevándola a la estructura Pieza_Simple 94
Atributos requeridos 94
Generando el ArrayList PiezaSimple 97
Java 97

C# 98
Visual Basic .NET 99
C++ 100
Object Pascal 101
JavaScript 102
PHP 103
¿Qué es cada Nodo en PiezaSimple? 105
Java 105
C# 105
Visual Basic .NET 105
C++ 106
Object Pascal 106
JavaScript 107
PHP 108
Usando el ArrayList PiezaSimple para generar el ArrayList PiezaEjecuta 109
Java 109
C# 110
Visual Basic .NET 111
C++ 111
Object Pascal 112
JavaScript 113
PHP 114
¿Qué es cada Nodo en PiezaEjecuta? 115
Java 115
C# 115
Visual Basic .NET 116
C++ 117
Object Pascal 117
JavaScript 119
PHP 120
El método Leer_Variables() 121
Java 121
C# 121
Visual Basic .NET 121
C++ 121
Object Pascal 121
JavaScript 121
PHP 121
El método Evalúa_Expresión() 122
Java 122
C# 122
Visual Basic .NET 123
C++ 124
Object Pascal 125
JavaScript 125
PHP 126
Un ejemplo de uso del evaluador de expresiones 128
Java 128
C# 129
Visual Basic .NET 130
C++ 132
Object Pascal 133

JavaScript 135
PHP 136
Conclusiones 138
Anexo 1. Código completo en Java 139
Anexo 2. Código completo en C# 148
Anexo 3. Código completo en Visual Basic .NET 158
Anexo 4. Código completo en C++ 170
Anexo 5. Código completo en Object Pascal 181
Anexo 6. Código completo en JavaScript 195
Anexo 7. Código completo en PHP 205

Dedicado a mi familia: José Alberto (mi padre), María del Rosario (mi madre), Diana Pilar (mi hermana) y Sally (mi gata).

Licencia de este libro

Licencia del software

Todo el software desarrollado aquí tiene licencia LGPL "Lesser General Public License"

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 ® Office ® Enlace: http://office.microsoft.com/en-us/excel

Microsoft ® Visual Studio ® Enlace: http://www.microsoft.com/visualstudio/ (Incluye C#, Visual Basic .NET, Visual C++)

Oracle ® Java ® Enlace: http://www.oracle.com/technetwork/java/index.html

Borland® Delphi 7® Enlace: http://www.embarcadero.com/products/delphi

PHP es una marca en poder de PHP.net. Enlace: http://php.net/

Oracle ® JavaScript ® Enlace: http://www.oracle.com/index.html

Oracle ® NetBeans ® Enlace: http://netbeans.org/about/legal/copyright.html

Piriform ® Speccy ® Enlace: http://www.piriform.com/speccy

Introducción

Así como existen diversos algoritmos para ordenar un arreglo unidimensional, algunos mejores que otros en cuanto a velocidad, uso de memoria o sencillez, sucede lo mismo con la evaluación de expresiones.

Mi investigación sobre vida artificial ha pisado el terreno de los algoritmos genéticos. Allí trabajo en el tema de la regresión simbólica, la cual consiste en encontrar la mejor ecuación que describa el comportamiento de una serie de datos. Debido a que es un proceso de intensivos cálculos matemáticos, la necesidad de un evaluador de expresiones rápido es imperiosa. Y fue durante ese desarrollo que encontré una manera mucho mejor de evaluar expresiones mucho más rápida y hasta más sencilla. ¿Deja obsoleta la versión anterior? La respuesta es sí, pero cabe anotar que es bueno tener registro de los diferentes algoritmos utilizados para resolver un problema. Quizás dentro de un año se publica una nueva versión más rápida o más sencilla, no lo sé, pero lo importante es ir mejorando cada vez más.

¿Qué es un evaluador de expresiones? Tenemos la expresión "K/(3.78 + cos(X/7.96+Y*1.554)-tan(3.7/X)+B)" almacenada en una variable de tipo Cadena (String). Necesitamos que un algoritmo resuelva esa expresión y retorne el valor cuantitativo una vez que las variables K, X, Y y B se les asignen un valor real.

¿Cómo hacer eso? Esa pregunta es respondida paso a paso en este libro.

¿Por qué aprender sobre cómo está hecho un evaluador de expresiones si esa librería o componente se puede conseguir fácilmente en Internet lista para ser usada? Hay varias respuestas a esta pregunta, aquí expongo algunas:

Usted es estudiante de Ingeniería de Sistemas o alguna carrera afín en la cual se estudia sobre algoritmos y programación; uno de los temas es el tratamiento de cadenas (Strings) y el evaluador de expresiones es un caso típico de estudio.

Usted es un desarrollador de software enfocado a los sistemas de información y ha recibido requisitos como por ejemplo*: "el software debe calcular la factura telefónica dependiendo si el consumo fue en día normal o en un día festivo o en un día especial (como el día del padre o el día de la madre), en algún horario especial"*, usualmente este tipo de requisitos para cumplirlos se debe:

a. Generar el algoritmo.
b. Implementarlo en el lenguaje de programación de la aplicación original.
c. Compilar y esperar que la nueva adición no afecte negativamente el código original.
d. Probar y generar una nueva versión.

Si el nuevo requisito llegase a cambiar, se debe repetir los pasos a. b. c. y d. con el continuo desgaste y el riesgo de dañar los datos o código de la aplicación original, además que el usuario final requerirá siempre del desarrollador para implementar una nueva idea de facturación para mejorar las ventas.

En vez de volver a programar, es mucho mejor tener un interpretador de lo que desea el usuario final con la facturación. Entonces el usuario solo ingresaría la nueva fórmula de facturación:

Valor Factura = (Consumo() * DescuentoDiaEspecial() – BonoRegaloEmpresa() + RecargoServicioEspecial()) * Impuesto()

La factura es calculada mediante una expresión algebraica, ¿pero cómo interpretarla si es una cadena (string)? Entender los evaluadores de expresiones algebraicos es un buen inicio para poder implementar evaluadores más complejos usados al interior de un sistema de información. Esta funcionalidad hará que un sistema de información sea bastante flexible, le dé más longevidad e inclusive el usuario final podrá por sí mismo hacer los cambios sin tener a un desarrollador siempre al lado.

Sobre el Autor

Rafael Alberto Moreno Parra. Ingeniero de Sistemas. Maestría en Ingeniería con Énfasis en Ingeniería de Sistemas y Computación.

Sitio web: http://darwin.50webs.com y correo: ramsoftware@gmail.com

El evaluador de expresiones es un componente fundamental para la investigación sobre la regresión simbólica, cuyo objetivo es buscar el patrón de una serie de datos. Ese patrón es una expresión matemática.

¿Por qué una nueva versión del evaluador de expresiones?

La razón es simple: velocidad. Este nuevo evaluador de expresiones es mucho más rápido que la versión mostrada en el libro anterior (fecha de publicación: 2012). A continuación se muestra una prueba de velocidad usando la herramienta Profiler de NetBeans 7.2

Imagen 1: Se hace uso de NetBeans IDE 7.2

En el siguiente equipo:

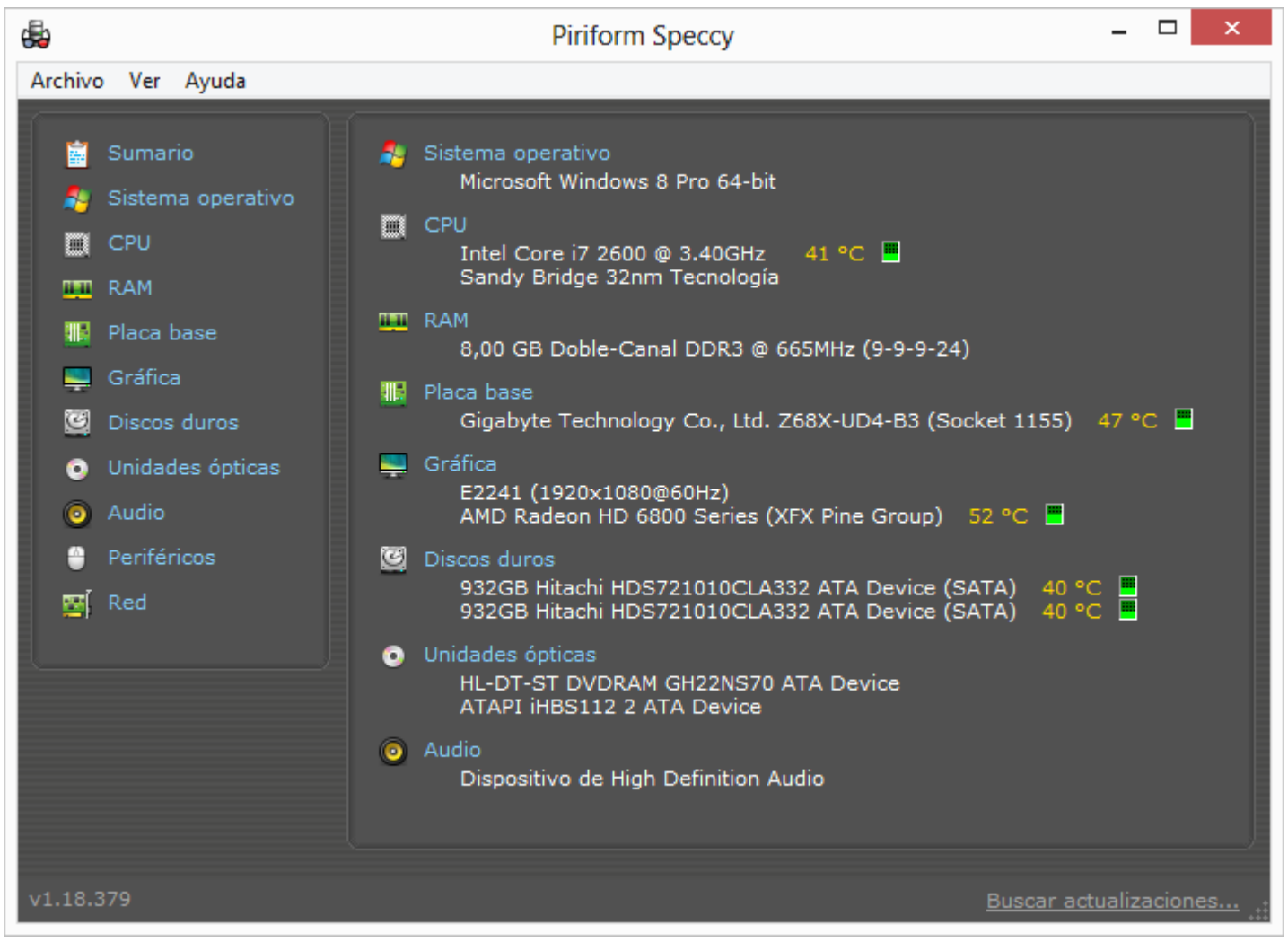


Imagen 2: Resumen en donde se prueba el evaluador de expresiones

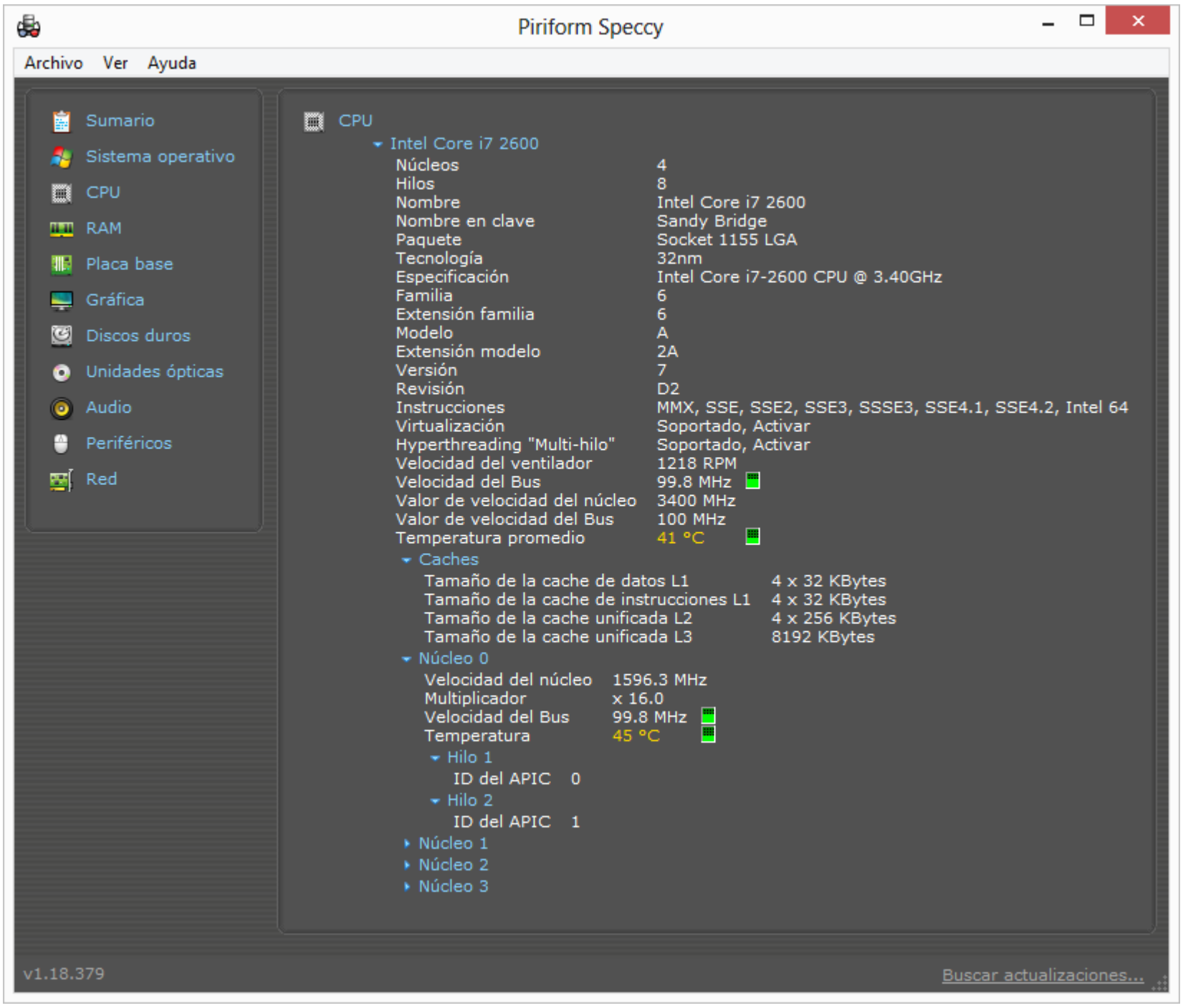


Imagen 3: CPU del equipo de pruebas

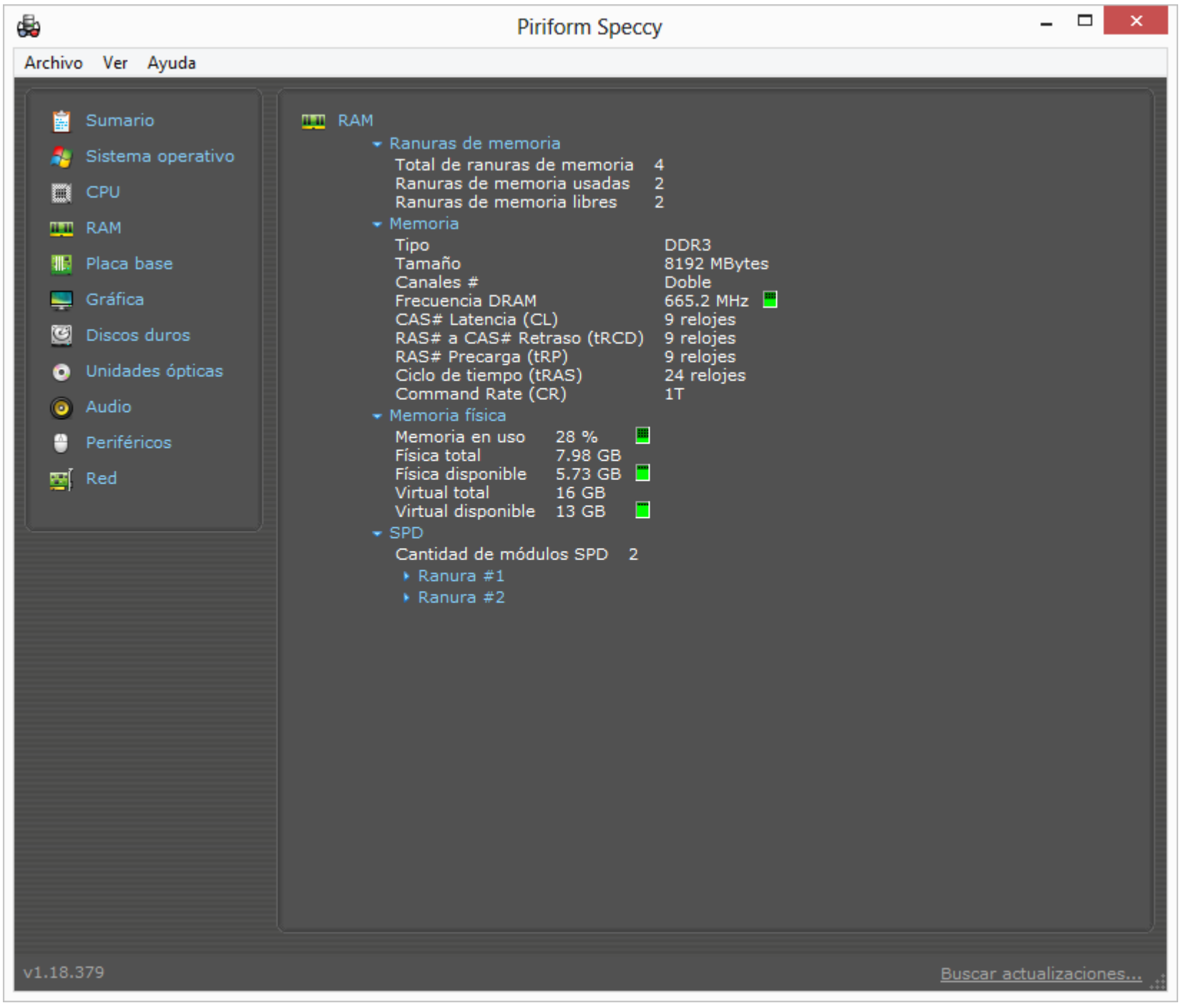


Imagen 4: Memoria del equipo de pruebas

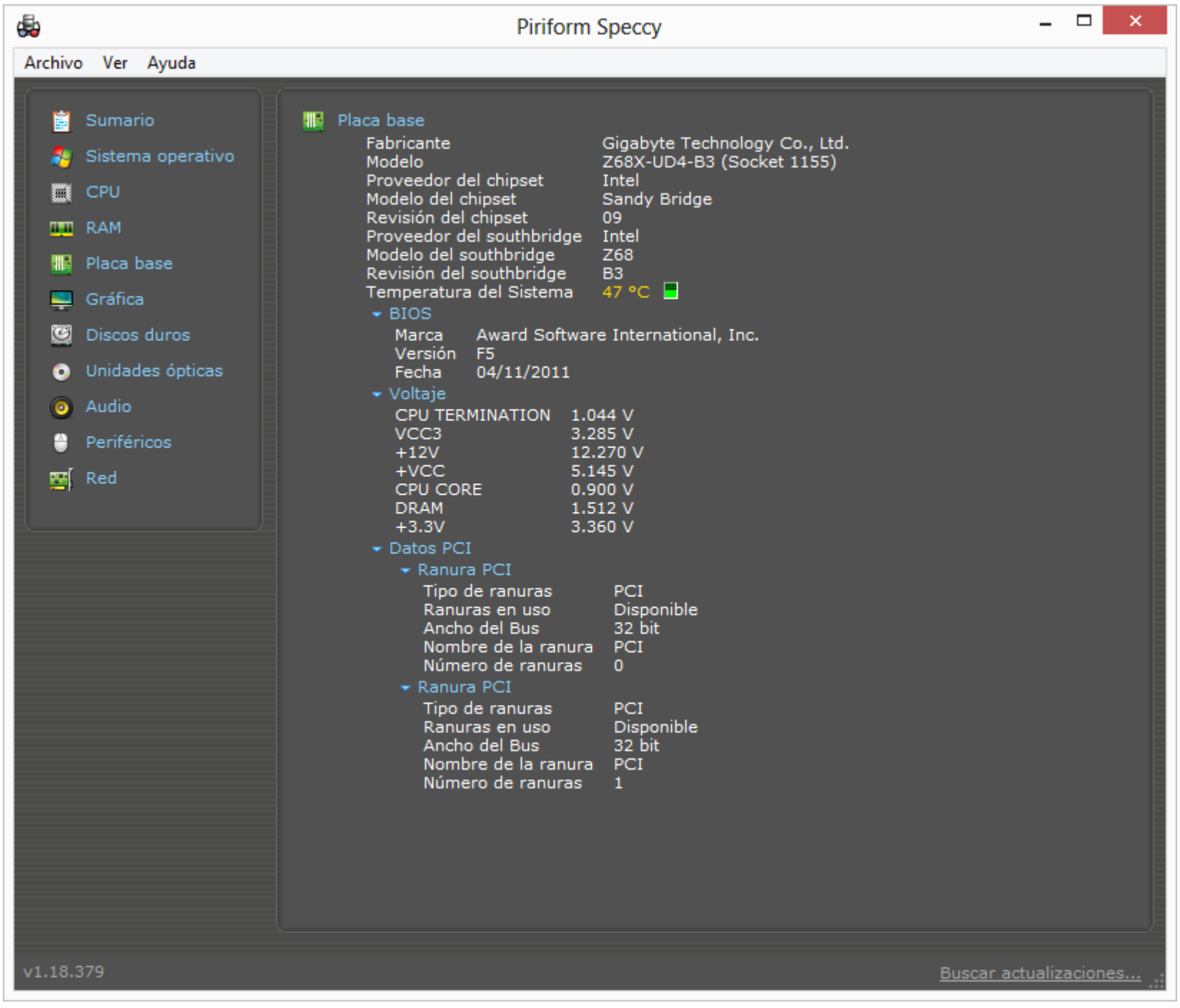


Imagen 5: Placa base del equipo de pruebas

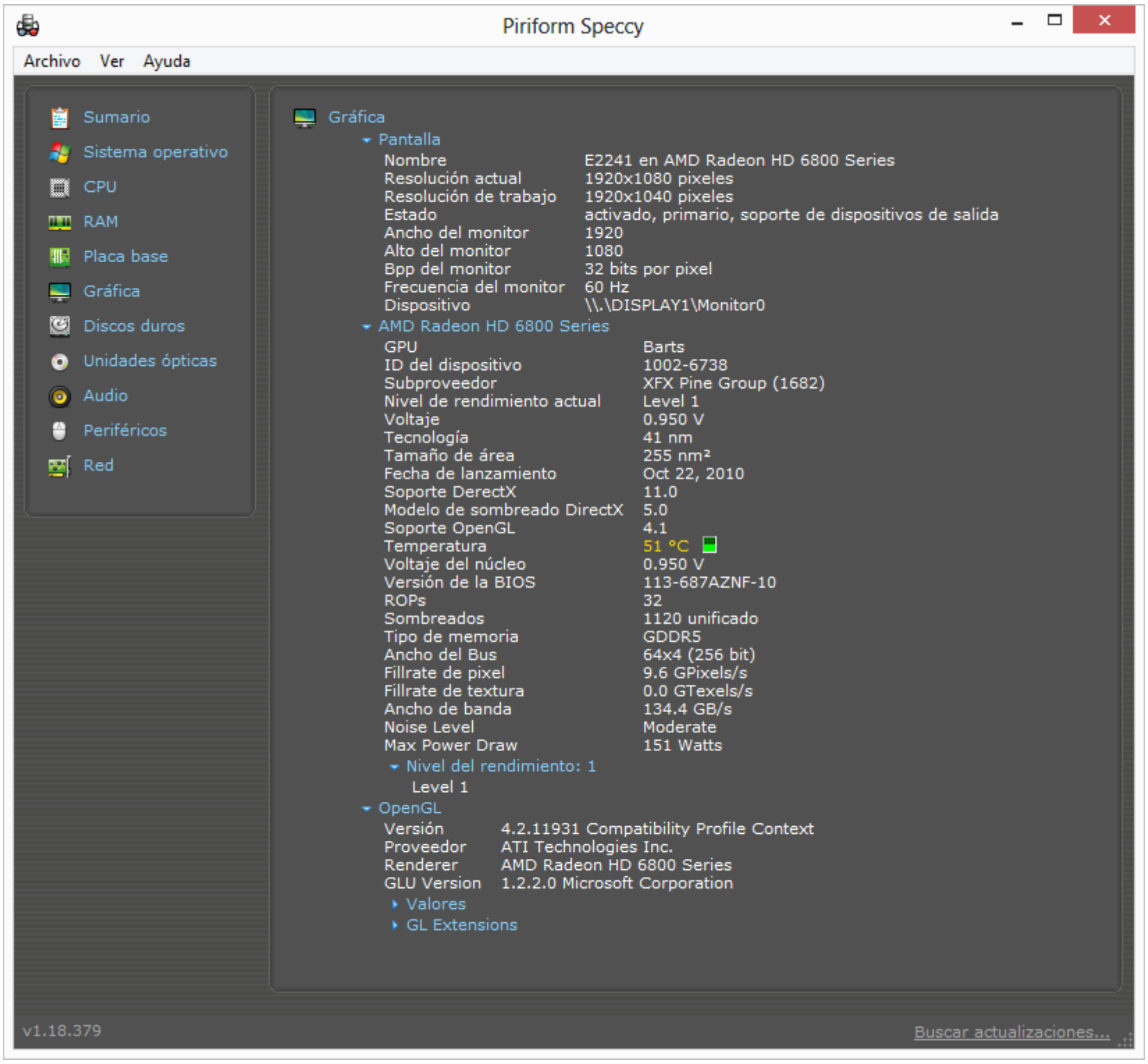


Imagen 6: Tarjeta de video del equipo de pruebas

Las pruebas de desempeño en calcular múltiples valores

Un evaluador de expresiones se utiliza de la siguiente forma:

```
Algoritmo de graficación matemática z=f(x,y)
Inicio
    Leer_expresión()
    TransformaExpresion()
    Chequea_sintaxis_expresión()
    Analiza_menos_unarios()
    Analiza_Expresión()
    Leer Xinicial, Xfinal, Yinicial, Yfinal
    Desde x=Xinicial hasta Xfinal
        Desde y=Yinicial hasta Yfinal
            Leer_variables
            z = Evalúa_Expresion()
            Graficar (x,y,z)
        Fin Desde
    Fin Desde
Fin
```

El procedimiento o método crítico es el "Evalúa_Expresion()" que está en el interior de los dos ciclos anidados. El esfuerzo en este libro es hacer ese método tengo el mejor desempeño posible. Las siguientes pruebas ilustran como mejoró ese método en la versión actual del evaluador de expresiones.

Se generan al azar multitud de ecuaciones de una sola variable independiente Y=f(X) de gran longitud (200 caracteres). Ejemplos:

```
Y = sen(9.18+atn(X/X+88.96))/X+(X-57.78*96.47-abs(41.35*X)+9.30+71.1+sen(((asn(87.46-(95.33-
atn(64.61*71.80+92.0*abs(abs(tan(acs(X)+X/X)-(X)))*(asn((X-tan((asn(cos(56.51-X/asn((X+sen(sen(X))*X*sen(((60.73/49.54/X)-
```

```
tan((53.37+X)+42.57+23.58))-89.43+X)*(atn(X-sen(X*19.48/asn(acs(65.30/72.86/asn(tan(X)/96.19)/asn((3.68-abs((tan(sen(6.34)-
asn((47.29-66.19+atn((tan(38.22*(asn((asn(X-60.99-X-X+(7.92+X/4.47))*35.95-abs(16.1-
((96.7/asn(41.93+abs((65.9*(X))))))))))))))))))))))))))))))))))))))))))

Y = 87.69*X*20.27+sen(68.39)+abs(abs((76.70)-3.16*86.0)*7.9+3.98-51.98-abs(tan(91.29+((X)*asn((sen((X/86.83/36.30)-
atn(X*78.36/56.4*X+(X*X+94.19)*X-82.13)))-abs(96.73-X/X)))-abs(72.18-X*31.63*X)*X*abs(51.78))+abs(abs(sen(62.13-
X*X))/13.83-X+(65.63*88.50)*91.79)-89.58*(X-
abs(asn((atn(cos((55.54/(abs((82.4+X+X))*X*atn(91.73/X*91.54+X*asn(X*(60.76)/X)-(sen(sen(X)/sen((atn(atn(12.66)))+X*39.0))-
56.40)/cos(54.32*70.42/atn(80.32)))))))))))))

Y = (37.37)-acs((tan(X-sen(sen(43.63)*(11.0+atn(61.46-atn(X/acs(((abs(((cos(67.63)*sen(X)-X/(sen((tan((X)+X-cos(X-
4.54)/18.80+(tan(45.56*cos(4.59*91.87)-(abs((75.70*atn(asn(20.62/tan(abs(asn(sen(acs(29.78)+acs(87.31/tan(20.29)-
cos(20.34/26.4-X)-20.97/(84.66*71.58)*6.93*44.57/sen(X*33.91*atn(acs(X))))/74.47))/sen(X-tan(abs(9.35+12.83*X)-
sen(abs(cos(sen(abs(X/tan(X-X-X-atn((11.6)))-(36.99))/((53.7/atn(67.76*tan(X))*35.38+(1.50-36.91*acs(41.27-85.0-
atn((70.82/atn(16.73*atn(62.64))))))))))))))))))))))))))))))))))))))))))
```

Por cada ecuación, se analiza y luego se evalúa el valor de Y cuando X toma los valores desde -100 hasta 100 aumentando de 1 en 1. Ejemplo: Y=f(-100), Y=f(-99), … Y=f(70), … Y=f(100)

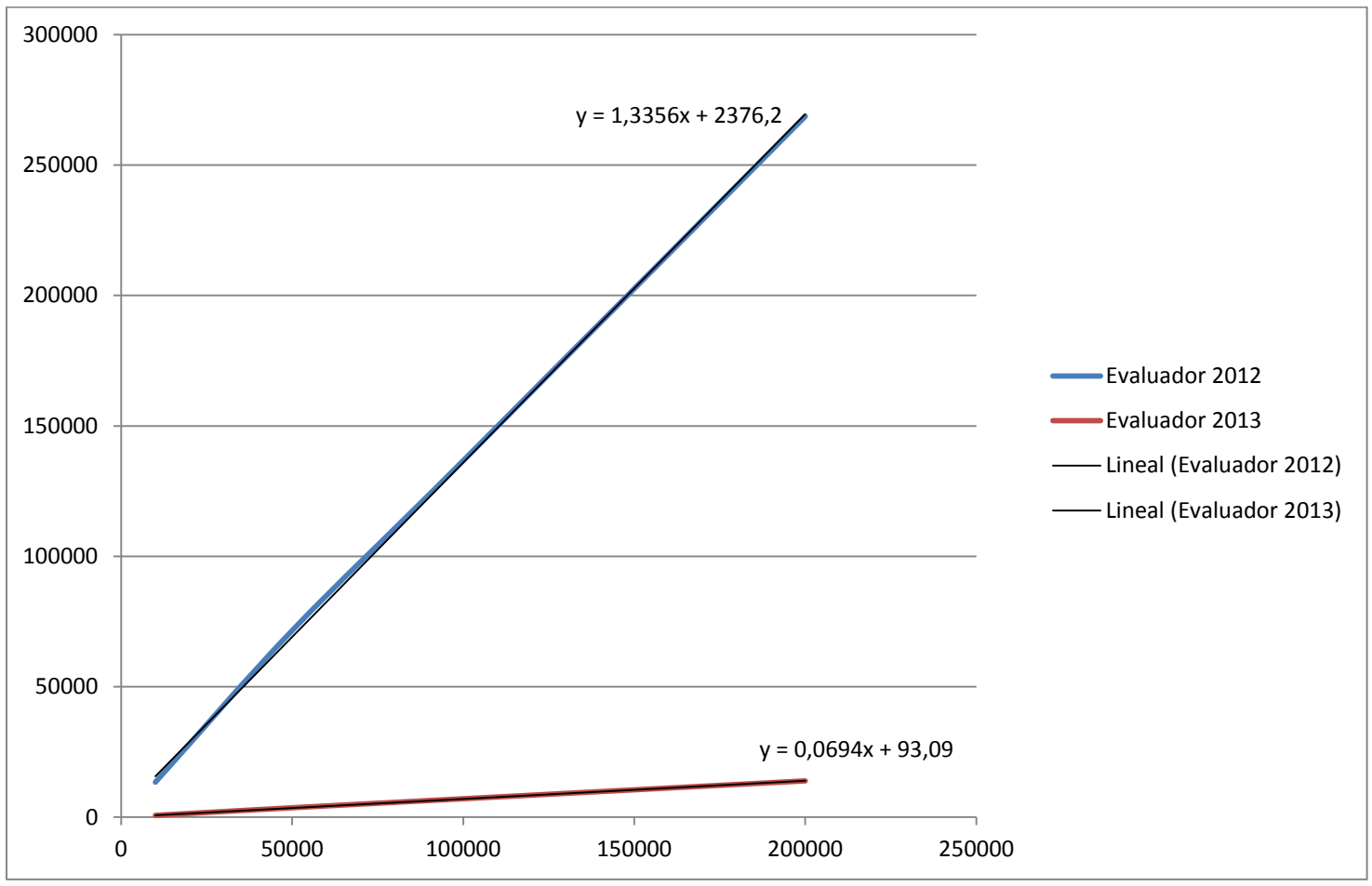
Se genera la estadística: Tiempo tomado por el método "Evalúa_Expresion()" para dado un valor de X, calcular el valor de Y.

Número de ecuaciones: 10.000	Evaluador 2012	Evaluador 2013
	Tiempo de cálculo (milisegundos)	Tiempo de cálculo (milisegundos)
Prueba 1	12.889	718
Prueba 2	14.411	710
Prueba 3	12.231	687
Prueba 4	13.102	704
Prueba 5	14.530	725
Promedio	**13.432,6**	**708,8**

Número de ecuaciones: 50.000	Evaluador 2012	Evaluador 2013
	Tiempo de cálculo (milisegundos)	Tiempo de cálculo (milisegundos)
Prueba 1	72.775	3.763
Prueba 2	81.397	3.506
Prueba 3	69.931	3.675
Prueba 4	67.302	3.618
Prueba 5	66.441	3.570
Promedio	**71.569,2**	**3626,4**

Número de ecuaciones: 100.000	Evaluador 2012	Evaluador 2013
	Tiempo de cálculo (milisegundos)	Tiempo de cálculo (milisegundos)
Prueba 1	133.583	7.131
Prueba 2	136.054	7.190
Prueba 3	148.704	6.819
Prueba 4	131.505	7.258
Prueba 5	133.626	7.002
Promedio	**136.694,4**	**7.080**

Número de ecuaciones: 200.000	Evaluador 2012	Evaluador 2013
	Tiempo de cálculo (milisegundos)	Tiempo de cálculo (milisegundos)
Prueba 1	295.274	13.547
Prueba 2	246.436	13.522
Prueba 3	285.384	14.163
Prueba 4	267.741	14.470
Prueba 5	248.366	13.940
Promedio	**268.640,2**	**13.928,4**



El nuevo evaluador que se explica en este libro (2013) es 19 veces más rápido que el evaluador del año anterior. Aparte de eso, la pendiente de la curva que representa el tiempo tomado por el evaluador del año pasado es considerablemente mayor que la versión de este año. Quizás en una versión futura del algoritmo ¿para el 2014? se obtenga una mejor velocidad en cálculo.

Las pruebas de desempeño en el análisis de expresiones

En las pruebas anteriores se mostró la velocidad de calcular valores del evaluador de expresiones cuando la expresión ya había sido analizada. La razón por mostrar esa comparativa primero es porque los evaluadores se utilizan primordialmente para hacer operaciones repetitivas. Pero falta ver el desempeño en la primera parte:

```
Algoritmo de graficación matemática z=f(x,y)
Inicio
        Leer_expresión()
        TransformaExpresion()
        Chequea_sintaxis_expresión()
        Analiza_menos_unarios()
        Analiza_Expresión()
        Leer Xinicial, Xfinal, Yinicial, Yfinal
        Desde x=Xinicial hasta Xfinal
                Desde y=Yinicial hasta Yfinal
                        Leer_variables
                        z = Evalúa_Expresion()
                        Graficar (x,y,z)
                Fin Desde
        Fin Desde
Fin
```

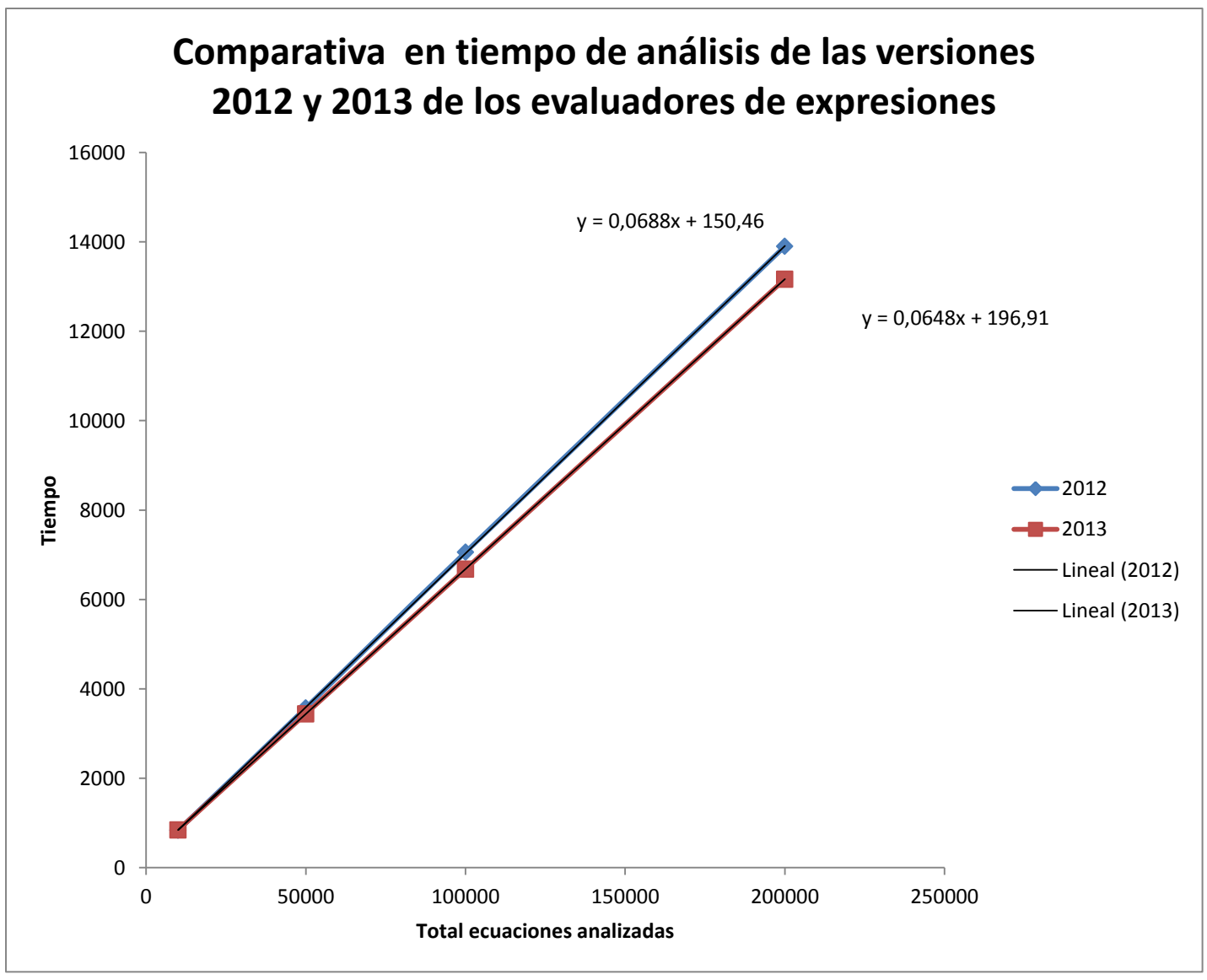
¿Cuánto tarda desde Leer_expresión() hasta Analiza_Expresión() en la nueva versión en comparación con la versión anterior?

Número de ecuaciones: 10.000	Evaluador 2012	Evaluador 2013
	Tiempo de análisis (milisegundos)	Tiempo de análisis (milisegundos)
Prueba 1	815	790
Prueba 2	840	794
Prueba 3	858	932
Prueba 4	842	916
Prueba 5	823	786
Promedio	**835,6**	**843,6**

Número de ecuaciones: 50.000	Evaluador 2012	Evaluador 2013
	Tiempo de análisis (milisegundos)	Tiempo de análisis (milisegundos)
Prueba 1	3.487	3.289
Prueba 2	3.546	3.299
Prueba 3	3.789	4.014
Prueba 4	3.467	3.246
Prueba 5	3.591	3.367
Promedio	**3.576**	**3.443**

Número de ecuaciones: 100.000	Evaluador 2012	Evaluador 2013
	Tiempo de análisis (milisegundos)	Tiempo de análisis (milisegundos)
Prueba 1	7.034	6.348
Prueba 2	6.961	6.371
Prueba 3	7.441	7.910
Prueba 4	6.951	6.392
Prueba 5	6.896	6.362
Promedio	**7.056,6**	**6.676,6**

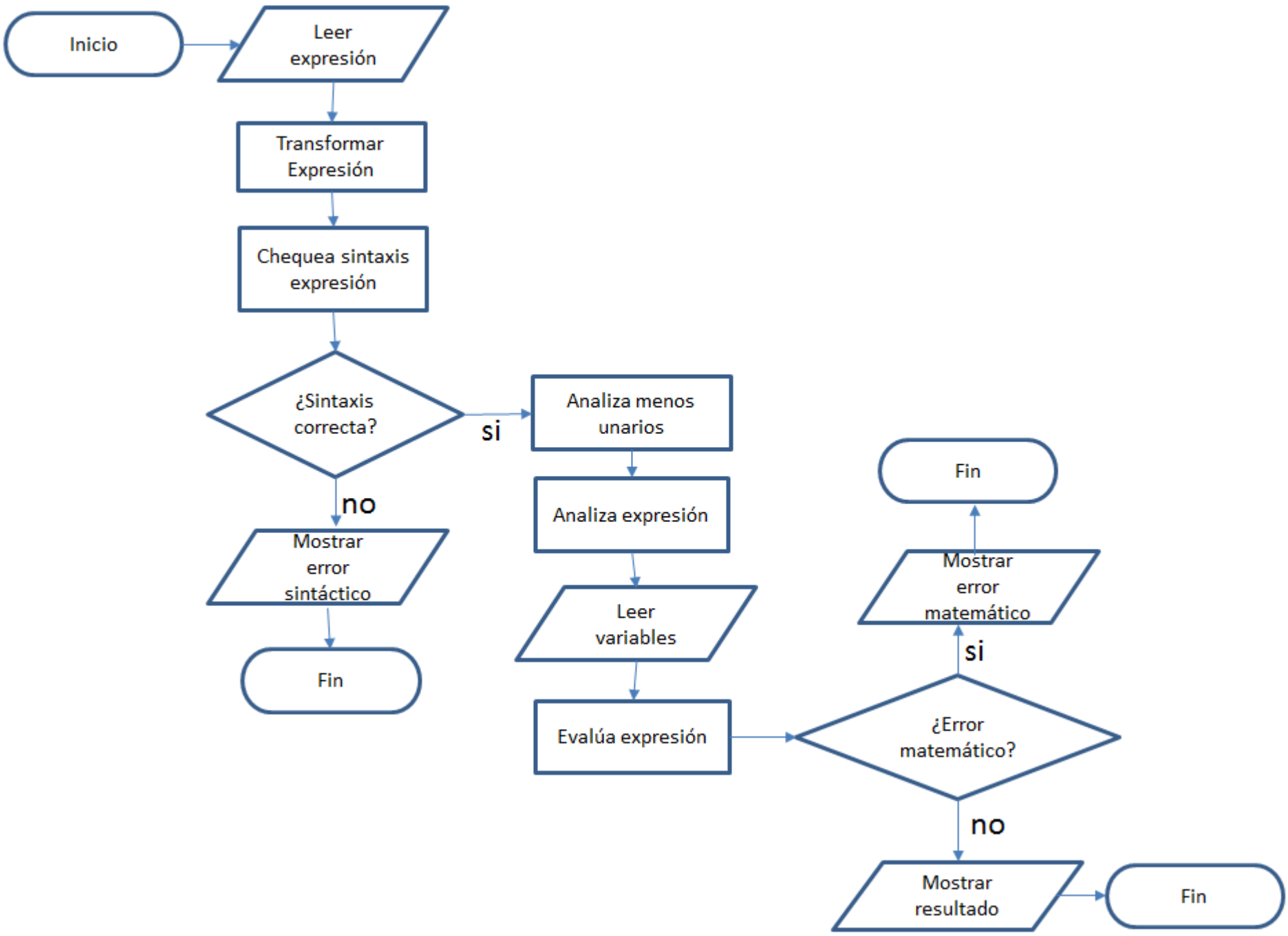
Número de ecuaciones: 200.000	Evaluador 2012	Evaluador 2013
	Tiempo de análisis (milisegundos)	Tiempo de análisis (milisegundos)
Prueba 1	13.768	12.691
Prueba 2	14.025	13.213
Prueba 3	13.743	13.060
Prueba 4	14.040	13.654
Prueba 5	13.923	13.203
Promedio	**13.899,8**	**13.164,2**



El nuevo evaluador es un poco más rápido que la versión anterior y eso que adicionalmente incluye una nueva funcionalidad: el manejo del menos unario.

¿Cómo hacer un evaluador de expresiones algebraicas?

Este es el algoritmo básico de un evaluador de expresiones:



Estas son las partes:

1. Leer expresión: Almacena en una variable de tipo String la expresión matemática por ejemplo: " 4 + sen(x / y * 3.89)-abs(y / x + 6.89)".

2. Transformar Expresión: Se convierte a minúsculas la expresión, se retiran los caracteres no permitidos que hayan en la cadena como los espacios, tabuladores, caracteres extraños y sólo se dejan los permitidos dentro de una expresión algebraica. Se encierra la expresión entre paréntesis.

3. Chequea sintaxis expresión: ¿la expresión cumple con las estrictas reglas sintácticas del algebra? Son 25 revisiones.

4. Analiza menos unarios: Expresiones que inician con negativo, por ejemplo, "-cos(7*z+x)" quedan convertidas a "0-cos(7*z+x)"; un paréntesis que abre seguido de un negativo, por ejemplo, "5*(-t/9+7)" quedan convertidas en "5*(0-t/9+7)"; expresiones como "5^-cos(k+3)+7.1" se convierten a "5^(0-cos(k+3))+7.1"

5. Analiza expresión: Procede a convertir esa ecuación en una estructura que permita posteriormente evaluarla.

6. Leer variables: El evaluador soporta variables en la expresión, de la "a" a la "z", luego para evaluar la expresión, hay que darle valores numéricos a esas variables.

7. Evalúa expresión: Proceder a evaluar la expresión y retorna el resultado. Si hay un error matemático (una división entre cero, raíz cuadrada de un número negativo, un arco seno de un número mayor de 1 o menor de -1) entonces el evaluador retorna o un NaN (Not a Number) o Infinity (infinito, por ejemplo en el caso de una tangente de 90 grados).

El procedimiento crítico en desempeño es el séptimo (7) porque, por lo general, cuando se trabajan con expresiones matemáticas es para hacer operaciones intensivas como graficar.

Ejemplo de uso del algoritmo:

```
Algoritmo de graficación matemática z=f(x,y)
Inicio
        Leer_expresión
        TransformaExpresion()
        Chequea_sintaxis_expresión()
        Analiza_menos_unarios()
        Analiza_Expresión()
        Leer Xinicial, Xfinal, Yinicial, Yfinal
        Desde x=Xinicial hasta Xfinal
                Desde y=Yinicial hasta Yfinal
                        Leer_variables()
                        z = Evalúa_Expresion()
                        Graficar (x,y,z)
                Fin Desde
        Fin Desde
Fin
```

En el algoritmo se puede ver que los métodos "Leer_Variables()" y "Evalúa_Expresion()" son los más usados y es crítico que sean muy rápidos, en cambio, los métodos: "TransformaExpresion", "Chequea_sintaxis_expresion", "Analiza_menos_unarios" y "Analiza_Expresion" sólo son llamados una vez. En conclusión, se puede sacrificar desempeño en los métodos que se llaman una vez si esto significa darle mayor velocidad a los que son llamados múltiples veces.

¿Una sola clase o varias?

Existe una métrica de software la cual recomienda mucho que haya pocos métodos por clase, pero el chequeo de sintaxis tiene 21 revisiones donde cada una es un método que retorna true (verdadero) si presenta un error y false (falso) si no se encontró ese error sintáctico buscado. Sólo es fallar en una revisión para rechazar la expresión y no continuar con el análisis. No hay sentido en partir esta clase. Pero queda la pregunta si los métodos y atributos del análisis y posterior evaluación pueden reunirse junto con los métodos de chequeo de sintaxis y hacer una sola clase autosuficiente. El diagrama de flujo anterior muestra que hay un curso lógico en la evaluación de expresiones. Luego la decisión tomada es hacer una sola clase y en eso se diferencia de la versión del 2012 que eran dos clases separadas.

¿Qué métodos serán públicos?

Esta es otra decisión no sencilla, si se llama al método de análisis de la expresión y esta expresión tiene un error de sintaxis, el método colapsa. En otras palabras, es requerido que haya pasado las pruebas de sintaxis para poder analizarla, y por supuesto, no se puede evaluar la expresión si no ha sido previamente analizada. En los lenguajes de programación existen clases y métodos que operan suponiendo que las condiciones están dados, por ejemplo, en C, si se va a almacenar una cadena, se supone que antes se ha asignado la suficiente memoria para esta operación, caso contrario, el programa colapsa.

Este libro va orientado al desarrollador de software y se advierte entonces que los métodos deben llamarse en un orden, además se debe validar que retorna cada método para poder continuar con el siguiente en el orden lógico.

A continuación se ve en detalle cada método y cómo fue implementado en cada lenguaje de programación.

El método TransformaExpresion()

Este método se encarga de convertir a minúsculas y encerrar en paréntesis la expresión, luego revisar carácter por carácter filtrando sólo aquellos que son permitidos dentro de una expresión algebraica que son:

abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()

Java

```java
//Retira caracteres inválidos. Pone la expresión entre paréntesis.
public final String TransformaExpresion(String expr)
{
  if (expr == null) return "";
  String validos = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
  StringBuilder nuevaExpr = new StringBuilder();
  String expr2 = expr.toLowerCase();
  nuevaExpr.append('(');
  for(int pos = 0; pos < expr2.length(); pos++)
  {
    char letra = expr2.charAt(pos);
    for(int valida = 0; valida < validos.length(); valida++)
      if (letra == validos.charAt(valida))
      {
        nuevaExpr.append(letra);
        break;
      }
  }
  nuevaExpr.append(')');
  return nuevaExpr.toString();
}
```

C#

```csharp
//Retira caracteres inválidos. Pone la expresión entre paréntesis.
public String TransformaExpresion(String expr)
{
  if (expr == null) return "";
  String validos = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
  StringBuilder nuevaExpr = new StringBuilder();
  String expr2 = expr.ToLower();
  nuevaExpr.Append('(');
  for(int pos = 0; pos < expr2.Length; pos++)
  {
    char letra = expr2[pos];
    for(int valida = 0; valida < validos.Length; valida++)
      if (letra == validos[valida])
      {
        nuevaExpr.Append(letra);
        break;
      }
  }
  nuevaExpr.Append(')');
  return nuevaExpr.ToString();
}
```

Visual Basic .NET

```vb
'Retira caracteres inválidos. Pone la expresión entre paréntesis.
Public Function TransformaExpresion(expr As [String]) As [String]
  If expr Is Nothing Then
    Return ""
  End If
  Dim validos As [String] = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()"
  Dim nuevaExpr As New Text.StringBuilder()
  Dim expr2 As [String] = expr.ToLower()

  nuevaExpr.Append("(")
  For pos As Integer = 0 To expr2.Length - 1
    Dim letra As Char = expr2(pos)
    For valida As Integer = 0 To validos.Length - 1
        If letra = validos(valida) Then
            nuevaExpr.Append(letra)
            Exit For
        End If
    Next
  Next
  nuevaExpr.Append(")")
  Return nuevaExpr.ToString()
End Function
```

C++

```cpp
//Retira caracteres inválidos. Pone la expresión entre paréntesis.
```

```
void Evaluar::TransformaExpresion(char *nuevaExpr, char *expr)
{
  char validos[50];
  strcpy(validos, "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()");

  //Convierte a minúsculas
  for (int cont=0; cont<strlen(expr); cont++)
     if (expr[cont] >= 'A' && expr[cont] <= 'Z')
         expr[cont] = expr[cont] - 'A' + 'a';
  strcpy(nuevaExpr, "(");
  int posnuevaExpr = 1;
  for(int pos = 0; *(expr+pos); pos++)
  {
    char letra = expr[pos];
    for(int valida = 0; *(validos+valida); valida++)
      if (letra == validos[valida])
      {
        nuevaExpr[posnuevaExpr++] = letra;
        break;
      }
  }
  nuevaExpr[posnuevaExpr]=')';
  nuevaExpr[posnuevaExpr+1]='\0';
}
```

Object Pascal

```
//Retira los espacios y caracteres inválidos.
function TEvaluar.TransformaExpresion(expr: string): string;
var
  validos, nuevaExpr, expr2: string;
  pos, valida: integer;
  letra: char;
begin
  validos := 'abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()';
  expr2 := lowercase(expr);
  nuevaExpr := '(';
  for pos := 1 to length(expr2) do
  begin
    letra := expr2[pos];
    for valida := 1 to length(validos) do
    begin
      if letra = validos[valida] then
      begin
          nuevaExpr := nuevaExpr + letra;
          break;
      end;
    end;
  end;
  nuevaExpr := nuevaExpr + ')';
  Result := nuevaExpr;
end;
```

JavaScript

```
  //Retira caracteres inválidos. Pone la expresión entre paréntesis.
  this.TransformaExpresion = function(expr)
  {
    var validos = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
    var expr2 = expr.toLowerCase();
    var nuevaExpr = "(";
    for(var pos = 0; pos < expr2.length; pos++)
    {
      var letra = expr2.charAt(pos);
      for(var valida = 0; valida < validos.length; valida++)
        if (letra == validos.charAt(valida))
        {
          nuevaExpr += letra;
          break;
        }
    }
    nuevaExpr += ')';
    return nuevaExpr;
  }
```

PHP

```
  //Retira caracteres inválidos. Pone la expresión entre paréntesis.
  public function TransformaExpresion($expr)
  {
    $validos = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
    $expr2 = strtolower($expr);
    $nuevaExpr = "(";
    for($pos = 0; $pos < strlen($expr2); $pos++)
    {
      $letra = $expr2{$pos};
```

```
      for($valida = 0; $valida < strlen($validos); $valida++)
        if ($letra == $validos{$valida})
        {
          $nuevaExpr .= $letra;
          break;
        }
    }
    $nuevaExpr .= ')';
    return $nuevaExpr;
  }
```

Validación de Sintaxis

Llamado a las diferentes pruebas.

El siguiente paso es validar la sintaxis de la expresión, para eso se hacen 21 pruebas empezando por la más sencilla que es validar que no sea cadena vacía. Cada validación a excepción de cadena vacía es una función aislada que retorna verdadero(true) si detecta el error y falso(false) si no detecta ese error. El método EvaluaSintaxis es el encargado de llamar a todas las pruebas y retorna el valor 0 si la cadena pasa todas las pruebas de sintaxis o un número entero entre 1 y 21 si detecta algún error.

Java

```
  //Valida la expresión algebraica
  public final int EvaluaSintaxis(String expresion)
  {
    //Hace 25 pruebas de sintaxis
    if (DobleTripleOperadorSeguido(expresion)) return 1;
    if (OperadorParentesisCierra(expresion)) return 2;
    if (ParentesisAbreOperador(expresion)) return 3;
    if (ParentesisDesbalanceados(expresion)) return 4;
    if (ParentesisVacio(expresion)) return 5;
    if (ParentesisBalanceIncorrecto(expresion)) return 6;
    if (ParentesisCierraNumero(expresion)) return 7;
    if (NumeroParentesisAbre(expresion)) return 8;
    if (DoblePuntoNumero(expresion)) return 9;
    if (ParentesisCierraVariable(expresion)) return 10;
    if (VariableluegoPunto(expresion)) return 11;
    if (PuntoluegoVariable(expresion)) return 12;
    if (NumeroAntesVariable(expresion)) return 13;
    if (VariableDespuesNumero(expresion)) return 14;
    if (Chequea4letras(expresion)) return 15;
    if (FuncionInvalida(expresion)) return 16;
    if (VariableInvalida(expresion)) return 17;
    if (VariableParentesisAbre(expresion)) return 18;
    if (ParCierraParAbre(expresion)) return 19;
    if (OperadorPunto(expresion)) return 20;
    if (ParAbrePunto(expresion)) return 21;
    if (PuntoParAbre(expresion)) return 22;
    if (ParCierraPunto(expresion)) return 23;
    if (PuntoOperador(expresion)) return 24;
    if (PuntoParCierra(expresion)) return 25;

    return 0; //No se detectó error de sintaxis
  }
```

C#

```
  //Valida la expresión algebraica
  public int EvaluaSintaxis(String expresion)
  {
    //Hace 25 pruebas de sintaxis
    if (DobleTripleOperadorSeguido(expresion)) return 1;
    if (OperadorParentesisCierra(expresion)) return 2;
    if (ParentesisAbreOperador(expresion)) return 3;
    if (ParentesisDesbalanceados(expresion)) return 4;
    if (ParentesisVacio(expresion)) return 5;
    if (ParentesisBalanceIncorrecto(expresion)) return 6;
    if (ParentesisCierraNumero(expresion)) return 7;
    if (NumeroParentesisAbre(expresion)) return 8;
    if (DoblePuntoNumero(expresion)) return 9;
    if (ParentesisCierraVariable(expresion)) return 10;
    if (VariableluegoPunto(expresion)) return 11;
    if (PuntoluegoVariable(expresion)) return 12;
    if (NumeroAntesVariable(expresion)) return 13;
    if (VariableDespuesNumero(expresion)) return 14;
    if (Chequea4letras(expresion)) return 15;
    if (FuncionInvalida(expresion)) return 16;
    if (VariableInvalida(expresion)) return 17;
    if (VariableParentesisAbre(expresion)) return 18;
    if (ParCierraParAbre(expresion)) return 19;
    if (OperadorPunto(expresion)) return 20;
    if (ParAbrePunto(expresion)) return 21;
    if (PuntoParAbre(expresion)) return 22;
    if (ParCierraPunto(expresion)) return 23;
    if (PuntoOperador(expresion)) return 24;
    if (PuntoParCierra(expresion)) return 25;

    return 0; //No se detectó error de sintaxis
  }
```

Visual Basic .NET

```
'Valida la expresión algebraica
Public Function EvaluaSintaxis(expresion As [String]) As Integer
  'Hace 25 pruebas de sintaxis
  If DobleTripleOperadorSeguido(expresion) Then
    Return 1
  End If
  If OperadorParentesisCierra(expresion) Then
```

```
    Return 2
  End If
  If ParentesisAbreOperador(expresion) Then
    Return 3
  End If
  If ParentesisDesbalanceados(expresion) Then
    Return 4
  End If
  If ParentesisVacio(expresion) Then
    Return 5
  End If
  If ParentesisBalanceIncorrecto(expresion) Then
    Return 6
  End If
  If ParentesisCierraNumero(expresion) Then
    Return 7
  End If
  If NumeroParentesisAbre(expresion) Then
    Return 8
  End If
  If DoblePuntoNumero(expresion) Then
    Return 9
  End If
  If ParentesisCierraVariable(expresion) Then
    Return 10
  End If
  If VariableluegoPunto(expresion) Then
    Return 11
  End If
  If PuntoluegoVariable(expresion) Then
    Return 12
  End If
  If NumeroAntesVariable(expresion) Then
    Return 13
  End If
  If VariableDespuesNumero(expresion) Then
    Return 14
  End If
  If Chequea4letras(expresion) Then
    Return 15
  End If
  If FuncionInvalida(expresion) Then
    Return 16
  End If
  If VariableInvalida(expresion) Then
    Return 17
  End If
  If VariableParentesisAbre(expresion) Then
    Return 18
  End If
  If ParCierraParAbre(expresion) Then
    Return 19
  End If
  If OperadorPunto(expresion) Then
    Return 20
  End If
  If ParAbrePunto(expresion) Then
    Return 21
  End If
  If PuntoParAbre(expresion) Then
    Return 22
  End If
  If ParCierraPunto(expresion) Then
    Return 23
  End If
  If PuntoOperador(expresion) Then
    Return 24
  End If
  If PuntoParCierra(expresion) Then
    Return 25
  End If

  Return 0
  'No se detectó error de sintaxis
End Function
```

C++

```
//Valida la expresión algebraica
int Evaluar::EvaluaSintaxis(char *expresion)
{
    //Hace 25 pruebas de sintaxis
    if (DobleTripleOperadorSeguido(expresion)) return 1;
    if (OperadorParentesisCierra(expresion)) return 2;
    if (ParentesisAbreOperador(expresion)) return 3;
    if (ParentesisDesbalanceados(expresion)) return 4;
    if (ParentesisVacio(expresion)) return 5;
    if (ParentesisBalanceIncorrecto(expresion)) return 6;
    if (ParentesisCierraNumero(expresion)) return 7;
    if (NumeroParentesisAbre(expresion)) return 8;
```

```
    if (DoblePuntoNumero(expresion)) return 9;
    if (ParentesisCierraVariable(expresion)) return 10;
    if (VariableluegoPunto(expresion)) return 11;
    if (PuntoluegoVariable(expresion)) return 12;
    if (NumeroAntesVariable(expresion)) return 13;
    if (VariableDespuesNumero(expresion)) return 14;
    if (Chequea4letras(expresion)) return 15;
    if (FuncionInvalida(expresion)) return 16;
    if (VariableInvalida(expresion)) return 17;
    if (VariableParentesisAbre(expresion)) return 18;
    if (ParCierraParAbre(expresion)) return 19;
    if (OperadorPunto(expresion)) return 20;
    if (ParAbrePunto(expresion)) return 21;
    if (PuntoParAbre(expresion)) return 22;
    if (ParCierraPunto(expresion)) return 23;
    if (PuntoOperador(expresion)) return 24;
    if (PuntoParCierra(expresion)) return 25;

    //No se detectó error de sintaxis
    return 0;
}
```

Object Pascal

```
//Valida la expresión algebraica
function TEvaluar.EvaluaSintaxis(expresion: string): integer;
begin
  //Hace 25 pruebas de sintaxis
  if DobleTripleOperadorSeguido(expresion)then begin Result :=  1; exit; end;
  if OperadorParentesisCierra(expresion)then begin Result :=  2; exit; end;
  if ParentesisAbreOperador(expresion)then begin Result :=  3; exit; end;
  if ParentesisDesbalanceados(expresion)then begin Result :=  4; exit; end;
  if ParentesisVacio(expresion)then begin Result :=  5; exit; end;
  if ParentesisBalanceIncorrecto(expresion)then begin Result :=  6; exit; end;
  if ParentesisCierraNumero(expresion)then begin Result :=  7; exit; end;
  if NumeroParentesisAbre(expresion)then begin Result :=  8; exit; end;
  if DoblePuntoNumero(expresion)then begin Result :=  9; exit; end;
  if ParentesisCierraVariable(expresion)then begin Result :=  10; exit; end;
  if VariableluegoPunto(expresion)then begin Result :=  11; exit; end;
  if PuntoluegoVariable(expresion)then begin Result :=  12; exit; end;
  if NumeroAntesVariable(expresion)then begin Result :=  13; exit; end;
  if VariableDespuesNumero(expresion)then begin Result :=  14; exit; end;
  if Chequea4letras(expresion)then begin Result :=  15; exit; end;
  if FuncionInvalida(expresion)then begin Result :=  16; exit; end;
  if VariableInvalida(expresion)then begin Result :=  17; exit; end;
  if VariableParentesisAbre(expresion)then begin Result :=  18; exit; end;
  if ParCierraParAbre(expresion)then begin Result :=  19; exit; end;
  if OperadorPunto(expresion)then begin Result :=  20; exit; end;
  if ParAbrePunto(expresion)then begin Result :=  21; exit; end;
  if PuntoParAbre(expresion)then begin Result :=  22; exit; end;
  if ParCierraPunto(expresion)then begin Result :=  23; exit; end;
  if PuntoOperador(expresion)then begin Result :=  24; exit; end;
  if PuntoParCierra(expresion)then begin Result :=  25; exit; end;
  Result := 0; //No se detectó error de sintaxis
end;
```

JavaScript

```
//Valida la expresión algebraica
this.EvaluaSintaxis = function(expresion)
{
  //Hace 25 pruebas de sintaxis
  if (this.DobleTripleOperadorSeguido(expresion)) return 1;
  if (this.OperadorParentesisCierra(expresion)) return 2;
  if (this.ParentesisAbreOperador(expresion)) return 3;
  if (this.ParentesisDesbalanceados(expresion)) return 4;
  if (this.ParentesisVacio(expresion)) return 5;
  if (this.ParentesisBalanceIncorrecto(expresion)) return 6;
  if (this.ParentesisCierraNumero(expresion)) return 7;
  if (this.NumeroParentesisAbre(expresion)) return 8;
  if (this.DoblePuntoNumero(expresion)) return 9;
  if (this.ParentesisCierraVariable(expresion)) return 10;
  if (this.VariableluegoPunto(expresion)) return 11;
  if (this.PuntoluegoVariable(expresion)) return 12;
  if (this.NumeroAntesVariable(expresion)) return 13;
  if (this.VariableDespuesNumero(expresion)) return 14;
  if (this.Chequea4letras(expresion)) return 15;
  if (this.FuncionInvalida(expresion)) return 16;
  if (this.VariableInvalida(expresion)) return 17;
  if (this.VariableParentesisAbre(expresion)) return 18;
  if (this.ParCierraParAbre(expresion)) return 19;
  if (this.OperadorPunto(expresion)) return 20;
  if (this.ParAbrePunto(expresion)) return 21;
  if (this.PuntoParAbre(expresion)) return 22;
  if (this.ParCierraPunto(expresion)) return 23;
  if (this.PuntoOperador(expresion)) return 24;
  if (this.PuntoParCierra(expresion)) return 25;

  return 0; //No se detectó error de sintaxis
```

```
}
```

PHP

```php
//Valida la expresión algebraica
public function EvaluaSintaxis($expresion)
{
  //Hace 25 pruebas de sintaxis
  if ($this->DobleTripleOperadorSeguido($expresion)) return 1;
  if ($this->OperadorParentesisCierra($expresion)) return 2;
  if ($this->ParentesisAbreOperador($expresion)) return 3;
  if ($this->ParentesisDesbalanceados($expresion)) return 4;
  if ($this->ParentesisVacio($expresion)) return 5;
  if ($this->ParentesisBalanceIncorrecto($expresion)) return 6;
  if ($this->ParentesisCierraNumero($expresion)) return 7;
  if ($this->NumeroParentesisAbre($expresion)) return 8;
  if ($this->DoblePuntoNumero($expresion)) return 9;
  if ($this->ParentesisCierraVariable($expresion)) return 10;
  if ($this->VariableluegoPunto($expresion)) return 11;
  if ($this->PuntoluegoVariable($expresion)) return 12;
  if ($this->NumeroAntesVariable($expresion)) return 13;
  if ($this->VariableDespuesNumero($expresion)) return 14;
  if ($this->Chequea4letras($expresion)) return 15;
  if ($this->FuncionInvalida($expresion)) return 16;
  if ($this->VariableInvalida($expresion)) return 17;
  if ($this->VariableParentesisAbre($expresion)) return 18;
  if ($this->ParCierraParAbre($expresion)) return 19;
  if ($this->OperadorPunto($expresion)) return 20;
  if ($this->ParAbrePunto($expresion)) return 21;
  if ($this->PuntoParAbre($expresion)) return 22;
  if ($this->ParCierraPunto($expresion)) return 23;
  if ($this->PuntoOperador($expresion)) return 24;
  if ($this->PuntoParCierra($expresion)) return 25;

  return 0; //No se detectó error de sintaxis
}
```

Validación de Sintaxis. Mensaje para cada prueba.

Si la validación de sintaxis presenta una falla, este método retorna en texto en que consiste la falla para que pueda ser leída por el usuario final. Usted como desarrollador puede cambiar las frases escritas en este método.

Java

```
//Muestra mensaje de error sintáctico
public final String MensajeSintaxis(int CodigoError)
{
  switch(CodigoError)
  {
    case 0:  return "No se detectó error sintáctico en las 25 pruebas que se hicieron.";
    case 1:  return "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3";
    case 2:  return "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7";
    case 3:  return "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)";
    case 4:  return "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))";
    case 5:  return "5. Que haya paréntesis vacío. Ejemplo: 2-()*3";
    case 6:  return "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4";
    case 7: return "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)";
    case 8: return "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)";
    case 9: return "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2";
    case 10: return "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1";
    case 11: return "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3";
    case 12: return "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1";
    case 13: return "13. Un número antes de una variable. Ejemplo: 3x+1";
    case 14: return "14. Un número después de una variable. Ejemplo: x21+4";
    case 15: return "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9";
    case 16: return "16. Función inexistente. Ejemplo: 5*alo(78)";
    case 17: return "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5";
    case 18: return "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)";
    case 19: return "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)";
    case 20: return "20. Después de operador sigue un punto. Ejemplo: -.3+7";
    case 21: return "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)";
    case 22: return "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)";
    case 23: return "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2";
    case 24: return "24. Punto seguido de operador. Ejemplo: 5.*9+1";
    default: return "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5";
  }
}
```

C#

```
//Muestra mensaje de error sintáctico
public String MensajeSintaxis(int CodigoError)
{
  switch(CodigoError)
  {
    case 0:  return "No se detectó error sintáctico en las 21 pruebas que se hicieron.";
    case 1:  return "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3";
    case 2:  return "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7";
    case 3:  return "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)";
    case 4:  return "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))";
    case 5:  return "5. Que haya paréntesis vacío. Ejemplo: 2-()*3";
    case 6:  return "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4";
    case 7: return "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)";
    case 8: return "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)";
    case 9: return "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2";
    case 10: return "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1";
    case 11: return "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3";
    case 12: return "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1";
    case 13: return "13. Un número antes de una variable. Ejemplo: 3x+1";
    case 14: return "14. Un número después de una variable. Ejemplo: x21+4";
    case 15: return "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9";
    case 16: return "16. Función inexistente. Ejemplo: 5*alo(78)";
    case 17: return "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5";
    case 18: return "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)";
    case 19: return "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)";
    case 20: return "20. Después de operador sigue un punto. Ejemplo: -.3+7";
    case 21: return "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)";
    case 22: return "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)";
    case 23: return "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2";
    case 24: return "24. Punto seguido de operador. Ejemplo: 5.*9+1";
    default: return "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5";
  }
}
```

Visual Basic .NET

```
'Muestra mensaje de error sintáctico
Public Function MensajeSintaxis(CodigoError As Integer) As [String]
  Select Case CodigoError
    Case 0
      Return "No se detectó error sintáctico en las 21 pruebas que se hicieron."
    Case 1
      Return "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3"
    Case 2
      Return "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7"
    Case 3
```

```
    Return "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)"
  Case 4
    Return "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))"
  Case 5
    Return "5. Que haya paréntesis vacío. Ejemplo: 2-()*3"
  Case 6
    Return "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4"
  Case 7
    Return "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)"
  Case 8
    Return "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)"
  Case 9
    Return "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2"
  Case 10
    Return "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1"
  Case 11
    Return "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3"
  Case 12
    Return "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1"
  Case 13
    Return "13. Un número antes de una variable. Ejemplo: 3x+1"
  Case 14
    Return "14. Un número después de una variable. Ejemplo: x21+4"
  Case 15
    Return "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9"
  Case 16
    Return "16. Función inexistente. Ejemplo: 5*alo(78)"
  Case 17
    Return "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5"
  Case 18
    Return "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)"
  Case 19
    Return "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)"
  Case 20
    Return "20. Después de operador sigue un punto. Ejemplo: -.3+7"
  Case 21
    Return "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)"
  Case 22
    Return "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)"
  Case 23
    Return "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2"
  Case 24
    Return "24. Punto seguido de operador. Ejemplo: 5.*9+1"
  Case Else
    Return "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5"
 End Select
End Function
```

C++

```
//Muestra mensaje de error sintáctico
void Evaluar::MensajeSintaxis(int CodigoError, char *mensaje)
{
    switch(CodigoError)
    {
      case 0:  strcpy(mensaje, "No se detectó error sintáctico en las 25 pruebas que se hicieron."); break;
      case 1:  strcpy(mensaje, "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3"); break;
      case 2:  strcpy(mensaje, "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7"); break;
      case 3:  strcpy(mensaje, "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)"); break;
      case 4:  strcpy(mensaje, "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))"); break;
      case 5:  strcpy(mensaje, "5. Que haya paréntesis vacío. Ejemplo: 2-()*3"); break;
      case 6:  strcpy(mensaje, "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4"); break;
      case 7: strcpy(mensaje, "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)"); break;
      case 8: strcpy(mensaje, "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)"); break;
      case 9: strcpy(mensaje, "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2"); break;
      case 10: strcpy(mensaje, "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1"); break;
      case 11: strcpy(mensaje, "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3"); break;
      case 12: strcpy(mensaje, "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1"); break;
      case 13: strcpy(mensaje, "13. Un número antes de una variable. Ejemplo: 3x+1"); break;
      case 14: strcpy(mensaje, "14. Un número después de una variable. Ejemplo: x21+4"); break;
      case 15: strcpy(mensaje, "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9"); break;
      case 16: strcpy(mensaje, "16. Función inexistente. Ejemplo: 5*alo(78)"); break;
      case 17: strcpy(mensaje, "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5"); break;
      case 18: strcpy(mensaje, "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)"); break;
      case 19: strcpy(mensaje, "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)"); break;
      case 20: strcpy(mensaje, "20. Después de operador sigue un punto. Ejemplo: -.3+7"); break;
      case 21: strcpy(mensaje, "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)"); break;
      case 22: strcpy(mensaje, "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)"); break;
      case 23: strcpy(mensaje, "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2"); break;
      case 24: strcpy(mensaje, "24. Punto seguido de operador. Ejemplo: 5.*9+1"); break;
      default: strcpy(mensaje, "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5"); break;
    }
}
```

Object Pascal

```
//Muestra mensaje de error sintáctico
function TEvaluar.MensajeSintaxis(CodigoError: integer): string;
begin
```

```
  case CodigoError of
    0: begin  Result := 'No se detectó error sintáctico en las 25 pruebas que se hicieron.'; exit; end;
    1: begin  Result := '1. Dos o más operadores estén seguidos. Ejemplo: begin 2++4, 5-*3'; exit; end;
    2: begin  Result := '2. Un operador seguido de un paréntesis que cierra. Ejemplo: begin 2-(4+)-7'; exit; end;
    3: begin  Result := '3. Un paréntesis que abre seguido de un operador. Ejemplo: begin 2-(*3)'; exit; end;
    4: begin  Result := '4. Que los paréntesis estén desbalanceados. Ejemplo: begin 3-(2*4))'; exit; end;
    5: begin  Result := '5. Que haya paréntesis vacío. Ejemplo: begin 2-()*3'; exit; end;
    6: begin  Result := '6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: begin 2+3)-2*(4'; exit; end;
    7: begin Result := '7. Un paréntesis que cierra y sigue un número. Ejemplo: begin (3-5)7-(1+2)'; exit; end;
    8: begin Result := '8. Un número seguido de un paréntesis que abre. Ejemplo: begin 7-2(5-6)'; exit; end;
    9: begin Result := '9. Doble punto en un número de tipo real. Ejemplo: begin 3-2..4+1  7-6.46.1+2'; exit; end;
    10: begin Result := '10. Un paréntesis que cierra seguido de una variable. Ejemplo: begin (12-4)y-1'; exit; end;
    11: begin Result := '11. Una variable seguida de un punto. Ejemplo: begin 4-z.1+3'; exit; end;
    12: begin Result := '12. Un punto seguido de una variable. Ejemplo: begin 7-2.p+1'; exit; end;
    13: begin Result := '13. Un número antes de una variable. Ejemplo: begin 3x+1'; exit; end;
    14: begin Result := '14. Un número después de una variable. Ejemplo: begin x21+4'; exit; end;
    15: begin Result := '15. Hay 4 o más letras seguidas. Ejemplo: begin 12+ramp+8.9'; exit; end;
    16: begin Result := '16. Función inexistente. Ejemplo: begin 5*alo(78)'; exit; end;
    17: begin Result := '17. Variable inválida (solo pueden tener una letra). Ejemplo: begin 5+tr-xc+5'; exit; end;
    18: begin Result := '18. Variable seguida de paréntesis que abre. Ejemplo: begin 5-a(7+3)'; exit; end;
    19: begin Result := '19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: begin (4-5)(2*x)'; exit; end;
    20: begin Result := '20. Después de operador sigue un punto. Ejemplo: begin -.3+7'; exit; end;
    21: begin Result := '21. Después de paréntesis que abre sigue un punto. Ejemplo: begin 3*(.5+4)'; exit; end;
    22: begin Result := '22. Un punto seguido de un paréntesis que abre. Ejemplo: begin 7+3.(2+6)'; exit; end;
    23: begin Result := '23. Paréntesis cierra y sigue punto. Ejemplo: begin (4+5).7-2'; exit; end;
    24: begin Result := '24. Punto seguido de operador. Ejemplo: begin 5.*9+1'; exit; end;
    else begin Result := '25. Punto seguido de paréntesis que cierra. Ejemplo: begin (3+2.)*5'; exit; end;
  end;
end;
```

JavaScript

```
//Muestra mensaje de error sintáctico
this.MensajeSintaxis = function(CodigoError)
{
  switch(CodigoError)
  {
    case 0:  return "No se detectó error sintáctico en las 25 pruebas que se hicieron.";
    case 1:  return "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3";
    case 2:  return "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7";
    case 3:  return "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)";
    case 4:  return "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))";
    case 5:  return "5. Que haya paréntesis vacío. Ejemplo: 2-()*3";
    case 6:  return "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4";
    case 7: return "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)";
    case 8: return "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)";
    case 9: return "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2";
    case 10: return "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1";
    case 11: return "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3";
    case 12: return "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1";
    case 13: return "13. Un número antes de una variable. Ejemplo: 3x+1";
    case 14: return "14. Un número después de una variable. Ejemplo: x21+4";
    case 15: return "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9";
    case 16: return "16. Función inexistente. Ejemplo: 5*alo(78)";
    case 17: return "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5";
    case 18: return "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)";
    case 19: return "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)";
    case 20: return "20. Después de operador sigue un punto. Ejemplo: -.3+7";
    case 21: return "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)";
    case 22: return "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)";
    case 23: return "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2";
    case 24: return "24. Punto seguido de operador. Ejemplo: 5.*9+1";
    default: return "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5";
  }
}
```

PHP

```
//Muestra mensaje de error sintáctico
public function MensajeSintaxis($CodigoError)
{
  switch($CodigoError)
  {
    case 0:  return "No se detectó error sintáctico en las 25 pruebas que se hicieron.";
    case 1:  return "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3";
    case 2:  return "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7";
    case 3:  return "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)";
    case 4:  return "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))";
    case 5:  return "5. Que haya paréntesis vacío. Ejemplo: 2-()*3";
    case 6:  return "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4";
    case 7:  return "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)";
    case 8:  return "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)";
    case 9:  return "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2";
    case 10: return "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1";
    case 11: return "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3";
    case 12: return "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1";
    case 13: return "13. Un número antes de una variable. Ejemplo: 3x+1";
    case 14: return "14. Un número después de una variable. Ejemplo: x21+4";
    case 15: return "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9";
```

```
        case 16: return "16. Función inexistente. Ejemplo: 5*alo(78)";
        case 17: return "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5";
        case 18: return "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)";
        case 19: return "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)";
        case 20: return "20. Después de operador sigue un punto. Ejemplo: -.3+7";
        case 21: return "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)";
        case 22: return "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)";
        case 23: return "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2";
        case 24: return "24. Punto seguido de operador. Ejemplo: 5.*9+1";
        default: return "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5";
    }
}
```

Dos o más operadores estén seguidos

Si detecta expresiones por ejemplo 5+*/7 o 8++9 o 13*--5 el programa retorna verdadero(true) que es un error. Hay que tener en cuenta que con la nueva funcionalidad de menos unario, una expresión como 5*-7 ya está permitida.

Java

```java
//1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
private static boolean DobleTripleOperadorSeguido(String expr)
{
    for(int pos = 0; pos < expr.length() - 1; pos++)
    {
        char car1 = expr.charAt(pos);   //Extrae un carácter
        char car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^')
                return true;
    }

    for (int pos = 0; pos < expr.length() - 2; pos++)
    {
        char car1 = expr.charAt(pos);   //Extrae un carácter
        char car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter
        char car3 = expr.charAt(pos + 2); //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^')
                if (car3 == '+' || car3 == '-' || car3 == '*' || car3 == '/' || car3 == '^')
                    return true;
    }

    return false;  //No encontró doble/triple operador seguido
}
```

C#

```csharp
//1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
private static Boolean DobleTripleOperadorSeguido(String expr)
{
    for(int pos = 0; pos < expr.Length - 1; pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter
        char car2 = expr[pos + 1]; //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^')
                return true;
    }

    for (int pos = 0; pos < expr.Length - 2; pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter
        char car2 = expr[pos + 1]; //Extrae el siguiente carácter
        char car3 = expr[pos + 2]; //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^')
                if (car3 == '+' || car3 == '-' || car3 == '*' || car3 == '/' || car3 == '^')
                    return true;
    }

    return false;  //No encontró doble/triple operador seguido
}
```

Visual Basic .NET

```vb
'1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
Private Shared Function DobleTripleOperadorSeguido(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
        Dim car1 As Char = expr(pos)
        'Extrae un carácter
        Dim car2 As Char = expr(pos + 1)
        'Extrae el siguiente carácter
        'Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        If car1 = "+" OrElse car1 = "-" OrElse car1 = "*" OrElse car1 = "/" OrElse car1 = "^" Then
            If car2 = "+" OrElse car2 = "*" OrElse car2 = "/" OrElse car2 = "^" Then
                Return True
            End If
        End If
    Next

    For pos As Integer = 0 To expr.Length - 3
        Dim car1 As Char = expr(pos)
        'Extrae un carácter
```

```
            Dim car2 As Char = expr(pos + 1)
            'Extrae el siguiente carácter
            Dim car3 As Char = expr(pos + 2)
            'Extrae el siguiente carácter

            'Compara si el carácter y el siguiente son operadores, dado el caso retorna true
            If car1 = "+" OrElse car1 = "-" OrElse car1 = "*" OrElse car1 = "/" OrElse car1 = "^" Then
                If car2 = "+" OrElse car2 = "-" OrElse car2 = "*" OrElse car2 = "/" OrElse car2 = "^" Then
                    If car3 = "+" OrElse car3 = "-" OrElse car3 = "*" OrElse car3 = "/" OrElse car3 = "^" Then
                        Return True
                    End If
                End If
            End If
        Next

        Return False
        'No encontró doble/triple operador seguido
    End Function
```

C++

```
//1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
bool Evaluar::DobleTripleOperadorSeguido(char *expr)
{
    for (int pos=0; *(expr+pos+1); pos++)
    {
        char car1 = expr[pos];  //Extrae un carácter
        char car2 = expr[pos+1]; //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^')
                return true;
    }

    for (int pos=0; *(expr+pos+2); pos++)
    {
        char car1 = expr[pos];  //Extrae un carácter
        char car2 = expr[pos+1]; //Extrae el siguiente carácter
        char car3 = expr[pos+2]; //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^')
                if (car3 == '+' || car3 == '-' || car3 == '*' || car3 == '/' || car3 == '^')
                    return true;
    }
    return false; //No encontró doble/triple operador seguido
}
```

Object Pascal

```
//1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
function TEvaluar.DobleTripleOperadorSeguido(expr: string): boolean;
var
  pos: integer;
  car1, car2, car3: char;
begin
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];   //Extrae un carácter
    car2 := expr[pos + 1]; //Extrae el siguiente carácter

    //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
    if (car1 = '+') or (car1 = '-') or (car1 = '*') or (car1 = '/') or (car1 = '^') then
      if (car2 = '+') or (car2 = '*') or (car2 = '/') or (car2 = '^') then
      begin
        Result := true;
        exit;
      end;
  end;

  for pos := 1 to length(expr) - 1 do
  begin
    car1 := expr[pos];   //Extrae un carácter
    car2 := expr[pos + 1]; //Extrae el siguiente carácter
    car3 := expr[pos + 2]; //Extrae el siguiente carácter

    //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
    if (car1 = '+') or (car1 = '-') or (car1 = '*') or (car1 = '/') or (car1 = '^') then
      if (car2 = '+') or (car2 = '-') or (car2 = '*') or (car2 = '/') or (car2 = '^') then
        if (car3 = '+') or (car3 = '-') or (car3 = '*') or (car3 = '/') or (car3 = '^') then
        begin
          Result := true;
          exit;
        end;
  end;

  Result := false;  //No encontró doble/triple operador seguido
end;
```

JavaScript

```
//1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
this.DobleTripleOperadorSeguido = function(expr)
{
    for(var pos = 0; pos < expr.length - 1; pos++)
    {
        var car1 = expr.charAt(pos);   //Extrae un carácter
        var car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^')
                return true;
    }

    for (var pos = 0; pos < expr.length - 2; pos++)
    {
        var car1 = expr.charAt(pos);   //Extrae un carácter
        var car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter
        var car3 = expr.charAt(pos + 2); //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^')
                if (car3 == '+' || car3 == '-' || car3 == '*' || car3 == '/' || car3 == '^')
                    return true;
    }

    return false;  //No encontró doble/triple operador seguido
}
```

PHP

```
//1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
function DobleTripleOperadorSeguido($expr)
{
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if ($car1 == '+' || $car1 == '-' || $car1 == '*' || $car1 == '/' || $car1 == '^')
            if ($car2 == '+' || $car2 == '*' || $car2 == '/' || $car2 == '^')
                return true;
    }

    for ($pos = 0; $pos < strlen($expr) - 2; $pos++)
    {
        $car1 = $expr{$pos};   //Extrae un carácter
        $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter
        $car3 = $expr{$pos + 2}; //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if ($car1 == '+' || $car1 == '-' || $car1 == '*' || $car1 == '/' || $car1 == '^')
            if ($car2 == '+' || $car2 == '-' || $car2 == '*' || $car2 == '/' || $car2 == '^')
                if ($car3 == '+' || $car3 == '-' || $car3 == '*' || $car3 == '/' || $car3 == '^')
                    return true;
    }

    return false;  //No encontró doble/triple operador seguido
}
```

Un operador seguido de un paréntesis que cierra

Si hay una cadena como 9+(7^)*3 es un error, después de un operador (+, -, *, /, ^) no debe haber un paréntesis que cierra.

Java

```java
    //2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
    private static boolean OperadorParentesisCierra(String expr)
    {
        for(int pos = 0; pos < expr.length() - 1; pos++)
        {
            char car1 = expr.charAt(pos);   //Extrae un carácter

            //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
            if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
                if (expr.charAt(pos + 1) == ')') return true;
        }
        return false; //No encontró operador seguido de un paréntesis que cierra
    }
```

C#

```csharp
    //2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
    private static Boolean OperadorParentesisCierra(String expr)
    {
        for(int pos = 0; pos < expr.Length - 1; pos++)
        {
            char car1 = expr[pos];   //Extrae un carácter

            //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
            if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
                if (expr[pos + 1] == ')') return true;
        }
        return false; //No encontró operador seguido de un paréntesis que cierra
    }
```

Visual Basic .NET

```vb
    '2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
    Private Shared Function OperadorParentesisCierra(expr As [String]) As [Boolean]
        For pos As Integer = 0 To expr.Length - 2
            Dim car1 As Char = expr(pos)
            'Extrae un carácter
            'Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
            If car1 = "+" OrElse car1 = "-" OrElse car1 = "*" OrElse car1 = "/" OrElse car1 = "^" Then
                If expr(pos + 1) = ")" Then
                    Return True
                End If
            End If
        Next
        Return False
        'No encontró operador seguido de un paréntesis que cierra
    End Function
```

C++

```cpp
//2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
bool Evaluar::OperadorParentesisCierra(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter

        //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (expr[pos+1] == ')') return true;
    }
    return false; //No encontró operador seguido de un paréntesis que cierra
}
```

Object Pascal

```
//2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
function TEvaluar.OperadorParentesisCierra(expr: string): boolean;
var
  pos: integer;
  car1: char;
begin
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];   //Extrae un carácter

    //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
    if (car1 = '+') or (car1 = '-') or (car1 = '*') or (car1 = '/') or (car1 = '^') then
      if (expr[pos + 1] = ')') then
        begin
          Result := true;
          exit;
        end;
  end;
```

```
  Result := false; //No encontró operador seguido de un paréntesis que cierra
end;
```

JavaScript

```
    //2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
    this.OperadorParentesisCierra = function(expr)
    {
        for(var pos = 0; pos < expr.length - 1; pos++)
        {
            var car1 = expr.charAt(pos);   //Extrae un carácter

            //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
            if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
                if (expr.charAt(pos + 1) == ')') return true;
        }
        return false; //No encontró operador seguido de un paréntesis que cierra
    }
```

PHP

```
    //2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
    function OperadorParentesisCierra($expr)
    {
        for($pos = 0; $pos < strlen($expr) - 1; $pos++)
        {
            $car1 = $expr{$pos};   //Extrae un carácter

            //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
            if ($car1 == '+' || $car1 == '-' || $car1 == '*' || $car1 == '/' || $car1 == '^')
                if ($expr{$pos + 1} == ')') return true;
        }
        return false; //No encontró operador seguido de un paréntesis que cierra
    }
```

Un paréntesis que abre seguido de un operador

Una expresión como 5-(*5/7) es errónea, luego se valida que después de un paréntesis que abre no siga un operador a excepción del menos "-" porque en esta versión si está permitido expresiones como 8*(-3+2)

Java

```java
    //3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
    private static boolean ParentesisAbreOperador(String expr)
    {
        for(int pos = 0; pos < expr.length() - 1; pos++)
        {
            char car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
            if (expr.charAt(pos) == '(')
                if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^') return true;
        }
        return false;  //No encontró paréntesis que abre seguido de un operador
    }
```

C#

```c#
    //3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
    private static Boolean ParentesisAbreOperador(String expr)
    {
        for(int pos = 0; pos < expr.Length - 1; pos++)
        {
            char car2 = expr[pos + 1]; //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
            if (expr[pos] == '(')
                if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^') return true;
        }
        return false;  //No encontró paréntesis que abre seguido de un operador
    }
```

Visual Basic .NET

```vb
    '3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
    Private Shared Function ParentesisAbreOperador(expr As [String]) As [Boolean]
        For pos As Integer = 0 To expr.Length - 2
            Dim car2 As Char = expr(pos + 1)
            'Extrae el siguiente carácter
            'Compara si el primer carácter es paréntesis que abre y el siguiente es operador
            If expr(pos) = "(" Then
                If car2 = "+" OrElse car2 = "*" OrElse car2 = "/" OrElse car2 = "^" Then
                    Return True
                End If
            End If
        Next
        Return False
        'No encontró paréntesis que abre seguido de un operador
    End Function
```

C++

```c++
//3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
bool Evaluar::ParentesisAbreOperador(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
    {
        char car2 = expr[pos+1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
        if (expr[pos] == '(')
            if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^') return true;
    }
    return false; //No encontró paréntesis que abre seguido de un operador
}
```

Object Pascal

```
//3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
function TEvaluar.ParentesisAbreOperador(expr: string): boolean;
var
  pos: integer;
  car2: char;
begin
  for pos := 1 to length(expr) do
  begin
    car2 := expr[pos + 1]; //Extrae el siguiente carácter

    //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
    if expr[pos] = '(' then
```

```
      if (car2 = '+') or (car2 = '*') or (car2 = '/') or (car2 = '^') then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;  //No encontró paréntesis que abre seguido de un operador
end;
```

JavaScript

```
    //3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
    this.ParentesisAbreOperador = function(expr)
    {
        for(var pos = 0; pos < expr.length - 1; pos++)
        {
            var car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
            if (expr.charAt(pos) == '(')
                if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^') return true;
        }
        return false;  //No encontró paréntesis que abre seguido de un operador
    }
```

PHP

```
    //3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
    function ParentesisAbreOperador($expr)
    {
        for($pos = 0; $pos < strlen($expr) - 1; $pos++)
        {
            $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

            //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
            if ($expr{$pos} == '(')
                if ($car2 == '+' || $car2 == '*' || $car2 == '/' || $car2 == '^') return true;
        }
        return false;  //No encontró paréntesis que abre seguido de un operador
    }
```

Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))

El número de paréntesis que abre debe ser igual al número de paréntesis que cierra.

Java

```java
    //4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
    private static boolean ParentesisDesbalanceados(String expr)
    {
        int parabre = 0, parcierra = 0;
        for(int pos = 0; pos < expr.length(); pos++)
        {
            char car1 = expr.charAt(pos);
            if (car1 == '(') parabre++;
            if (car1 == ')') parcierra++;
        }
        return parabre != parcierra;
    }
```

C#

```csharp
    //4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
    private static Boolean ParentesisDesbalanceados(String expr)
    {
        int parabre = 0, parcierra = 0;
        for(int pos = 0; pos < expr.Length; pos++)
        {
            char car1 = expr[pos];
            if (car1 == '(') parabre++;
            if (car1 == ')') parcierra++;
        }
        return parabre != parcierra;
    }
```

Visual Basic .NET

```vbnet
    '4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
    Private Shared Function ParentesisDesbalanceados(expr As [String]) As [Boolean]
        Dim parabre As Integer = 0, parcierra As Integer = 0
        For pos As Integer = 0 To expr.Length - 1
            Dim car1 As Char = expr(pos)
            If car1 = "(" Then
                parabre += 1
            End If
            If car1 = ")" Then
                parcierra += 1
            End If
        Next
        Return parabre <> parcierra
    End Function
```

C++

```cpp
//4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
bool Evaluar::ParentesisDesbalanceados(char *expr)
{
    int parabre = 0, parcierra = 0;
    for(int pos = 0; *(expr+pos); pos++)
    {
        char car1 = expr[pos];
        if (car1 == '(') parabre++;
        if (car1 == ')') parcierra++;
    }
    return (parabre != parcierra);
}
```

Object Pascal

```
//4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
function TEvaluar.ParentesisDesbalanceados(expr: string): boolean;
var
  parabre, parcierra, pos: integer;
  car1: char;
begin
  parabre := 0; parcierra := 0;
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];
    if car1 = '(' then Inc(parabre);
    if car1 = ')' then Inc(parcierra);
  end;
  if parabre <> parcierra then
    Result := true
  else
    Result := false;
end;
```

JavaScript

```
//4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
this.ParentesisDesbalanceados = function(expr)
{
    var parabre = 0, parcierra = 0;
    for(var pos = 0; pos < expr.length; pos++)
    {
        var car1 = expr.charAt(pos);
        if (car1 == '(') parabre++;
        if (car1 == ')') parcierra++;
    }
    return parabre != parcierra;
}
```

PHP

```
//4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
function ParentesisDesbalanceados($expr)
{
    $parabre = 0; $parcierra = 0;
    for($pos = 0; $pos < strlen($expr); $pos++)
    {
        $car1 = $expr{$pos};
        if ($car1 == '(') $parabre++;
        if ($car1 == ')') $parcierra++;
    }
    return $parabre != $parcierra;
}
```

Que haya paréntesis vacío. Ejemplo: 2-()*3

Siempre debe haber algo entre un paréntesis que abre y cierra

Java

```java
    //5. Que haya paréntesis vacío. Ejemplo: 2-()*3
    private static boolean ParentesisVacio(String expr)
    {
        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        for(int pos = 0; pos < expr.length() - 1; pos++)
            if (expr.charAt(pos) == '(' && expr.charAt(pos + 1) == ')') return true;
        return false;
    }
```

C#

```csharp
    //5. Que haya paréntesis vacío. Ejemplo: 2-()*3
    private static Boolean ParentesisVacio(String expr)
    {
        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        for(int pos = 0; pos < expr.Length - 1; pos++)
            if (expr[pos] == '(' && expr[pos + 1] == ')') return true;
        return false;
    }
```

Visual Basic .NET

```vbnet
    '5. Que haya paréntesis vacío. Ejemplo: 2-()*3
    Private Shared Function ParentesisVacio(expr As [String]) As [Boolean]
        'Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        For pos As Integer = 0 To expr.Length - 2
            If expr(pos) = "(" AndAlso expr(pos + 1) = ")" Then
                Return True
            End If
        Next
        Return False
    End Function
```

C++

```cpp
//5.Que haya paréntesis vacío. Ejemplo: 2-()*3
bool Evaluar::ParentesisVacio(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] == '(' && expr[pos+1] == ')') return true;
    return false;
}
```

Object Pascal

```
//5. Que haya paréntesis vacío. Ejemplo: 2-()*3
function TEvaluar.ParentesisVacio(expr: string): boolean;
var
  pos: integer;
begin
  //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] = '(') and (expr[pos + 1] = ')') then
    begin
      Result := true;
      Exit;
    end;
  end;
  Result := false;
end;
```

JavaScript

```javascript
    //5. Que haya paréntesis vacío. Ejemplo: 2-()*3
    this.ParentesisVacio = function(expr)
    {
        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        for(var pos = 0; pos < expr.length - 1; pos++)
            if (expr.charAt(pos) == '(' && expr.charAt(pos + 1) == ')') return true;
        return false;
    }
```

PHP

```php
    //5. Que haya paréntesis vacío. Ejemplo: 2-()*3
```

```php
function ParentesisVacio($expr)
{
    //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
        if ($expr{$pos} == '(' && $expr{$pos + 1} == ')') return true;
    return false;
}
```

Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4

Puede que el número de paréntesis que abre y cierra sea el mismo, pero no exista correspondencia

Java

```java
//6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
private static boolean ParentesisBalanceIncorrecto(String expr)
{
    int balance = 0;
    for(int pos = 0; pos < expr.length(); pos++)
    {
        char car1 = expr.charAt(pos);   //Extrae un carácter
        if (car1 == '(') balance++;
        if (car1 == ')') balance--;
        if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
    }
    return false;
}
```

C#

```csharp
//6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
private static Boolean ParentesisBalanceIncorrecto(String expr)
{
    int balance = 0;
    for(int pos = 0; pos < expr.Length; pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter
        if (car1 == '(') balance++;
        if (car1 == ')') balance--;
        if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
    }
    return false;
}
```

Visual Basic .NET

```vbnet
'6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
Private Shared Function ParentesisBalanceIncorrecto(expr As [String]) As [Boolean]
    Dim balance As Integer = 0
    For pos As Integer = 0 To expr.Length - 1
        Dim car1 As Char = expr(pos)
        'Extrae un carácter
        If car1 = "(" Then
            balance += 1
        End If
        If car1 = ")" Then
            balance -= 1
        End If
        If balance < 0 Then
            Return True
            'Si cae por debajo de cero es que el balance es erróneo
        End If
    Next
    Return False
End Function
```

C++

```cpp
//6.Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
bool Evaluar::ParentesisBalanceIncorrecto(char *expr)
{
    int balance = 0;
    for(int pos = 0; *(expr+pos); pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter
        if (car1 == '(') balance++;
        if (car1 == ')') balance--;
        if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
    }
    return false;
}
```

Object Pascal

```
//6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
function TEvaluar.ParentesisBalanceIncorrecto(expr: string): boolean;
var
  balance, pos: integer;
  car1: char;
begin
  balance := 0;
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];   //Extrae un carácter
```

```
    if car1 = '(' then Inc(balance);
    if car1 = ')' then Dec(balance);
    if balance < 0 then begin Result := true; Exit; end;
  end;
  Result := false;
end;
```

JavaScript

```
    //6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
    this.ParentesisBalanceIncorrecto = function(expr)
    {
        var balance = 0;
        for(var pos = 0; pos < expr.length; pos++)
        {
            var car1 = expr.charAt(pos);   //Extrae un carácter
            if (car1 == '(') balance++;
            if (car1 == ')') balance--;
            if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
        }
        return false;
    }
```

PHP

```
    //6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
    function ParentesisBalanceIncorrecto($expr)
    {
        $balance = 0;
        for($pos = 0; $pos < strlen($expr); $pos++)
        {
            $car1 = $expr{$pos};   //Extrae un carácter
            if ($car1 == '(') $balance++;
            if ($car1 == ')') $balance--;
            if ($balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
        }
        return false;
    }
```

Un paréntesis que cierra seguido de un número o paréntesis que abre

Después de un paréntesis que cierra, sólo sigue operadores, otros paréntesis que cierran o es el fin de la cadena

Java

```java
//7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)
private static boolean ParentesisCierraNumero(String expr)
{
  for(int pos = 0; pos < expr.length() - 1; pos++)
  {
    char car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

    //Compara si el primer carácter es paréntesis que cierra y el siguiente es número
    if (expr.charAt(pos) == ')')
      if (car2 >= '0' && car2 <= '9') return true;
  }
  return false;
}
```

C#

```csharp
//7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
private static Boolean ParentesisCierraNumero(String expr)
{
  for(int pos = 0; pos < expr.Length - 1; pos++)
  {
    char car2 = expr[pos + 1]; //Extrae el siguiente carácter

    //Compara si el primer carácter es paréntesis que cierra y el siguiente es número
    if (expr[pos] == ')')
      if (car2 >= '0' && car2 <= '9') return true;
  }
  return false;
}
```

Visual Basic .NET

```vb
'7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
Private Shared Function ParentesisCierraNumero(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
        Dim car2 As Char = expr(pos + 1)
        'Extrae el siguiente carácter
        'Compara si el primer carácter es paréntesis que cierra y el siguiente es número
        If expr(pos) = ")" Then
            If car2 >= "0" AndAlso car2 <= "9" Then
                Return True
            End If
        End If
    Next
    Return False
End Function
```

C++

```cpp
//7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
bool Evaluar::ParentesisCierraNumero(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
    {
        char car2 = expr[pos+1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        if (expr[pos] == ')')
            if (car2 >= '0' && car2 <= '9') return true;
    }
    return false;
}
```

Object Pascal

```
//7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
function TEvaluar.ParentesisCierraNumero(expr: string): boolean;
var
  pos: integer;
  car2: char;
begin
  for pos := 1 to length(expr) do
  begin
    car2 := expr[pos + 1]; //Extrae el siguiente carácter

    //Compara si el primer carácter es paréntesis que cierra y el siguiente es número
    if (expr[pos] = ')') then
      if (car2 >= '0') and (car2 <= '9')  then
      begin
        Result := true;
```

```
        Exit;
      end;
  end;
  Result := false;
end;
```

JavaScript

```
  //7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
  this.ParentesisCierraNumero = function(expr)
  {
    for(var pos = 0; pos < expr.length - 1; pos++)
    {
      var car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

      //Compara si el primer carácter es paréntesis que cierra y el siguiente es número
      if (expr.charAt(pos) == ')')
        if (car2 >= '0' && car2 <= '9') return true;
    }
    return false;
  }
```

PHP

```
  //7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
  function ParentesisCierraNumero($expr)
  {
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
    {
      $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

      //Compara si el primer carácter es paréntesis que cierra y el siguiente es número
      if ($expr{$pos} == ')')
        if ($car2 >= '0' && $car2 <= '9') return true;
    }
    return false;
  }
```

Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)

Java

```java
//8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
private static boolean NumeroParentesisAbre(String expr)
{
  for(int pos = 0; pos < expr.length() - 1; pos++)
  {
    char car1 = expr.charAt(pos);   //Extrae un carácter

    //Compara si el primer carácter es número y el siguiente es paréntesis que abre
    if (car1 >= '0' && car1 <= '9')
      if (expr.charAt(pos + 1) == '(') return true;
  }
  return false;
}
```

C#

```csharp
//8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
private static Boolean NumeroParentesisAbre(String expr)
{
  for(int pos = 0; pos < expr.Length - 1; pos++)
  {
    char car1 = expr[pos];   //Extrae un carácter

    //Compara si el primer carácter es número y el siguiente es paréntesis que abre
    if (car1 >= '0' && car1 <= '9')
      if (expr[pos + 1] == '(') return true;
  }
  return false;
}
```

Visual Basic .NET

```vb
'8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
Private Shared Function NumeroParentesisAbre(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
        Dim car1 As Char = expr(pos)
        'Extrae un carácter
        'Compara si el primer carácter es número y el siguiente es paréntesis que abre
        If car1 >= "0" AndAlso car1 <= "9" Then
            If expr(pos + 1) = "(" Then
                Return True
            End If
        End If
    Next
    Return False
End Function
```

C++

```cpp
//8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
bool Evaluar::NumeroParentesisAbre(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter

        //Compara si el primer carácter es número y el siguiente es paréntesis que abre
        if (car1 >= '0' && car1 <= '9')
            if (expr[pos+1] == '(') return true;
    }
    return false;
}
```

Object Pascal

```
//8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
function TEvaluar.NumeroParentesisAbre(expr: string): boolean;
var
  pos: integer;
  car1: char;
begin
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];   //Extrae un carácter

    //Compara si el primer carácter es número y el siguiente es paréntesis que abre
    if (car1 >= '0') and (car1 <= '9') then
      if expr[pos + 1] = '(' then
      begin
        Result := true;
        Exit;
```

```
      end;
  end;
  Result := false;
end;
```

JavaScript

```
  //8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
  this.NumeroParentesisAbre = function(expr)
  {
    for(var pos = 0; pos < expr.length - 1; pos++)
    {
      var car1 = expr.charAt(pos);   //Extrae un carácter

      //Compara si el primer carácter es número y el siguiente es paréntesis que abre
      if (car1 >= '0' && car1 <= '9')
        if (expr.charAt(pos + 1) == '(') return true;
    }
    return false;
  }
```

PHP

```
  //8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
  function NumeroParentesisAbre($expr)
  {
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
    {
      $car1 = $expr{$pos};   //Extrae un carácter

      //Compara si el primer carácter es número y el siguiente es paréntesis que abre
      if ($car1 >= '0' && $car1 <= '9')
        if ($expr{$pos + 1} == '(') return true;
    }
    return false;
  }
```

Doble punto en un número de tipo real. Ejemplo: 3-2..4+1 7-6.46.1+2

Java

```java
//9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
private static boolean DoblePuntoNumero(String expr)
{
    int totalpuntos = 0;
    for(int pos = 0; pos < expr.length(); pos++)
    {
        char car1 = expr.charAt(pos);   //Extrae un carácter
        if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
        if (car1 == '.') totalpuntos++;
        if (totalpuntos > 1) return true;
    }
    return false;
}
```

C#

```csharp
//9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
private static Boolean DoblePuntoNumero(String expr)
{
    int totalpuntos = 0;
    for(int pos = 0; pos < expr.Length; pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter
        if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
        if (car1 == '.') totalpuntos++;
        if (totalpuntos > 1) return true;
    }
    return false;
}
```

Visual Basic .NET

```vb
'9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
Private Shared Function DoblePuntoNumero(expr As [String]) As [Boolean]
    Dim totalpuntos As Integer = 0
    For pos As Integer = 0 To expr.Length - 1
        Dim car1 As Char = expr(pos)
        'Extrae un carácter
        If (car1 < "0" OrElse car1 > "9") AndAlso car1 <> "." Then
            totalpuntos = 0
        End If
        If car1 = "." Then
            totalpuntos += 1
        End If
        If totalpuntos > 1 Then
            Return True
        End If
    Next
    Return False
End Function
```

C++

```cpp
//9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
bool Evaluar::DoblePuntoNumero(char *expr)
{
    int totalpuntos = 0;
    for(int pos = 0; *(expr+pos); pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter
        if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
        if (car1 == '.') totalpuntos++;
        if (totalpuntos > 1) return true;
    }
    return false;
}
```

Object Pascal

```pascal
//9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
function TEvaluar.DoblePuntoNumero(expr: string): boolean;
var
  totalpuntos, pos: integer;
  car1: char;
begin
  totalpuntos := 0;
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];   //Extrae un carácter
    if ((car1 < '0') or (car1 > '9')) and (car1 <> '.') then totalpuntos := 0;
    if (car1 = '.') then Inc(totalpuntos);
```

```
    if (totalpuntos > 1) then
    begin
      Result := true;
      Exit;
    end;
  end;
  Result := false;
end;
```

JavaScript

```
    //9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
    this.DoblePuntoNumero = function(expr)
    {
        var totalpuntos = 0;
        for(var pos = 0; pos < expr.length; pos++)
        {
            var car1 = expr.charAt(pos);   //Extrae un carácter
            if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
            if (car1 == '.') totalpuntos++;
            if (totalpuntos > 1) return true;
        }
        return false;
    }
```

PHP

```
    //9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
    function DoblePuntoNumero($expr)
    {
        $totalpuntos = 0;
        for($pos = 0; $pos < strlen($expr); $pos++)
        {
            $car1 = $expr{$pos};   //Extrae un carácter
            if (($car1 < '0' || $car1 > '9') && $car1 != '.') $totalpuntos = 0;
            if ($car1 == '.') $totalpuntos++;
            if ($totalpuntos > 1) return true;
        }
        return false;
    }
```

Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1

Java

```java
//10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
private static boolean ParentesisCierraVariable(String expr)
{
    for(int pos = 0; pos < expr.length() - 1; pos++)
        if (expr.charAt(pos) == ')') //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
                return true;
    return false;
}
```

C#

```csharp
//10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
private static Boolean ParentesisCierraVariable(String expr)
{
    for(int pos = 0; pos < expr.Length - 1; pos++)
        if (expr[pos] == ')') //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (expr[pos + 1] >= 'a' && expr[pos + 1] <= 'z')
                return true;
    return false;
}
```

Visual Basic .NET

```vbnet
'10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
Private Shared Function ParentesisCierraVariable(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
        If expr(pos) = ")" Then
            'Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            If expr(pos + 1) >= "a" AndAlso expr(pos + 1) <= "z" Then
                Return True
            End If
        End If
    Next
    Return False
End Function
```

C++

```cpp
//10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
bool Evaluar::ParentesisCierraVariable(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] == ')') //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (expr[pos+1] >= 'a' && expr[pos+1] <= 'z') return true;
    return false;
}
```

Object Pascal

```
//10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
function TEvaluar.ParentesisCierraVariable(expr: string): boolean;
var
  pos: integer;
begin
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] = ')') then //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
      if (expr[pos + 1] >= 'a') and (expr[pos + 1] <= 'z') then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;
```

JavaScript

```javascript
//10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
this.ParentesisCierraVariable = function(expr)
{
    for(var pos = 0; pos < expr.length - 1; pos++)
        if (expr.charAt(pos) == ')') //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
                return true;
    return false;
}
```

PHP

```php
//10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
function ParentesisCierraVariable($expr)
{
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
        if ($expr{$pos} == ')') //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if ($expr{$pos + 1} >= 'a' && $expr{$pos + 1} <= 'z')
                return true;
    return false;
}
```

Una variable seguida de un punto. Ejemplo: 4-z.1+3

Java

```java
//11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
private static boolean VariableluegoPunto(String expr)
{
    for(int pos = 0; pos < expr.length() - 1; pos++)
        if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
            if (expr.charAt(pos + 1) == '.') return true;
    return false;
}
```

C#

```csharp
//11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
private static Boolean VariableluegoPunto(String expr)
{
    for(int pos = 0; pos < expr.Length - 1; pos++)
        if (expr[pos] >= 'a' && expr[pos] <= 'z')
            if (expr[pos + 1] == '.') return true;
    return false;
}
```

Visual Basic .NET

```vbnet
'11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
Private Shared Function VariableluegoPunto(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
        If expr(pos) >= "a" AndAlso expr(pos) <= "z" Then
            If expr(pos + 1) = "." Then
                Return True
            End If
        End If
    Next
    Return False
End Function
```

C++

```cpp
//11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
bool Evaluar::VariableluegoPunto(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] >= 'a' && expr[pos] <= 'z')
            if (expr[pos+1] == '.') return true;
    return false;
}
```

Object Pascal

```
//11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
function TEvaluar.VariableluegoPunto(expr: string): boolean;
var
  pos: integer;
begin
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] >= 'a') and (expr[pos] <= 'z') then
      if expr[pos + 1] = '.'then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;
```

JavaScript

```javascript
//11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
this.VariableluegoPunto = function(expr)
{
    for(var pos = 0; pos < expr.length - 1; pos++)
        if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
            if (expr.charAt(pos + 1) == '.') return true;
    return false;
}
```

PHP

```php
    //11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
    function VariableluegoPunto($expr)
    {
        for($pos = 0; $pos < strlen($expr) - 1; $pos++)
            if ($expr{$pos} >= 'a' && $expr{$pos} <= 'z')
                if ($expr{$pos + 1} == '.') return true;
        return false;
    }
```

Un punto seguido de una variable. Ejemplo: 7-2.p+1

Java

```java
    //12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
    private static boolean PuntoluegoVariable(String expr)
    {
        for(int pos = 0; pos < expr.length() - 1; pos++)
            if (expr.charAt(pos) == '.')
                if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
                    return true;
        return false;
    }
```

C#

```c#
    //12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
    private static Boolean PuntoluegoVariable(String expr)
    {
        for(int pos = 0; pos < expr.Length - 1; pos++)
            if (expr[pos] == '.')
                if (expr[pos + 1] >= 'a' && expr[pos + 1] <= 'z')
                    return true;
        return false;
    }
```

Visual Basic .NET

```vbnet
    '12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
    Private Shared Function PuntoluegoVariable(expr As [String]) As [Boolean]
        For pos As Integer = 0 To expr.Length - 2
            If expr(pos) = "." Then
                If expr(pos + 1) >= "a" AndAlso expr(pos + 1) <= "z" Then
                    Return True
                End If
            End If
        Next
        Return False
    End Function
```

C++

```c++
//12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
bool Evaluar::PuntoluegoVariable(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] == '.')
            if (expr[pos+1] >= 'a' && expr[pos+1] <= 'z')
                return true;
    return false;
}
```

Object Pascal

```
//12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
function TEvaluar.PuntoluegoVariable(expr: string): boolean;
var
  pos: integer;
begin
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] = '.') then
      if (expr[pos + 1] >= 'a') and (expr[pos + 1] <= 'z') then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;
```

JavaScript

```javascript
    //12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
    this.PuntoluegoVariable = function(expr)
    {
        for(var pos = 0; pos < expr.length - 1; pos++)
            if (expr.charAt(pos) == '.')
                if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
                    return true;
        return false;
    }
```

PHP

```php
//12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
function PuntoluegoVariable($expr)
{
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
        if ($expr{$pos} == '.')
            if ($expr{$pos + 1} >= 'a' && $expr{$pos + 1} <= 'z')
                return true;
    return false;
}
```

Un número antes de una variable. Ejemplo: 3x+1

Java

```
    //13. Un número antes de una variable. Ejemplo: 3x+1
    //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
    private static boolean NumeroAntesVariable(String expr)
    {
        for(int pos = 0; pos < expr.length() - 1; pos++)
            if (expr.charAt(pos) >= '0' && expr.charAt(pos) <= '9')
                if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
                    return true;
        return false;
    }
```

C#

```
    //13. Un número antes de una variable. Ejemplo: 3x+1
    //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
    private static Boolean NumeroAntesVariable(String expr)
    {
        for(int pos = 0; pos < expr.Length - 1; pos++)
            if (expr[pos] >= '0' && expr[pos] <= '9')
                if (expr[pos + 1] >= 'a' && expr[pos + 1] <= 'z')
                    return true;
        return false;
    }
```

Visual Basic .NET

```
    '13. Un número antes de una variable. Ejemplo: 3x+1
    'Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
    Private Shared Function NumeroAntesVariable(expr As [String]) As [Boolean]
        For pos As Integer = 0 To expr.Length - 2
            If expr(pos) >= "0" AndAlso expr(pos) <= "9" Then
                If expr(pos + 1) >= "a" AndAlso expr(pos + 1) <= "z" Then
                    Return True
                End If
            End If
        Next
        Return False
    End Function
```

C++

```
//13. Un número antes de una variable. Ejemplo: 3x+1
//Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
bool Evaluar::NumeroAntesVariable(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] >= '0' && expr[pos] <= '9')
            if (expr[pos+1] >= 'a' && expr[pos+1] <= 'z') return true;
    return false;
}
```

Object Pascal

```
//13. Un número antes de una variable. Ejemplo: 3x+1
//Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
function TEvaluar.NumeroAntesVariable(expr: string): boolean;
var
  pos: integer;
begin
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] >= '0') and (expr[pos] <= '9') then
      if (expr[pos + 1] >= 'a') and (expr[pos + 1] <= 'z') then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;
```

JavaScript

```
    //13. Un número antes de una variable. Ejemplo: 3x+1
```

```
    //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
    this.NumeroAntesVariable = function(expr)
    {
        for(var pos = 0; pos < expr.length - 1; pos++)
            if (expr.charAt(pos) >= '0' && expr.charAt(pos) <= '9')
                if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
                    return true;
        return false;
    }
```

PHP

```
    //13. Un número antes de una variable. Ejemplo: 3x+1
    //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
    function NumeroAntesVariable($expr)
    {
        for($pos = 0; $pos < strlen($expr) - 1; $pos++)
            if ($expr{$pos} >= '0' && $expr{$pos} <= '9')
                if ($expr{$pos + 1} >= 'a' && $expr{$pos + 1} <= 'z')
                    return true;
        return false;
    }
```

Un número después de una variable. Ejemplo: x21+4

Java

```java
//14. Un número después de una variable. Ejemplo: x21+4
private static boolean VariableDespuesNumero(String expr)
{
    for(int pos = 0; pos < expr.length() - 1; pos++)
        if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
            if (expr.charAt(pos + 1) >= '0' && expr.charAt(pos + 1) <= '9')
                return true;
    return false;
}
```

C#

```csharp
//14. Un número después de una variable. Ejemplo: x21+4
private static Boolean VariableDespuesNumero(String expr)
{
    for(int pos = 0; pos < expr.Length - 1; pos++)
        if (expr[pos] >= 'a' && expr[pos] <= 'z')
            if (expr[pos + 1] >= '0' && expr[pos + 1] <= '9')
                return true;
    return false;
}
```

Visual Basic .NET

```vbnet
'14. Un número después de una variable. Ejemplo: x21+4
Private Shared Function VariableDespuesNumero(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
        If expr(pos) >= "a" AndAlso expr(pos) <= "z" Then
            If expr(pos + 1) >= "0" AndAlso expr(pos + 1) <= "9" Then
                Return True
            End If
        End If
    Next
    Return False
End Function
```

C++

```cpp
//14. Un número después de una variable. Ejemplo: x21+4
bool Evaluar::VariableDespuesNumero(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] >= 'a' && expr[pos] <= 'z')
            if (expr[pos+1] >= '0' && expr[pos+1] <= '9')
                return true;
    return false;
}
```

Object Pascal

```
//14. Un número después de una variable. Ejemplo: x21+4
function TEvaluar.VariableDespuesNumero(expr: string): boolean;
var
  pos: integer;
begin
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] >= 'a') and (expr[pos] <= 'z') then
      if (expr[pos + 1] >= '0') and (expr[pos + 1] <= '9') then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;
```

JavaScript

```javascript
//14. Un número después de una variable. Ejemplo: x21+4
this.VariableDespuesNumero = function(expr)
{
    for(var pos = 0; pos < expr.length - 1; pos++)
        if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
            if (expr.charAt(pos + 1) >= '0' && expr.charAt(pos + 1) <= '9')
                return true;
    return false;
}
```

PHP

```php
    //14. Un número después de una variable. Ejemplo: x21+4
    function VariableDespuesNumero($expr)
    {
        for($pos = 0; $pos < strlen($expr) - 1; $pos++)
            if ($expr{$pos} >= 'a' && $expr{$pos} <= 'z')
                if ($expr{$pos + 1} >= '0' && $expr{$pos + 1} <= '9')
                    return true;
        return false;
    }
```

Chequea si hay 4 o más letras seguidas

Java

```
//15. Chequea si hay 4 o más letras seguidas
private static boolean Chequea4letras(String expr)
{
    for(int pos = 0; pos < expr.length() - 3; pos++)
    {
      char car1 = expr.charAt(pos);
      char car2 = expr.charAt(pos + 1);
      char car3 = expr.charAt(pos + 2);
      char car4 = expr.charAt(pos + 3);

      if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <= 'z')
         return true;
    }
    return false;
}
```

C#

```
//15. Chequea si hay 4 o más letras seguidas
private static Boolean Chequea4letras(String expr)
{
  for(int pos = 0; pos < expr.Length - 3; pos++)
  {
    char car1 = expr[pos];
    char car2 = expr[pos + 1];
    char car3 = expr[pos + 2];
    char car4 = expr[pos + 3];

    if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <= 'z')
      return true;
  }
  return false;
}
```

Visual Basic .NET

```
    '15. Chequea si hay 4 o más letras seguidas
    Private Shared Function Chequea4letras(expr As [String]) As [Boolean]
        For pos As Integer = 0 To expr.Length - 4
            Dim car1 As Char = expr(pos)
            Dim car2 As Char = expr(pos + 1)
            Dim car3 As Char = expr(pos + 2)
            Dim car4 As Char = expr(pos + 3)

            If car1 >= "a" AndAlso car1 <= "z" AndAlso car2 >= "a" AndAlso car2 <= "z" AndAlso car3 >= "a" AndAlso car3 <= "z"
AndAlso car4 >= "a" AndAlso car4 <= "z" Then
                Return True
            End If
        Next
        Return False
    End Function
```

C++

```
//15. Chequea si hay 4 o más letras seguidas
bool Evaluar::Chequea4letras(char *expr)
{
  for(int pos = 0; *(expr+pos+3); pos++)
  {
    char car1 = expr[pos];
    char car2 = expr[pos+1];
    char car3 = expr[pos+2];
    char car4 = expr[pos+3];

    if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <= 'z')
      return true;
  }
  return false;
}
```

Object Pascal

```
//15. Chequea si hay 4 o más letras seguidas
function TEvaluar.Chequea4letras(expr: string): boolean;
var
  pos: integer;
  car1, car2, car3, car4: char;
begin
  for pos := 1 to length(expr)-3 do
  begin
    car1 := expr[pos];
```

```
    car2 := expr[pos + 1];
    car3 := expr[pos + 2];
    car4 := expr[pos + 3];

    if (car1 >= 'a') and (car1 <= 'z') and (car2 >= 'a') and (car2 <= 'z') and (car3 >= 'a') and (car3 <= 'z') and (car4 >= 'a')
and (car4 <= 'z') then
    begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;
```

JavaScript

```
//15. Chequea si hay 4 o más letras seguidas
this.Chequea4letras = function(expr)
{
  for(var pos = 0; pos < expr.length - 3; pos++)
  {
     var car1 = expr.charAt(pos);
     var car2 = expr.charAt(pos + 1);
     var car3 = expr.charAt(pos + 2);
     var car4 = expr.charAt(pos + 3);

     if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <= 'z')
        return true;
  }
  return false;
}
```

PHP

```
    //15. Chequea si hay 4 o más letras seguidas
    function Chequea4letras($expr)
    {
        for($pos = 0; $pos < strlen($expr) - 3; $pos++)
        {
            $car1 = $expr{$pos};
            $car2 = $expr{$pos + 1};
            $car3 = $expr{$pos + 2};
            $car4 = $expr{$pos + 3};

            if ($car1 >= 'a' && $car1 <= 'z' && $car2 >= 'a' && $car2 <= 'z' && $car3 >= 'a' && $car3 <= 'z' && $car4 >= 'a' &&
$car4 <= 'z')
                return true;
        }
        return false;
    }
```

Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no

Java

```java
//16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
private boolean FuncionInvalida(String expr)
{
    for(int pos = 0; pos < expr.length() - 2; pos++)
    {
        char car1 = expr.charAt(pos);
        char car2 = expr.charAt(pos + 1);
        char car3 = expr.charAt(pos + 2);

        //Si encuentra tres letras seguidas
        if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
        {
            if (pos >= expr.length() - 4) return true; //Hay un error porque no sigue paréntesis
            if (expr.charAt(pos + 3) != '(') return true; //Hay un error porque no hay paréntesis
            if (EsFuncionInvalida(car1, car2, car3)) return true;
        }
    }
    return false;
}

//Chequea si las tres letras enviadas son una función
private static boolean EsFuncionInvalida(char car1, char car2, char car3)
{
    String listafunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";
    for(int pos = 0; pos <= listafunciones.length() - 3; pos+=3)
    {
        char listfunc1 = listafunciones.charAt(pos);
        char listfunc2 = listafunciones.charAt(pos + 1);
        char listfunc3 = listafunciones.charAt(pos + 2);
        if (car1 == listfunc1 && car2 == listfunc2 && car3 == listfunc3) return false;
    }
    return true;
}
```

C#

```csharp
//16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
private Boolean FuncionInvalida(String expr)
{
    for(int pos = 0; pos < expr.Length - 2; pos++)
    {
        char car1 = expr[pos];
        char car2 = expr[pos + 1];
        char car3 = expr[pos + 2];

        //Si encuentra tres letras seguidas
        if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
        {
            if (pos >= expr.Length - 4) return true; //Hay un error porque no sigue paréntesis
            if (expr[pos + 3] != '(') return true; //Hay un error porque no hay paréntesis
            if (FuncionInvalida(car1, car2, car3)) return true;
        }
    }
    return false;
}

//Chequea si las tres letras enviadas son una función
private static Boolean FuncionInvalida(char car1, char car2, char car3)
{
    String listafunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";
    for(int pos = 0; pos <= listafunciones.Length - 3; pos+=3)
    {
        char listfunc1 = listafunciones[pos];
        char listfunc2 = listafunciones[pos + 1];
        char listfunc3 = listafunciones[pos + 2];
        if (car1 == listfunc1 && car2 == listfunc2 && car3 == listfunc3) return false;
    }
    return true;
}
```

Visual Basic .NET

```vb
'16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
Private Function FuncionInvalida(expr As [String]) As [Boolean]
 For pos As Integer = 0 To expr.Length - 3
   Dim car1 As Char = expr(pos)
   Dim car2 As Char = expr(pos + 1)
   Dim car3 As Char = expr(pos + 2)

   'Si encuentra tres letras seguidas
   If car1 >= "a" AndAlso car1 <= "z" AndAlso car2 >= "a" AndAlso car2 <= "z" AndAlso car3 >= "a" AndAlso car3 <= "z" Then
     If pos >= expr.Length - 4 Then
       Return True
     End If
     'Hay un error porque no sigue paréntesis
```

```
      If expr(pos + 3) <> "(" Then
        Return True
      End If
      'Hay un error porque no hay paréntesis
      If FuncionInvalida(car1, car2, car3) Then
         Return True
      End If
    End If
  Next
  Return False
End Function

'Chequea si las tres letras enviadas son una función
Private Shared Function FuncionInvalida(car1 As Char, car2 As Char, car3 As Char) As [Boolean]
    Dim listafunciones As [String] = "sinsencostanabsasnacsatnlogceiexpsqrrcb"
    For pos As Integer = 0 To listafunciones.Length - 3 Step 3
      Dim listfunc1 As Char = listafunciones(pos)
      Dim listfunc2 As Char = listafunciones(pos + 1)
      Dim listfunc3 As Char = listafunciones(pos + 2)
      If car1 = listfunc1 AndAlso car2 = listfunc2 AndAlso car3 = listfunc3 Then
        Return False
      End If
    Next
    Return True
End Function
```

C++

```
//16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
bool Evaluar::FuncionInvalida(char *expr)
{
    for(int pos = 0; *(expr+pos+2); pos++)
    {
        char car1 = expr[pos];
        char car2 = expr[pos+1];
        char car3 = expr[pos+2];

        //Si encuentra tres letras seguidas
        if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
        {
            if (pos >= strlen(expr) - 4) return true; //Hay un error porque no sigue paréntesis
            if (*(expr+pos+3) != '(') return true; //Hay un error porque no hay paréntesis
            if (EsFuncionInvalida(car1, car2, car3)) return true;
        }
    }
    return false;
}

//Chequea si las tres letras enviadas son una función
bool Evaluar::EsFuncionInvalida(char car1, char car2, char car3)
{
    char *listafunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";
    for(int pos = 0; *(listafunciones+pos+2); pos+=3)
    {
        char listfunc1 = listafunciones[pos];
        char listfunc2 = listafunciones[pos + 1];
        char listfunc3 = listafunciones[pos + 2];
        if (car1 == listfunc1 && car2 == listfunc2 && car3 == listfunc3) return false;
    }
    return true;
}
```

Object Pascal

```
//16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
function TEvaluar.FuncionInvalida(expr: string): boolean;
var
  pos: integer;
  car1, car2, car3: char;
begin
  for pos := 1 to length(expr)-2 do
  begin
    car1 := expr[pos];
    car2 := expr[pos + 1];
    car3 := expr[pos + 2];

    //Si encuentra tres letras seguidas
    if (car1 >= 'a') and (car1 <= 'z') and (car2 >= 'a') and (car2 <= 'z') and (car3 >= 'a') and (car3 <= 'z') then
    begin
      if pos >= length(expr) - 4 then begin Result:= true; Exit; end;//Hay un error porque no sigue paréntesis
      if expr[pos + 3] <> '(' then begin Result:= true; Exit; end; //Hay un error porque no hay paréntesis
      if EsFuncionInvalida(car1, car2, car3) then begin Result:= true; Exit; end;
    end;
  end;
  Result := false;
end;

//Chequea si las tres letras enviadas son una función
```

```
function TEvaluar.EsFuncionInvalida(car1: char; car2: char; car3: char): boolean;
var
  pos, tamfunciones:integer;
  listfunc1, listfunc2, listfunc3: char;
  listafunciones: string;
begin
  listafunciones := 'sinsencostanabsasnacsatnlogceiexpsqrrcb';
  tamfunciones := length(listafunciones);
  pos := 1;
  while (pos <= tamfunciones - 2) do
  begin
    listfunc1 := listafunciones[pos];
    listfunc2 := listafunciones[pos + 1];
    listfunc3 := listafunciones[pos + 2];

    if (car1 = listfunc1) and (car2 = listfunc2) and (car3 = listfunc3) then
    begin
      Result := true;
      exit;
    end;

    pos := pos + 3;
  end;
  Result := false;
end;
```

JavaScript

```
//16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
this.FuncionInvalida = function(expr)
{
    for(var pos = 0; pos < expr.length - 2; pos++)
    {
        var car1 = expr.charAt(pos);
        var car2 = expr.charAt(pos + 1);
        var car3 = expr.charAt(pos + 2);

        //Si encuentra tres letras seguidas
        if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
        {
            if (pos >= expr.length - 4) return true; //Hay un error porque no sigue paréntesis
            if (expr.charAt(pos + 3) != '(') return true; //Hay un error porque no hay paréntesis
            if (this.EsFuncionInvalida(car1, car2, car3)) return true;
        }
    }
    return false;
}

//Chequea si las tres letras enviadas son una función
this.EsFuncionInvalida = function(car1, car2, car3)
{
    var listafunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";
    for(var pos = 0; pos <= listafunciones.length - 3; pos+=3)
    {
        var listfunc1 = listafunciones.charAt(pos);
        var listfunc2 = listafunciones.charAt(pos + 1);
        var listfunc3 = listafunciones.charAt(pos + 2);
        if (car1 == listfunc1 && car2 == listfunc2 && car3 == listfunc3) return false;
    }
    return true;
}
```

PHP

```
//16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
function FuncionInvalida($expr)
{
    for($pos = 0; $pos < strlen($expr) - 2; $pos++)
    {
        $car1 = $expr{$pos};
        $car2 = $expr{$pos + 1};
        $car3 = $expr{$pos + 2};

        //Si encuentra tres letras seguidas
        if ($car1 >= 'a' && $car1 <= 'z' && $car2 >= 'a' && $car2 <= 'z' && $car3 >= 'a' && $car3 <= 'z')
        {
            if ($pos >= strlen($expr) - 4) return true; //Hay un error porque no sigue paréntesis
            if ($expr{$pos + 3} != '(') return true; //Hay un error porque no hay paréntesis
            if ($this->EsFuncionInvalida($car1, $car2, $car3)) return true;
        }
    }
    return false;
}

//Chequea si las tres letras enviadas son una función
function EsFuncionInvalida($car1, $car2, $car3)
{
```

```
        $listafunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";
        for($pos = 0; $pos <= strlen($listafunciones) - 3; $pos+=3)
        {
            $listfunc1 = $listafunciones{$pos};
            $listfunc2 = $listafunciones{$pos + 1};
            $listfunc3 = $listafunciones{$pos + 2};
            if ($car1 == $listfunc1 && $car2 == $listfunc2 && $car3 == $listfunc3) return false;
        }
        return true;
    }
```

Si detecta sólo dos letras seguidas es un error

Java

```java
//17. Si detecta sólo dos letras seguidas es un error
private static boolean VariableInvalida(String expr)
{
    int cuentaletras = 0;
    for(int pos = 0; pos < expr.length(); pos++)
    {
        if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
            cuentaletras++;
        else
        {
            if (cuentaletras == 2) return true;
            cuentaletras = 0;
        }
    }
    return cuentaletras == 2;
}
```

C#

```csharp
//17. Si detecta sólo dos letras seguidas es un error
private static Boolean VariableInvalida(String expr)
{
    int cuentaletras = 0;
    for(int pos = 0; pos < expr.Length; pos++)
    {
        if (expr[pos] >= 'a' && expr[pos] <= 'z')
            cuentaletras++;
        else
        {
            if (cuentaletras == 2) return true;
            cuentaletras = 0;
        }
    }
    return cuentaletras == 2;
}
```

Visual Basic .NET

```vb
'17. Si detecta sólo dos letras seguidas es un error
Private Shared Function VariableInvalida(expr As [String]) As [Boolean]
    Dim cuentaletras As Integer = 0
    For pos As Integer = 0 To expr.Length - 1
        If expr(pos) >= "a" AndAlso expr(pos) <= "z" Then
            cuentaletras += 1
        Else
            If cuentaletras = 2 Then
                Return True
            End If
            cuentaletras = 0
        End If
    Next
    Return cuentaletras = 2
End Function
```

C++

```cpp
//17. Si detecta sólo dos letras seguidas es un error
bool Evaluar::VariableInvalida(char *expr)
{
    int cuentaletras = 0;
    for(int pos = 0; *(expr+pos); pos++)
    {
      char car1 = expr[pos];
      if (car1 >= 'a' && car1 <= 'z')
        cuentaletras++;
      else
      {
        if (cuentaletras == 2) return true;
        cuentaletras = 0;
      }
    }
    return cuentaletras == 2;
}
```

Object Pascal

```
//17. Si detecta sólo dos letras seguidas es un error
function TEvaluar.VariableInvalida(expr: string): boolean;
var
  cuentaletras, pos: integer;
begin
```

```
  cuentaletras := 0;
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] >= 'a') and (expr[pos] <= 'z') then
      Inc(cuentaletras)
    else
    begin
      if cuentaletras = 2 then
      begin
        Result := true;
        exit;
      end;
      cuentaletras := 0;
    end;
  end;
  if cuentaletras = 2 then
    Result := true
  else
    Result := false;
end;
```

JavaScript

```
    //17. Si detecta sólo dos letras seguidas es un error
    this.VariableInvalida = function(expr)
    {
        var cuentaletras = 0;
        for(var pos = 0; pos < expr.length; pos++)
        {
            if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
                cuentaletras++;
            else
            {
                if (cuentaletras == 2) return true;
                cuentaletras = 0;
            }
        }
        return cuentaletras == 2;
    }
```

PHP

```
    //17. Si detecta sólo dos letras seguidas es un error
    function VariableInvalida($expr)
    {
        $cuentaletras = 0;
        for($pos = 0; $pos < strlen($expr); $pos++)
        {
            if ($expr{$pos} >= 'a' && $expr{$pos} <= 'z')
                $cuentaletras++;
            else
            {
                if ($cuentaletras == 2) return true;
                $cuentaletras = 0;
            }
        }
        return $cuentaletras == 2;
    }
```

Antes de paréntesis que abre hay una letra

Java

```java
//18. Antes de paréntesis que abre hay una letra
private static boolean VariableParentesisAbre(String expr)
{
    int cuentaletras = 0;
    for(int pos = 0; pos < expr.length(); pos++)
    {
        char car1 = expr.charAt(pos);
        if (car1 >= 'a' && car1 <= 'z')
            cuentaletras++;
        else if (car1 == '(' && cuentaletras == 1)
            return true;
        else
            cuentaletras = 0;
    }
    return false;
}
```

C#

```csharp
//18. Antes de paréntesis que abre hay una letra
private static Boolean VariableParentesisAbre(String expr)
{
    int cuentaletras = 0;
    for(int pos = 0; pos < expr.Length; pos++)
    {
        char car1 = expr[pos];
        if (car1 >= 'a' && car1 <= 'z')
            cuentaletras++;
        else if (car1 == '(' && cuentaletras == 1)
            return true;
        else
            cuentaletras = 0;
    }
    return false;
}
```

Visual Basic .NET

```vbnet
'18. Antes de paréntesis que abre hay una letra
Private Shared Function VariableParentesisAbre(expr As [String]) As [Boolean]
    Dim cuentaletras As Integer = 0
    For pos As Integer = 0 To expr.Length - 1
        Dim car1 As Char = expr(pos)
        If car1 >= "a" AndAlso car1 <= "z" Then
            cuentaletras += 1
        ElseIf car1 = "(" AndAlso cuentaletras = 1 Then
            Return True
        Else
            cuentaletras = 0
        End If
    Next
    Return False
End Function
```

C++

```cpp
//18. Antes de paréntesis que abre hay una letra
bool Evaluar::VariableParentesisAbre(char *expr)
{
    int cuentaletras = 0;
    for(int pos = 0; *(expr+pos); pos++)
    {
        char car1 = expr[pos];
        if (car1 >= 'a' && car1 <= 'z')
           cuentaletras++;
        else if (car1 == '(' && cuentaletras == 1)
           return true;
        else
           cuentaletras = 0;
    }
    return false;
}
```

Object Pascal

```
//18. Antes de paréntesis que abre hay una letra
function TEvaluar.VariableParentesisAbre(expr: string): boolean;
var
  pos, cuentaletras: integer;
  car1: char;
begin
```

```
  cuentaletras := 0;
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];
    if (car1 >= 'a') and (car1 <= 'z') then
      Inc(cuentaletras)
    else if (car1 = '(') and (cuentaletras = 1) then
    begin
      Result := true;
      Exit;
    end
    else
      cuentaletras := 0;
  end;
  Result := false;
end;
```

JavaScript

```
    //18. Antes de paréntesis que abre hay una letra
    this.VariableParentesisAbre = function(expr)
    {
        var cuentaletras = 0;
        for(var pos = 0; pos < expr.length; pos++)
        {
            var car1 = expr.charAt(pos);
            if (car1 >= 'a' && car1 <= 'z')
                cuentaletras++;
            else if (car1 == '(' && cuentaletras == 1)
                return true;
            else
                cuentaletras = 0;
        }
        return false;
    }
```

PHP

```
    //18. Antes de paréntesis que abre hay una letra
    function VariableParentesisAbre($expr)
    {
        $cuentaletras = 0;
        for($pos = 0; $pos < strlen($expr); $pos++)
        {
            $car1 = $expr{$pos};
            if ($car1 >= 'a' && $car1 <= 'z')
                $cuentaletras++;
            else if ($car1 == '(' && $cuentaletras == 1)
                return true;
            else
                $cuentaletras = 0;
        }
        return false;
    }
```

Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)

Java

```java
//19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
private static boolean ParCierraParAbre(String expr)
{
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)==')' && expr.charAt(pos+1)=='(')
            return true;
    return false;
}
```

C#

```csharp
//19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
private static Boolean ParCierraParAbre(String expr)
{
    for (int pos = 0; pos < expr.Length-1; pos++)
        if (expr[pos]==')' && expr[pos+1]=='(')
            return true;
    return false;
}
```

Visual Basic .NET

```vbnet
'19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
Private Shared Function ParCierraParAbre(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
        If expr(pos) = ")" AndAlso expr(pos + 1) = "(" Then
            Return True
        End If
    Next
    Return False
End Function
```

C++

```cpp
//19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
bool Evaluar::ParCierraParAbre(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]==')' && expr[pos+1]=='(')
              return true;
      return false;
}
```

Object Pascal

```
//19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
function TEvaluar.ParCierraParAbre(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]=')') and (expr[pos+1]='(') then
          begin
              Result := true;
              Exit;
          end;
      end;
      Result := false;
end;
```

JavaScript

```javascript
//19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
this.ParCierraParAbre = function(expr)
{
    for (var pos = 0; pos < expr.length-1; pos++)
        if (expr.charAt(pos)==')' && expr.charAt(pos+1)=='(')
            return true;
    return false;
}
```

PHP

```php
//19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
function ParCierraParAbre($expr)
{
```

```
        for ($pos = 0; $pos < strlen($expr)-1; $pos++)
            if ($expr{$pos}==')' && $expr{$pos+1}=='(')
                return true;
        return false;
    }
```

Después de operador sigue un punto. Ejemplo: -.3+7

Java

```
  //20. Después de operador sigue un punto. Ejemplo: -.3+7
  private static boolean OperadorPunto(String expr)
  {
      for (int pos = 0; pos < expr.length()-1; pos++)
          if (expr.charAt(pos)=='+' || expr.charAt(pos)=='-' || expr.charAt(pos)=='*' || expr.charAt(pos)=='/' ||
expr.charAt(pos)=='^')
                  if (expr.charAt(pos+1)=='.')
                    return true;
      return false;
  }
```

C#

```
  //20. Después de operador sigue un punto. Ejemplo: -.3+7
  private static Boolean OperadorPunto(String expr)
  {
      for (int pos = 0; pos < expr.Length-1; pos++)
          if (expr[pos]=='+' || expr[pos]=='-' || expr[pos]=='*' || expr[pos]=='/' || expr[pos]=='^')
                  if (expr[pos+1]=='.')
                    return true;
      return false;
  }
```

Visual Basic .NET

```
    '20. Después de operador sigue un punto. Ejemplo: -.3+7
    Private Shared Function OperadorPunto(expr As [String]) As [Boolean]
        For pos As Integer = 0 To expr.Length - 2
            If expr(pos) = "+" OrElse expr(pos) = "-" OrElse expr(pos) = "*" OrElse expr(pos) = "/" OrElse expr(pos) = "^" Then
                If expr(pos + 1) = "." Then
                    Return True
                End If
            End If
        Next
        Return False
    End Function
```

C++

```
//20. Después de operador sigue un punto. Ejemplo: -.3+7
bool Evaluar::OperadorPunto(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]=='+' || expr[pos]=='-' || expr[pos]=='*' || expr[pos]=='/' || expr[pos]=='^')
                  if (expr[pos+1]=='.')
                    return true;
      return false;
}
```

Object Pascal

```
//20. Después de operador sigue un punto. Ejemplo: -.3+7
function TEvaluar.OperadorPunto(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]='+') or (expr[pos]='-') or (expr[pos]='*') or (expr[pos]='/') or (expr[pos]='^') then
                  if expr[pos+1]='.' then
                  begin
                    Result := true;
                    Exit;
                  end;
      end;
      Result := false;
end;
```

JavaScript

```
  //20. Después de operador sigue un punto. Ejemplo: -.3+7
  this.OperadorPunto = function(expr)
  {
      for (var pos = 0; pos < expr.length-1; pos++)
          if (expr.charAt(pos)=='+' || expr.charAt(pos)=='-' || expr.charAt(pos)=='*' || expr.charAt(pos)=='/' ||
expr.charAt(pos)=='^')
                  if (expr.charAt(pos+1)=='.')
                    return true;
      return false;
```

```
}
```

PHP

```
//20. Después de operador sigue un punto. Ejemplo: -.3+7
function OperadorPunto($expr)
{
    for ($pos = 0; $pos < strlen($expr)-1; $pos++)
        if ($expr{$pos}=='+' || $expr{$pos}=='-' || $expr{$pos}=='*' || $expr{$pos}=='/' || $expr{$pos}=='^')
                if ($expr{$pos+1}=='.')
                  return true;
    return false;
}
```

Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)

Java

```java
//21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
private static boolean ParAbrePunto(String expr)
{
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)=='(' && expr.charAt(pos+1)=='.')
            return true;
    return false;
}
```

C#

```csharp
//21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
private static Boolean ParAbrePunto(String expr)
{
    for (int pos = 0; pos < expr.Length-1; pos++)
        if (expr[pos]=='(' && expr[pos+1]=='.')
            return true;
    return false;
}
```

Visual Basic .NET

```vb
'21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
Private Shared Function ParAbrePunto(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
        If expr(pos) = "(" AndAlso expr(pos + 1) = "." Then
            Return True
        End If
    Next
    Return False
End Function
```

C++

```cpp
//21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
bool Evaluar::ParAbrePunto(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]=='(' && expr[pos+1]=='.')
              return true;
      return false;
}
```

Object Pascal

```
//21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
function TEvaluar.ParAbrePunto(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]='(') and (expr[pos+1]='.') then
          begin
              Result := true;
              Exit;
          end;
      end;
      Result := false;
end;
```

JavaScript

```javascript
//21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
this.ParAbrePunto = function(expr)
{
    for (var pos = 0; pos < expr.length-1; pos++)
        if (expr.charAt(pos)=='(' && expr.charAt(pos+1)=='.')
            return true;
    return false;
}
```

PHP

```php
//21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
function ParAbrePunto($expr)
{
```

```
        for ($pos = 0; $pos < strlen($expr)-1; $pos++)
            if ($expr{$pos}=='(' && $expr{$pos+1}=='.')
                return true;
        return false;
    }
```

Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)

Java

```java
//22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
private static boolean PuntoParAbre(String expr)
{
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)=='.' && expr.charAt(pos+1)=='(')
            return true;
    return false;
}
```

C#

```csharp
//22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
private static Boolean PuntoParAbre(String expr)
{
    for (int pos = 0; pos < expr.Length-1; pos++)
        if (expr[pos]=='.' && expr[pos+1]=='(')
            return true;
    return false;
}
```

Visual Basic .NET

```vbnet
    '22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
    Private Shared Function PuntoParAbre(expr As [String]) As [Boolean]
        For pos As Integer = 0 To expr.Length - 2
            If expr(pos) = "." AndAlso expr(pos + 1) = "(" Then
                Return True
            End If
        Next
        Return False
    End Function
```

C++

```cpp
//22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
bool Evaluar::PuntoParAbre(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]=='.' && expr[pos+1]=='(')
              return true;
      return false;
}
```

Object Pascal

```
//22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
function TEvaluar.PuntoParAbre(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]='.') and (expr[pos+1]='(') then
          begin
              Result := true;
              Exit;
          end;
      end;
      Result := false;
end;
```

JavaScript

```javascript
  //22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
  this.PuntoParAbre = function(expr)
  {
      for (var pos = 0; pos < expr.length-1; pos++)
          if (expr.charAt(pos)=='.' && expr.charAt(pos+1)=='(')
              return true;
      return false;
  }
```

PHP

```php
  //22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
  function PuntoParAbre($expr)
```

```
{
    for ($pos = 0; $pos < strlen($expr)-1; $pos++)
        if ($expr{$pos}=='.' && $expr{$pos+1}=='(')
            return true;
    return false;
}
```

Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2

Java

```java
//23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
private static boolean ParCierraPunto(String expr)
{
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)==')' && expr.charAt(pos+1)=='.')
            return true;
    return false;
}
```

C#

```csharp
//23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
private static Boolean ParCierraPunto(String expr)
{
    for (int pos = 0; pos < expr.Length-1; pos++)
        if (expr[pos]==')' && expr[pos+1]=='.')
            return true;
    return false;
}
```

Visual Basic .NET

```vbnet
'23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
Private Shared Function ParCierraPunto(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
        If expr(pos) = ")" AndAlso expr(pos + 1) = "." Then
            Return True
        End If
    Next
    Return False
End Function
```

C++

```cpp
//23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
bool Evaluar::ParCierraPunto(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]==')' && expr[pos+1]=='.')
              return true;
      return false;
}
```

Object Pascal

```
//23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
function TEvaluar.ParCierraPunto(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]=')') and (expr[pos+1]='.') then
          begin
              Result := true;
              Exit;
          end;
      end;
      Result := false;
end;
```

JavaScript

```javascript
//23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
this.ParCierraPunto = function(expr)
{
    for (var pos = 0; pos < expr.length-1; pos++)
        if (expr.charAt(pos)==')' && expr.charAt(pos+1)=='.')
            return true;
    return false;
}
```

PHP

```php
//23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
function ParCierraPunto($expr)
{
```

```
        for ($pos = 0; $pos < strlen($expr)-1; $pos++)
            if ($expr{$pos}==')' && $expr{$pos+1}=='.')
                return true;
        return false;
    }
```

Punto seguido de operador. Ejemplo: 5.*9+1

Java

```
  //24. Punto seguido de operador. Ejemplo: 5.*9+1
  private static boolean PuntoOperador(String expr)
  {
      for (int pos = 0; pos < expr.length()-1; pos++)
          if (expr.charAt(pos)=='.')
            if (expr.charAt(pos+1)=='+' || expr.charAt(pos+1)=='-' || expr.charAt(pos+1)=='*' || expr.charAt(pos+1)=='/' ||
expr.charAt(pos+1)=='^')
                    return true;
      return false;
  }
```

C#

```
  //24. Punto seguido de operador. Ejemplo: 5.*9+1
  private static Boolean PuntoOperador(String expr)
  {
      for (int pos = 0; pos < expr.Length-1; pos++)
          if (expr[pos]=='.')
            if (expr[pos+1]=='+' || expr[pos+1]=='-' || expr[pos+1]=='*' || expr[pos+1]=='/' || expr[pos+1]=='^')
                    return true;
      return false;
  }
```

Visual Basic .NET

```
    '24. Punto seguido de operador. Ejemplo: 5.*9+1
    Private Shared Function PuntoOperador(expr As [String]) As [Boolean]
        For pos As Integer = 0 To expr.Length - 2
            If expr(pos) = "." Then
                If expr(pos + 1) = "+" OrElse expr(pos + 1) = "-" OrElse expr(pos + 1) = "*" OrElse expr(pos + 1) = "/" OrElse
expr(pos + 1) = "^" Then
                    Return True
                End If
            End If
        Next
        Return False
    End Function
```

C++

```
//24. Punto seguido de operador. Ejemplo: 5.*9+1
bool Evaluar::PuntoOperador(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]=='.')
            if (expr[pos+1]=='+' || expr[pos+1]=='-' || expr[pos+1]=='*' || expr[pos+1]=='/' || expr[pos+1]=='^')
                    return true;
      return false;
}
```

Object Pascal

```
//24. Punto seguido de operador. Ejemplo: 5.*9+1
function TEvaluar.PuntoOperador(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]='.') then
            if (expr[pos+1]='+') or (expr[pos+1]='-') or (expr[pos+1]='*') or (expr[pos+1]='/') or (expr[pos+1]='^') then
            begin
                    Result := true;
                    Exit;
            end;
      end;
      Result := false;
end;
```

JavaScript

```
  //24. Punto seguido de operador. Ejemplo: 5.*9+1
  this.PuntoOperador = function(expr)
  {
      for (var pos = 0; pos < expr.length-1; pos++)
          if (expr.charAt(pos)=='.')
            if (expr.charAt(pos+1)=='+' || expr.charAt(pos+1)=='-' || expr.charAt(pos+1)=='*' || expr.charAt(pos+1)=='/' ||
expr.charAt(pos+1)=='^')
                    return true;
```

```
        return false;
    }
```

PHP

```
    //24. Punto seguido de operador. Ejemplo: 5.*9+1
    function PuntoOperador($expr)
    {
        for ($pos = 0; $pos < strlen($expr)-1; $pos++)
            if ($expr{$pos}=='.')
              if ($expr{$pos+1}=='+' || $expr{$pos+1}=='-' || $expr{$pos+1}=='*' || $expr{$pos+1}=='/' || $expr{$pos+1}=='^')
                    return true;
        return false;
    }
```

Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5

Java

```java
//25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
private static boolean PuntoParCierra(String expr)
{
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)=='.' && expr.charAt(pos+1)==')')
            return true;
    return false;
}
```

C#

```csharp
//25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
private static Boolean PuntoParCierra(String expr)
{
    for (int pos = 0; pos < expr.Length-1; pos++)
        if (expr[pos]=='.' && expr[pos+1]==')')
            return true;
    return false;
}
```

Visual Basic .NET

```vbnet
'25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
Private Shared Function PuntoParCierra(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
        If expr(pos) = "." AndAlso expr(pos + 1) = ")" Then
            Return True
        End If
    Next
    Return False
End Function
```

C++

```cpp
//25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
bool Evaluar::PuntoParCierra(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]=='.' && expr[pos+1]==')')
              return true;
      return false;
}
```

Object Pascal

```
//25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
function TEvaluar.PuntoParCierra(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]='.') and (expr[pos+1]=')') then
          begin
              Result := true;
              Exit;
          end;
      end;
      Result := false;
end;
```

JavaScript

```javascript
//25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
this.PuntoParCierra = function(expr)
{
    for (var pos = 0; pos < expr.length-1; pos++)
        if (expr.charAt(pos)=='.' && expr.charAt(pos+1)==')')
            return true;
    return false;
}
```

PHP

```php
//25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
function PuntoParCierra($expr)
```

```
{
    for ($pos = 0; $pos < strlen($expr)-1; $pos++)
        if ($expr{$pos}=='.' && $expr{$pos+1}==')')
            return true;
    return false;
}
```

El método Analizar_menos_unarios()

Una nueva funcionalidad que tiene este evaluador de expresiones con respecto a la anterior es la interpretación del menos unario. Este evaluador puede interpretar por ejemplo:

1. 7*-3
2. -5+12
3. 9/(-5 + 3)
4. 8*-3*(-4+5)
5. 7^-2
6. 3*-cos(56)+sen(89)/-tan(12)
7. -3*-8.985/(-4.23*-cos(34/-7)+3^-4)

¿Cómo lo hace? En este caso lo que hace el algoritmo es transformar la expresión para que desaparezca el menos unario y se mantenga la misma lógica matemática. Hay que tener un muy especial cuidado porque no es lo mismo 0-2^2 que -2^2 , en el primero el resultado es -4 y en el segundo es 4. Es muy tentador el agregar un cero al inicio para eliminar unos casos de menos unarios, pero no funciona en todos.

Obsérvese estos casos:

1. Después de un operador suma(+), resta(-), multiplicación(*) sigue el menos unario

   Ejemplos: 4*-3 8+-17 91 - -35

   Solución: Reemplazar el menos unario por un (0-1)* y se respeta la lógica

   Resultados: 4*(0-1)*3 8+(0-1)*17 91 – (0-1)*35

2. Después de un operador potencia(^) y división (/) sigue el menos unario

   Ejemplos: 156/-41 3^-2

   Solución: El problema es que no serviría reemplazar el menos unario por (0-1)* porque la precedencia de los operadores hace que la división y la potencia sea igual o mayor que la multiplicación respectivamente, **no** es lo mismo 156/-41 que 156/(0-1)*41, **no** es lo mismo 3^-2 que 3^(0-1)*2 . Se requeriría que se evaluase primero el (0-1)* antes que cualquier otra cosa y para solucionar esto, se hace uso de un nuevo símbolo, en este algoritmo se usa el # el cual es una operación con la mayor precedencia. Entonces quedaría así: 156/(0-1)#41, se resuelve primero los paréntesis 156/-1#41, luego el #, 156/-41 y por lo tanto se respeta la operación matemática. En el caso de 3^-2 quedaría 3^(0-1)#2, se resuelve los paréntesis 3^-1#2, luego el #, 3^-2 y por lo tanto se respeta la operación matemática.

   Resultados: 156/(0-1)#41, 3^(0-1)#2

   Como el procedimiento en el punto 2 se puede aplicar en el punto 1, entonces se reemplaza el menos unario por un (0-1)#

3. Inicia con menos unario justo después de un paréntesis

   Ejemplos: (-3+2) (6*(-5+9)

   Solución: Similar al punto anterior, se reemplaza el menos unario por (0-1)#

   Resultados: ((0-1)#3+2) (6*((0-1)#5+9)

Java

```
/* Convierte una expresión con el menos unario en una expresión válida para el evaluador de expresiones */
public final String ArreglaNegativos(String expresion)
{
  StringBuilder NuevaExpresion = new StringBuilder();
  StringBuilder NuevaExpresion2 = new StringBuilder();

  //Si detecta un operador y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  for (int pos=0; pos<expresion.length(); pos++)
  {
    char letra1 = expresion.charAt(pos);
    if (letra1=='+' || letra1=='-' || letra1=='*' || letra1=='/' || letra1=='^')
      if (expresion.charAt(pos+1)=='-')
      {
        NuevaExpresion.append(letra1).append("(0-1)#");
        pos++;
        continue;
      }
    NuevaExpresion.append(letra1);
  }

  //Si detecta un paréntesis que abre y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  for (int pos=0; pos<NuevaExpresion.length(); pos++)
  {
    char letra1 = NuevaExpresion.charAt(pos);
    if (letra1=='(')
      if (NuevaExpresion.charAt(pos+1)=='-')
      {
```

```
            NuevaExpresion2.append(letra1).append("(0-1)#");
            pos++;
            continue;
          }
        NuevaExpresion2.append(letra1);
    }

    return NuevaExpresion2.toString();
}
```

C#

```
    /* Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones:
     * 1. Si encuentra un - al inicio le agrega un cero
     * 2. Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
     * 3. Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
     */
    public String ArreglaNegativos(String expresion)
    {
        char letra1, letra2=')';
        StringBuilder NuevaExpresion = new StringBuilder();
        StringBuilder NuevaExpresion2 = new StringBuilder();

        //Si detecta al inicio un - le pone un 0 al inicio
        if (expresion[0]=='-') NuevaExpresion.Append('0');

        //Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
        for (int pos=0; pos<expresion.Length-1; pos++)
        {
            letra1 = expresion[pos];
            letra2 = expresion[pos+1];
            NuevaExpresion.Append(letra1);
            if (letra1=='(' && letra2=='-')  NuevaExpresion.Append('0');
        }
        NuevaExpresion.Append(letra2);

        //Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
        for (int pos=0; pos<NuevaExpresion.Length; pos++)
        {
            letra1 = NuevaExpresion[pos];
            if (letra1=='+' || letra1=='-' || letra1=='*' || letra1=='/' || letra1=='^')
                if (NuevaExpresion[pos+1]=='-')
                {
                    NuevaExpresion2.Append(letra1).Append("(0-1)#");
                    pos++;
                    continue;
                }
            NuevaExpresion2.Append(letra1);
        }
        return NuevaExpresion2.ToString();
    }
```

Visual Basic .NET

```
    ' Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones:
    ' 1. Si encuentra un - al inicio le agrega un cero
    ' 2. Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
    ' 3. Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
    '
    Public Function ArreglaNegativos(expresion As [String]) As [String]
        Dim letra1 As Char, letra2 As Char = ")"
        Dim NuevaExpresion As New Text.StringBuilder()
        Dim NuevaExpresion2 As New Text.StringBuilder()

        'Si detecta al inicio un - le pone un 0 al inicio
        If expresion(0) = "-" Then
            NuevaExpresion.Append("0")
        End If

        'Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
        For pos As Integer = 0 To expresion.Length - 2
            letra1 = expresion(pos)
            letra2 = expresion(pos + 1)
            NuevaExpresion.Append(letra1)
            If letra1 = "(" AndAlso letra2 = "-" Then
                NuevaExpresion.Append("0")
            End If
        Next
        NuevaExpresion.Append(letra2)

        'Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
        For pos As Integer = 0 To NuevaExpresion.Length - 1
            letra1 = NuevaExpresion(pos)
            If letra1 = "+" OrElse letra1 = "-" OrElse letra1 = "*" OrElse letra1 = "/" OrElse letra1 = "^" Then
                If NuevaExpresion(pos + 1) = "-" Then
                    NuevaExpresion2.Append(letra1).Append("(0-1)#")
                    pos += 1
                    Continue For
```

```
                End If
            End If
            NuevaExpresion2.Append(letra1)
        Next
        Return NuevaExpresion2.ToString()
    End Function
```

C++

```
/* Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones:
 * 1. Si encuentra un - al inicio le agrega un cero
 * 2. Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
 * 3. Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
 */
void Evaluar::ArreglaNegativos(char *expresion, char *NuevaExpresion2)
{
  char letra1, letra2=')';
  char *NuevaExpresion = (char *) malloc(strlen(expresion)+5);
  NuevaExpresion[0] = '\0';
  int posNuevaExpr = 0;
  int posNuevaExpr2 = 0;

  //Si detecta al inicio un - le pone un 0 al inicio
  if (expresion[0]=='-') { strcat(NuevaExpresion, "0"); posNuevaExpr++; }

  //Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
  for (int pos=0; pos<strlen(expresion)-1; pos++)
  {
    letra1 = expresion[pos];
    letra2 = expresion[pos+1];
    NuevaExpresion[posNuevaExpr++]=letra1;
    if (letra1=='(' && letra2=='-')  NuevaExpresion[posNuevaExpr++]='0';
  }
  NuevaExpresion[posNuevaExpr++]=letra2;
  NuevaExpresion[posNuevaExpr]='\0';

  //Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
  for (int pos=0; pos<posNuevaExpr; pos++)
        {
            letra1 = NuevaExpresion[pos];
            if (letra1=='+' || letra1=='-' || letra1=='*' || letra1=='/' || letra1=='^')
                if (NuevaExpresion[pos+1]=='-')
                {
                    NuevaExpresion2[posNuevaExpr2++] = letra1;
                    NuevaExpresion2[posNuevaExpr2++] = '(';
                    NuevaExpresion2[posNuevaExpr2++] = '0';
                    NuevaExpresion2[posNuevaExpr2++] = '-';
                    NuevaExpresion2[posNuevaExpr2++] = '1';
                    NuevaExpresion2[posNuevaExpr2++] = ')';
                    NuevaExpresion2[posNuevaExpr2++] = '#';
                    pos++;
                    continue;
                }
            NuevaExpresion2[posNuevaExpr2++] =  letra1;
        }
        NuevaExpresion2[posNuevaExpr2++] = '\0';
        free(NuevaExpresion);
}
```

Object Pascal

```
{ Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones:
 1. Si encuentra un - al inicio le agrega un cero
 2. Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
 3. Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
}
function TEvaluar.ArreglaNegativos(expresion: string): string;
var
  letra1, letra2: char;
  NuevaExpresion, NuevaExpresion2: string;
  pos: integer;
begin
  letra2 :=')';
  NuevaExpresion := '';
  NuevaExpresion2 := '';

  //Si detecta al inicio un - le pone un 0 al inicio
  if expresion[1]='-' then NuevaExpresion := NuevaExpresion + '0';

  //Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
  for pos :=1 to length(expresion)-1 do
  begin
    letra1 := expresion[pos];
    letra2 := expresion[pos+1];
    NuevaExpresion := NuevaExpresion + letra1;
    if (letra1='(') and (letra2='-') then NuevaExpresion := NuevaExpresion + '0';
  end;
  NuevaExpresion := NuevaExpresion + letra2;
```

```
  //Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
  pos := 1;
  while (pos <= length(NuevaExpresion)) do
  begin
    letra1 := NuevaExpresion[pos];
    if (letra1='+') or (letra1='-') or (letra1='*') or (letra1='/') or (letra1='^') then
      if NuevaExpresion[pos+1]='-' then
      begin
        NuevaExpresion2 := NuevaExpresion2 + letra1 + '(0-1)#';
        Inc(pos); Inc(pos);
        continue;
      end;
    NuevaExpresion2 := NuevaExpresion2 + letra1;
    Inc(pos);
  end;
  Result := NuevaExpresion2;
end;
```

JavaScript

```
    /* Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones:
     * 1. Si encuentra un - al inicio le agrega un cero
     * 2. Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
     * 3. Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
     */
    this.ArreglaNegativos = function(expresion)
    {
        var letra1;
        var letra2=')';
        var NuevaExpresion = "";
        var NuevaExpresion2 = "";

        //Si detecta al inicio un - le pone un 0 al inicio
        if (expresion.charAt(0)=='-') NuevaExpresion += '0';

        //Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
        for (var pos=0; pos<expresion.length-1; pos++)
        {
            letra1 = expresion.charAt(pos);
            letra2 = expresion.charAt(pos+1);
            NuevaExpresion += letra1;
            if (letra1=='(' && letra2=='-')  NuevaExpresion += '0';
        }
        NuevaExpresion += letra2;

        //Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
        for (var pos=0; pos<NuevaExpresion.length; pos++)
        {
            letra1 = NuevaExpresion.charAt(pos);
            if (letra1=='+' || letra1=='-' || letra1=='*' || letra1=='/' || letra1=='^')
                if (NuevaExpresion.charAt(pos+1)=='-')
                {
                    NuevaExpresion2 += letra1 + "(0-1)#";
                    pos++;
                    continue;
                }
            NuevaExpresion2 += letra1;
        }
        return NuevaExpresion2;
    }
```

PHP

```
    /* Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones:
     * 1. Si encuentra un - al inicio le agrega un cero
     * 2. Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
     * 3. Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
     */
    public function ArreglaNegativos($expresion)
    {
        $letra2=')';
        $NuevaExpresion = "";
        $NuevaExpresion2 = "";

        //Si detecta al inicio un - le pone un 0 al inicio
        if ($expresion{0}=='-') $NuevaExpresion .= '0';

        //Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
        for ($pos=0; $pos<strlen($expresion)-1; $pos++)
        {
            $letra1 = $expresion{$pos};
            $letra2 = $expresion{$pos+1};
            $NuevaExpresion .= $letra1;
            if ($letra1=='(' && $letra2=='-')  $NuevaExpresion .= '0';
        }
        $NuevaExpresion .= $letra2;
```

```
        //Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
        for ($pos=0; $pos<strlen($NuevaExpresion); $pos++)
        {
            $letra1 = $NuevaExpresion{$pos};
            if ($letra1=='+' || $letra1=='-' || $letra1=='*' || $letra1=='/' || $letra1=='^')
                if ($NuevaExpresion{$pos+1}=='-')
                {
                    $NuevaExpresion2 .= $letra1 . "(0-1)#";
                    $pos++;
                    continue;
                }
            $NuevaExpresion2 .= $letra1;
        }
        return $NuevaExpresion2;
    }
```

El método "Analiza_Expresión"

Tiene como tarea convertir la expresión algebraica en una estructura que:

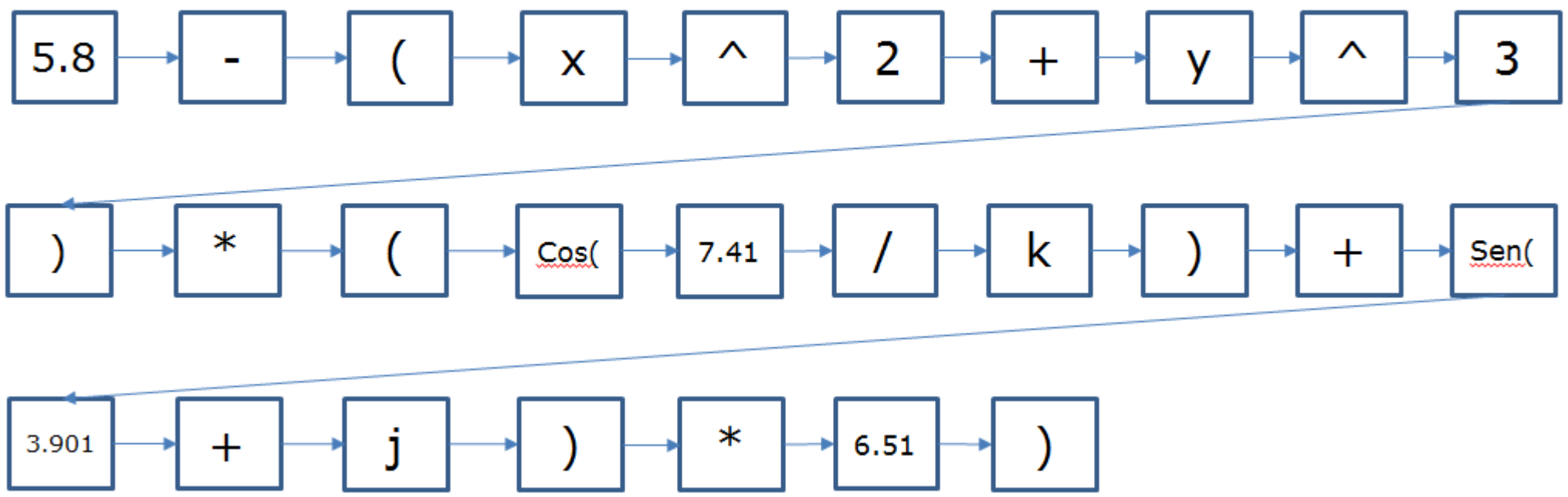
1. Respeta las estrictas reglas del algebra
2. Da soporte a variables (mínimo las 26 de la 'a' a la 'z')
3. Da soporte a los operadores suma(+), resta(-), multiplicación(*), división(/) y potencia(^).
4. Da soporte a paréntesis y anidación de paréntesis
5. Da soporte a funciones como las trigonométricas
6. La estructura creada debe permitir que fácilmente se puedan dar valores numéricos a las variables y sea evaluada rápidamente.

Para crear la estructura, el primer paso es dividir la expresión en una lista donde cada ítem de esta puede ser:

1. Un paréntesis que abre
2. Un paréntesis que cierra
3. Un número
4. Un operador
5. Una variable
6. Una función

Ejemplo:

5.8 – (x^2+y^3) * (cos(7.41 / k) + sen(3.901 + j) * 6.51)



La estructura generada es un ArrayList llamado **PiezaSimple**

Terminada esa estructura el siguiente paso del análisis es convertirla a una serie de operaciones simples del tipo:

Acumulador = Numero/Variable/Acumulador Operador Numero/Variable/Acumulador

Acumulador = Función (Acumulador)

Siguiendo el ejemplo de la ecuación anterior

Cómo se va abreviando la ecuación	**Acumulador encontrado**
(5.8 - (x^2+y^3) * (cos(7.41 / k) + sen(3.901 + j) * 6.51))	[0] = 3.901 + j
(5.8 - (x^2+y^3) * (cos(7.41 / k) + sen([0]) * 6.51))	[1] = sen ([0])
(5.8 - (x^2+y^3) * (cos(7.41 / k) + [1] * 6.51))	[2] = 7.41 / k
(5.8 - (x^2+y^3) * (cos([2]) + [1] * 6.51))	[3] = cos([2])
(5.8 - (x^2+y^3) * ([3] + [1] * 6.51))	[4] = [1] * 6.51
(5.8 - (x^2+y^3) * ([3] + [4]))	[5] = [3] + [4]
(5.8 - (x^2+y^3) * ([5]))	[6] = [5] + 0
(5.8 - (x^2+y^3) * [6])	[7] = x ^ 2
(5.8 - ([7]+y^3) * [6])	[8] = y ^ 3

(5.8 - ([7]+[8]) * [6])	[9] = [7] + [8]
(5.8 - ([9]) * [6])	[10] = [9] + 0
(5.8 - [10] * [6])	[11] = [10] * [6]
(5.8 - [11])	[12] = 5.8 - [11]
([12])	[13] = [12] + 0

Evaluando desde [0] hasta [13] se obtiene el valor numérico de la expresión en forma muy rápida.

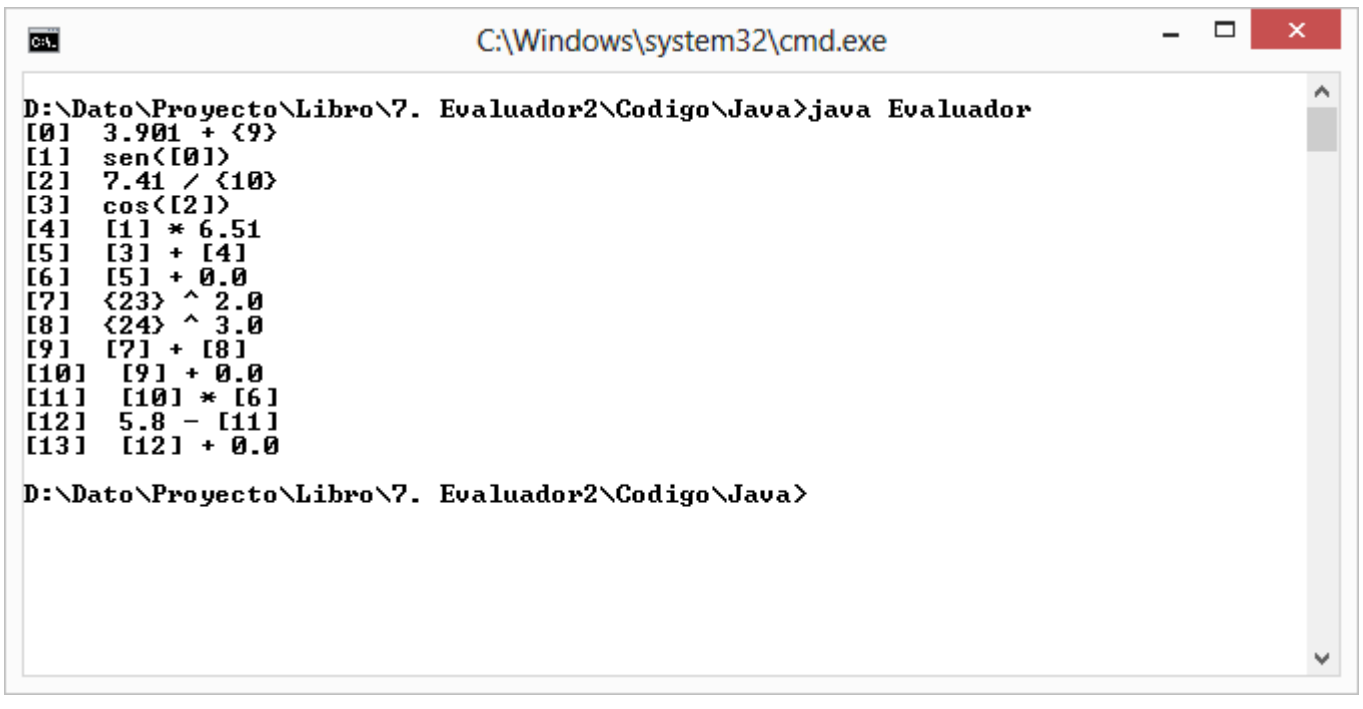
Se genera entonces una nueva estructura (ArrayList) llamado **PiezaEjecuta** que tiene esos acumuladores de [0] a [13].

Cabe aclarar dos puntos del nuevo algoritmo:

1. Toda expresión se le adiciona al inicio un paréntesis que abre y al final un paréntesis que cierra.
2. Un paréntesis interno genera un acumulador adicional, por ejemplo, ([12]) genera a [13] = [12] + 0

Los dos puntos anteriores podrían obviarse y hacer aún más rápida la evaluación, pero implementar esta optimización hace el código de análisis más complejo y la ganancia no sería mucha.

En el software escrito en Java, se muestra cómo se realiza el análisis



Las variables son mostradas así { número } , la variable **j** es la novena letra del alfabeto y por eso aparece así: {9}, la variable **k** es {10}, la variable **x** es {23} y la variable **y** es {24} . Las variables representan posiciones de un arreglo unidimensional de tipo double de tamaño fijo de 25 posiciones para dar cabida a las variables de la a..z . Cuando se dan valores a las variables, simplemente se cambia el valor de una posición de un arreglo unidimensional.

Orden de llamado en Analiza_Expresion

Para llevar esa tarea, requiere llamar los métodos de esta forma:

Java

```
  //Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
  public final void Analizar(String expresion)
  {
    PiezaSimple.clear();
    PiezaEjecuta.clear();
    Generar_Piezas_Simples(expresion);
    Generar_Piezas_Ejecucion();
  }
```

C#

```
  //Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
  public void Analizar(String expresion)
  {
    PiezaSimple.Clear();
    PiezaEjecuta.Clear();
    Generar_Piezas_Simples(expresion);
    Generar_Piezas_Ejecucion();
  }
```

Visual Basic .NET

```
  'Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
  Public Sub Analizar(expresion As [String])
    PiezaSimple.Clear()
    PiezaEjecuta.Clear()
    Generar_Piezas_Simples(expresion)
    Generar_Piezas_Ejecucion()
  End Sub
```

C++

```
//Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
void Evaluar::Analizar(char *expresion)
{
  PiezaSimple.clear();
  PiezaEjecuta.clear();
  Generar_Piezas_Simples(expresion);
  Generar_Piezas_Ejecucion();
  free(exprTmp);
}
```

Object Pascal

```
//Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
procedure TEvaluar.Analizar(expresion: string);
begin
  PiezaSimple.Free;
  PiezaEjecuta.Free;
  PiezaSimple := TObjectList.Create;
  PiezaEjecuta := TObjectList.Create;
  Generar_Piezas_Simples(expresion);
  Generar_Piezas_Ejecucion();
end;
```

JavaScript

```
    //Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
    this.Analizar = function(expresion)
    {
      this.PiezaSimple.length = 0;
      this.PiezaEjecuta.length = 0;
      this.Generar_Piezas_Simples(expresion);
      this.Generar_Piezas_Ejecucion();
    }
```

PHP

```
    //Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
   function Analizar($expresion)
   {
      unset($this->PiezaSimple);
      unset($this->PiezaEjecuta);
      $this->Generar_Piezas_Simples($expresion);
      $this->Generar_Piezas_Ejecucion();
   }
```

Partiendo la expresión y llevándola a la estructura Pieza_Simple

Atributos requeridos

Se requieren los siguientes atributos

Java

```java
    /* Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 */
    private static final int ASCIINUMERO = 48;

    /* Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 */
    private static final int ASCIILETRA = 97;

    /* Las funciones que soporta este evaluador */
    private static final int TAMANOFUNCION = 39;
    private static final String listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

    /* 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;

    //Listado de Piezas de análisis
    private ArrayList<Pieza_Simple> PiezaSimple = new ArrayList<Pieza_Simple>();

    //Listado de Piezas de ejecución
    private ArrayList<Pieza_Ejecuta> PiezaEjecuta = new ArrayList<Pieza_Ejecuta>();
    private int Contador_Acumula = 0;
```

C#

```csharp
    /* Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 */
    private static int ASCIINUMERO = 48;

    /* Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 */
    private static int ASCIILETRA = 97;

    /* Las funciones que soporta este evaluador */
    private static int TAMANOFUNCION = 39;
    private static String listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

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

    //Listado de Piezas de análisis
    private List<Pieza_Simple> PiezaSimple = new List<Pieza_Simple>();

    //Listado de Piezas de ejecución
    private List<Pieza_Ejecuta> PiezaEjecuta = new List<Pieza_Ejecuta>();
    int Contador_Acumula = 0;
```

Visual Basic .NET

```vb
    ' Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7
    Private Shared ASCIINUMERO As Integer = 48

    ' Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1
    Private Shared ASCIILETRA As Integer = 97

    ' Las funciones que soporta este evaluador
    Private Shared TAMANOFUNCION As Integer = 39
    Private Shared listaFunciones As [String] = "sinsencostanabsasnacsatnlogceiexpsqrrcb"

    ' Constantes de los diferentes tipos de datos que tendrán las piezas
    Private Shared ESFUNCION As Integer = 1
    Private Shared ESPARABRE As Integer = 2
    Private Shared ESPARCIERRA As Integer = 3
    Private Shared ESOPERADOR As Integer = 4
    Private Shared ESNUMERO As Integer = 5
    Private Shared ESVARIABLE As Integer = 6

    'Listado de Piezas de análisis
    Private PiezaSimple As New List(Of Pieza_Simple)()

    'Listado de Piezas de ejecución
    Private PiezaEjecuta As New List(Of Pieza_Ejecuta)()
    Private Contador_Acumula As Integer = 0
```

C++

```
    /* Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 */
    static const int ASCIINUMERO = 48;

    /* Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 */
    static const int ASCIILETRA = 97;

    /* Las funciones que soporta este evaluador */
    static const int TAMANOFUNCION = 39;
    static const char *listaFunciones;

    /* Constantes de los diferentes tipos de datos que tendrán las piezas */
    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 Piezas de análisis
    std::vector<Pieza_Simple> PiezaSimple;

    //Listado de Piezas de ejecución
    std::vector<Pieza_Ejecuta> PiezaEjecuta;
    int Contador_Acumula;
```

Object Pascal

```
    { Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 }
    ASCIINUMERO: integer;

    { Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 }
    ASCIILETRA: integer;

    { Las funciones que soporta este evaluador }
    TAMANOFUNCION: integer;
    listaFunciones: string;

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

    //Listado de Piezas de análisis
    PiezaSimple: TobjectList;
    objPiezaSimple: TPieza_Simple;

    //Listado de Piezas de ejecución
    PiezaEjecuta: TobjectList;
    objPiezaEjecuta: TPieza_Ejecuta;

    Contador_Acumula: integer;
```

Nota: Se debe inicializar las constantes en un constructor

JavaScript

```
/* Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 */
this.ASCIINUMERO = 48;

/* Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 */
this.ASCIILETRA = 97;

/* Las funciones que soporta este evaluador */
this.TAMANOFUNCION = 39;
this.listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

/* 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;

//Listado de Piezas de análisis
this.PiezaSimple = [];

//Listado de Piezas de ejecución
this.PiezaEjecuta = [];
this.Contador_Acumula = 0;
```

PHP

```php
/* Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 */
var $ASCIINUMERO = 48;

/* Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 */
var $ASCIILETRA = 97;

/* Las funciones que soporta este evaluador */
var $TAMANOFUNCION = 39;
var $listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

/* 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;

//Listado de Piezas de análisis
var $PiezaSimple = array();

//Listado de Piezas de ejecución
var $PiezaEjecuta = array();
var $Contador_Acumula = 0;
```

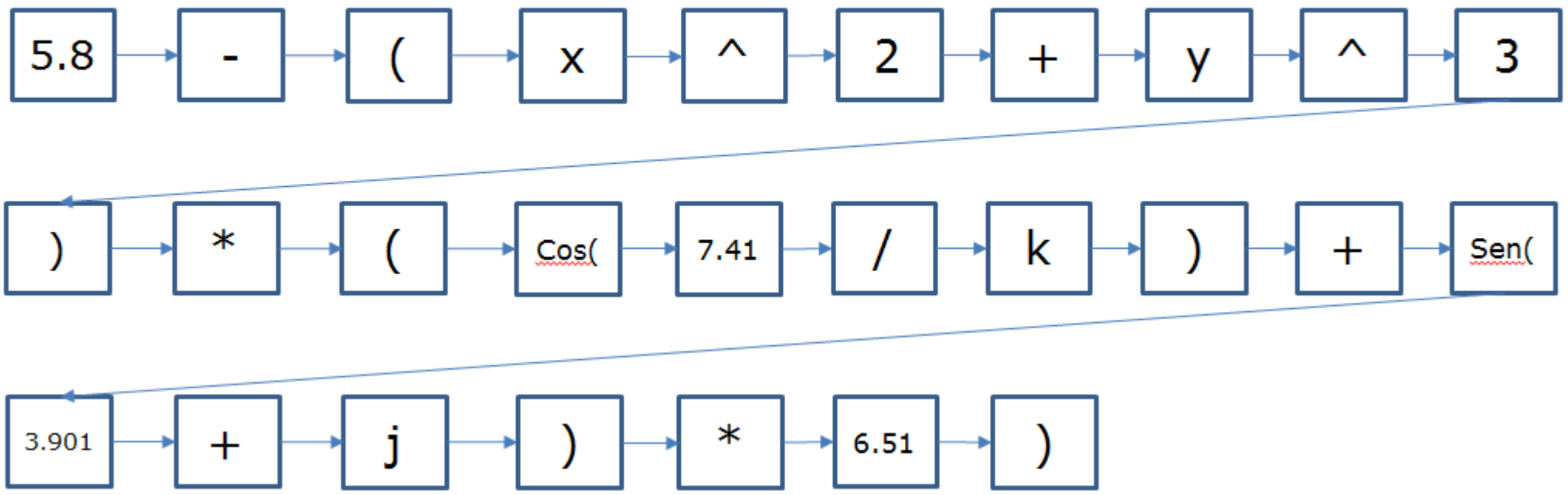
Generando el ArrayList PiezaSimple

Se crea una estructura tipo ArrayList llamada **PiezaSimple** y donde cada ítem de esta puede ser:

1. Un paréntesis que abre
2. Un paréntesis que cierra
3. Un número
4. Un operador
5. Una variable
6. Una función

Ejemplo:

5.8 – (x^2+y^3) * (cos(7.41 / k) + sen(3.901 + j) * 6.51)



Este es el método que genera la lista PiezaSimple

Java

```
//Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
private void Generar_Piezas_Simples(String expresion)
{
  int longExpresion = expresion.length();

  //Variables requeridas para armar un número
  double parteentera = 0;
  double partedecimal = 0;
  double divide = 1;
  boolean entero = true;
  boolean armanumero = false;

  for (int cont = 0; cont < longExpresion; cont++) //Va de letra en letra de la expresión
  {
    char letra = expresion.charAt(cont);
    if (letra == '.')  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
      entero = false;
    else if (letra >= '0' && letra <= '9')  //Si es un número, entonces lo va armando
    {
      armanumero = true;
      if (entero)
        parteentera = parteentera * 10 + letra - ASCIINUMERO; //La parte entera del número
      else
      {
        divide *= 10;
        partedecimal = partedecimal * 10 + letra - ASCIINUMERO; //La parte decimal del número
      }
    }
    else
    {
      if (armanumero) //Si tenía armado un número, entonces crea la pieza ESNUMERO
      {
        PiezaSimple.add(new Pieza_Simple(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0));
        parteentera = 0;
        partedecimal = 0;
        divide = 1;
        entero = true;
        armanumero = false;
      }

      if (letra == '+' || letra == '-' || letra == '*' || letra == '/' || letra == '^' || letra == '#') PiezaSimple.add(new
Pieza_Simple(ESOPERADOR, 0, letra, 0, 0));
      else if (letra == '(') PiezaSimple.add(new Pieza_Simple(ESPARABRE, 0, '0', 0, 0)); //¿Es paréntesis que abre?
      else if (letra == ')') PiezaSimple.add(new Pieza_Simple(ESPARCIERRA, 0, '0', 0, 0));//¿Es paréntesis que cierra?
```

```
        else if (letra >= 'a' && letra <= 'z') //¿Es variable o función?
        {
          /* Detecta si es una función porque tiene dos letras seguidas */
          if (cont < longExpresion - 1)
          {
            char letra2 = expresion.charAt(cont + 1); /* Chequea si el siguiente carácter es una letra, dado el caso es una
función */
            if (letra2 >= 'a' && letra2 <= 'z')
            {
              char letra3 = expresion.charAt(cont + 2);
              int funcionDetectada = 1;  /* Identifica la función */
              for (int funcion = 0; funcion <= TAMANOFUNCION; funcion += 3)
              {
                if (letra == listaFunciones.charAt(funcion)
                    && letra2 == listaFunciones.charAt(funcion + 1)
                    && letra3 == listaFunciones.charAt(funcion + 2))
                  break;
                funcionDetectada++;
              }
              PiezaSimple.add(new Pieza_Simple(ESFUNCION, funcionDetectada, '0', 0, 0));  //Adiciona función a la lista
              cont += 3; /* Mueve tres caracteres  sin(  [s][i][n][(] */
            }
            else /* Es una variable, no una función */
              PiezaSimple.add(new Pieza_Simple(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA));
          }
          else /* Es una variable, no una función */
            PiezaSimple.add(new Pieza_Simple(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA));
        }
      }
    }
    if (armanumero) PiezaSimple.add(new Pieza_Simple(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0));
  }
```

C#

```
  //Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
  private void Generar_Piezas_Simples(String expresion)
  {
    int longExpresion = expresion.Length;

    //Variables requeridas para armar un número
    double parteentera = 0;
    double partedecimal = 0;
    double divide = 1;
    bool entero = true;
    bool armanumero = false;

    for (int cont = 0; cont < longExpresion; cont++) //Va de letra en letra de la expresión
    {
      char letra = expresion[cont];
      if (letra == '.')  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
        entero = false;
      else if (letra >= '0' && letra <= '9')  //Si es un número, entonces lo va armando
      {
        armanumero = true;
        if (entero)
          parteentera = parteentera * 10 + letra - ASCIINUMERO; //La parte entera del número
        else
        {
          divide *= 10;
          partedecimal = partedecimal * 10 + letra - ASCIINUMERO; //La parte decimal del número
        }
      }
      else
      {
        if (armanumero) //Si tenía armado un número, entonces crea la pieza ESNUMERO
        {
          PiezaSimple.Add(new Pieza_Simple(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0, 0));
          parteentera = 0;
          partedecimal = 0;
          divide = 1;
          entero = true;
          armanumero = false;
        }

        if (letra == '+' || letra == '-' || letra == '*' || letra == '/' || letra == '^' || letra == '#') PiezaSimple.Add(new
Pieza_Simple(ESOPERADOR, 0, letra, 0, 0, 0));
        else if (letra == '(') PiezaSimple.Add(new Pieza_Simple(ESPARABRE, 0, '0', 0, 0, 0)); //¿Es paréntesis que abre?
        else if (letra == ')') PiezaSimple.Add(new Pieza_Simple(ESPARCIERRA, 0, '0', 0, 0, 0));//¿Es paréntesis que cierra?
        else if (letra >= 'a' && letra <= 'z') //¿Es variable o función?
        {
          /* Detecta si es una función porque tiene dos letras seguidas */
          if (cont < longExpresion - 1)
          {
            char letra2 = expresion[cont + 1]; /* Chequea si el siguiente carácter es una letra, dado el caso es una función */
            if (letra2 >= 'a' && letra2 <= 'z')
            {
              char letra3 = expresion[cont + 2];
              int funcionDetectada = 1;  /* Identifica la función */
              for (int funcion = 0; funcion <= TAMANOFUNCION; funcion += 3)
```

```
            {
              if (letra == listaFunciones[funcion]
                 && letra2 == listaFunciones[funcion + 1]
                 && letra3 == listaFunciones[funcion + 2])
                 break;
              funcionDetectada++;
            }
            PiezaSimple.Add(new Pieza_Simple(ESFUNCION, funcionDetectada, '0', 0, 0, 0));  //Adiciona función a la lista
            cont += 3; /* Mueve tres caracteres  sin(  [s][i][n][(] */
          }
          else /* Es una variable, no una función */
             PiezaSimple.Add(new Pieza_Simple(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA, 0));
        }
        else /* Es una variable, no una función */
          PiezaSimple.Add(new Pieza_Simple(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA, 0));
      }
    }
  }
  if (armanumero) PiezaSimple.Add(new Pieza_Simple(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0, 0));
}
```

Visual Basic .NET

```
'Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
Private Sub Generar_Piezas_Simples(expresion As [String])
  Dim longExpresion As Integer = expresion.Length

  'Variables requeridas para armar un número
  Dim parteentera As Double = 0
  Dim partedecimal As Double = 0
  Dim divide As Double = 1
  Dim entero As Boolean = True
  Dim armanumero As Boolean = False

  For cont As Integer = 0 To longExpresion - 1
    'Va de letra en letra de la expresión
    Dim letra As Char = expresion(cont)
    If letra = "." Then
      'Si letra es . entonces el resto de digitos leídos son la parte decimal del número
      entero = False
    ElseIf letra >= "0" AndAlso letra <= "9" Then
      'Si es un número, entonces lo va armando
      armanumero = True
      If entero Then
        parteentera = parteentera * 10 + Asc(letra) - ASCIINUMERO
      Else
        'La parte entera del número
        divide *= 10
        'La parte decimal del número
        partedecimal = partedecimal * 10 + Asc(letra) - ASCIINUMERO
      End If
    Else
      If armanumero Then
        'Si tenía armado un número, entonces crea la pieza ESNUMERO
        PiezaSimple.Add(New Pieza_Simple(ESNUMERO, 0, "0", parteentera + partedecimal / divide, 0, 0))
        parteentera = 0
        partedecimal = 0
        divide = 1
        entero = True
        armanumero = False
      End If

      If letra = "+" OrElse letra = "-" OrElse letra = "*" OrElse letra = "/" OrElse letra = "^" OrElse letra = "#" Then
        PiezaSimple.Add(New Pieza_Simple(ESOPERADOR, 0, letra, 0, 0, 0))
      ElseIf letra = "(" Then
        PiezaSimple.Add(New Pieza_Simple(ESPARABRE, 0, "0", 0, 0, 0))
        '¿Es paréntesis que abre?
      ElseIf letra = ")" Then
        PiezaSimple.Add(New Pieza_Simple(ESPARCIERRA, 0, "0", 0, 0, 0))
        '¿Es paréntesis que cierra?
      ElseIf letra >= "a" AndAlso letra <= "z" Then
        '¿Es variable o función?
        ' Detecta si es una función porque tiene dos letras seguidas

        If cont < longExpresion - 1 Then
          Dim letra2 As Char = expresion(cont + 1)
          ' Chequea si el siguiente carácter es una letra, dado el caso es una función
          If letra2 >= "a" AndAlso letra2 <= "z" Then
            Dim letra3 As Char = expresion(cont + 2)
            Dim funcionDetectada As Integer = 1
            ' Identifica la función
            For funcion As Integer = 0 To TAMANOFUNCION Step 3
              If letra = listaFunciones(funcion) AndAlso letra2 = listaFunciones(funcion + 1) AndAlso letra3 =
listaFunciones(funcion + 2) Then
                Exit For
              End If
              funcionDetectada += 1
            Next
            PiezaSimple.Add(New Pieza_Simple(ESFUNCION, funcionDetectada, "0", 0, 0, 0))
            'Adiciona función a la lista
```

```
            ' Mueve tres caracteres  sin(  [s][i][n][(]
            cont += 3
          Else
            ' Es una variable, no una función
            PiezaSimple.Add(New Pieza_Simple(ESVARIABLE, 0, "0", 0, Asc(letra) - ASCIILETRA, 0))
          End If
        Else
          ' Es una variable, no una función
          PiezaSimple.Add(New Pieza_Simple(ESVARIABLE, 0, "0", 0, Asc(letra) - ASCIILETRA, 0))
        End If
      End If
    End If
  Next
  If armanumero Then
    PiezaSimple.Add(New Pieza_Simple(ESNUMERO, 0, "0", parteentera + partedecimal / divide, 0, 0))
  End If
End Sub
```

C++

```
//Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
void Evaluar::Generar_Piezas_Simples(char *expresion)
{
  int longExpresion = strlen(expresion);

  //Variables requeridas para armar un número
  double parteentera = 0;
  double partedecimal = 0;
  double divide = 1;
  bool entero = true;
  bool armanumero = false;

  for (int cont = 0; cont < longExpresion; cont++) //Va de letra en letra de la expresión
  {
    char letra = expresion[cont];
    if (letra == '.')  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
      entero = false;
    else if (letra >= '0' && letra <= '9')  //Si es un número, entonces lo va armando
    {
      armanumero = true;
      if (entero)
        parteentera = parteentera * 10 + letra - ASCIINUMERO; //La parte entera del número
      else
      {
        divide *= 10;
        partedecimal = partedecimal * 10 + letra - ASCIINUMERO; //La parte decimal del número
      }
    }
    else
    {
      if (armanumero) //Si tenía armado un número, entonces crea la pieza ESNUMERO
      {
        Pieza_Simple objeto(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0);
        PiezaSimple.push_back(objeto);
        parteentera = 0;
        partedecimal = 0;
        divide = 1;
        entero = true;
        armanumero = false;
      }

      if (letra == '+' || letra == '-' || letra == '*' || letra == '/' || letra == '^' || letra == '#'){ Pieza_Simple
objeto(ESOPERADOR, 0, letra, 0, 0); PiezaSimple.push_back(objeto); }
      else if (letra == '('){ Pieza_Simple objeto(ESPARABRE, 0, '0', 0, 0); PiezaSimple.push_back(objeto); }//¿Es paréntesis que
abre?
      else if (letra == ')'){ Pieza_Simple objeto(ESPARCIERRA, 0, '0', 0, 0); PiezaSimple.push_back(objeto); }//¿Es paréntesis
que cierra?
      else if (letra >= 'a' && letra <= 'z') //¿Es variable o función?
      {
        /* Detecta si es una función porque tiene dos letras seguidas */
        if (cont < longExpresion - 1)
        {
          char letra2 = expresion[cont + 1]; /* Chequea si el siguiente carácter es una letra, dado el caso es una función */
          if (letra2 >= 'a' && letra2 <= 'z')
          {
            char letra3 = expresion[cont + 2];
            int funcionDetectada = 1;  /* Identifica la función */
            for (int funcion = 0; funcion <= TAMANOFUNCION; funcion += 3)
            {
              if (letra == listaFunciones[funcion]
                && letra2 == listaFunciones[funcion + 1]
                && letra3 == listaFunciones[funcion + 2])
                break;
              funcionDetectada++;
            }
            Pieza_Simple objeto(ESFUNCION, funcionDetectada, '0', 0, 0);
            PiezaSimple.push_back(objeto);  //Adiciona función a la lista
            cont += 3; /* Mueve tres caracteres  sin(  [s][i][n][(] */
          }
          else /* Es una variable, no una función */
```

```
          {
            Pieza_Simple objeto(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA);
            PiezaSimple.push_back(objeto);
          }
        }
        else /* Es una variable, no una función */
        {
          Pieza_Simple objeto(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA);
          PiezaSimple.push_back(objeto);
        }
      }
    }
  }
  if (armanumero) { Pieza_Simple objeto(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0); PiezaSimple.push_back(objeto);  }
}
```

Object Pascal

```
//Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
procedure TEvaluar.Generar_Piezas_Simples(expresion: string);
var
  longExpresion, cont, funciondetectada, funcion: integer;
  parteentera, partedecimal, divide: double;
  entero, armanumero: boolean;
  letra, letra2, letra3: char;
begin
  longExpresion := length(expresion);

  //Variables requeridas para armar un número
  parteentera := 0;
  partedecimal := 0;
  divide := 1;
  entero := true;
  armanumero := false;

  cont := 1;
  while cont <= longExpresion do  //Va de letra en letra de la expresión
  begin
    letra := expresion[cont];
    if letra = '.' then  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
    begin
      entero := false
    end
    else if (letra >= '0') and (letra <= '9') then  //Si es un número, entonces lo va armando
    begin
      armanumero := true;
      if (entero) then
        parteentera := parteentera * 10 + ord(letra) - ASCIINUMERO //La parte entera del número
      else
      begin
        divide := divide * 10;
        partedecimal := partedecimal * 10 + ord(letra) - ASCIINUMERO; //La parte decimal del número
      end;
    end
    else
    begin
      if armanumero = true then //Si tenía armado un número, entonces crea la pieza ESNUMERO
      begin
        objPiezaSimple := TPieza_Simple.Create(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0, 0);
        PiezaSimple.Add(objPiezaSimple);
        parteentera := 0;
        partedecimal := 0;
        divide := 1;
        entero := true;
        armanumero := false;
      end;
      if (letra = '+') or (letra = '-') or (letra = '*') or (letra = '/') or (letra = '^') or (letra = '#') then
      begin
        objPiezaSimple := TPieza_Simple.Create(ESOPERADOR, 0, letra, 0, 0, 0);
        PiezaSimple.Add(objPiezaSimple);
      end
      else if (letra = '(') then
      begin
        objPiezaSimple := TPieza_Simple.Create(ESPARABRE, 0, '0', 0, 0, 0);
        PiezaSimple.Add(objPiezaSimple);
      end
      else if (letra = ')') then
      begin
        objPiezaSimple := TPieza_Simple.Create(ESPARCIERRA, 0, '0', 0, 0, 0);
        PiezaSimple.Add(objPiezaSimple);
      end
      else if (letra >= 'a') and (letra <= 'z') then //¿Es variable o función?
      begin
        // Detecta si es una función porque tiene dos letras seguidas
        if (cont < longExpresion - 1) then
        begin
          letra2 := expresion[cont + 1]; // Chequea si el siguiente carácter es una letra, dado el caso es una función
          if (letra2 >= 'a') and (letra2 <= 'z') then
          begin
            letra3 := expresion[cont + 2];
```

```
            funcionDetectada := 1;  // Identifica la función
            funcion := 1;
            while (funcion <= TAMANOFUNCION) do
            begin
              if (letra = listaFunciones[funcion]) and (letra2 = listaFunciones[funcion + 1]) and (letra3 =
listaFunciones[funcion + 2]) then break;
              Inc(funcionDetectada);
              funcion := funcion + 3;
            end;
            objPiezaSimple := TPieza_Simple.Create(ESFUNCION, funcionDetectada, '0', 0, 0, 0); //Adiciona función a la lista
            PiezaSimple.Add(objPiezaSimple);
            cont := cont + 3; // Mueve tres caracteres  sin(  [s][i][n][(]
          end
        else // Es una variable, no una función
        begin
          objPiezaSimple := TPieza_Simple.Create(ESVARIABLE, 0, '0', 0, ord(letra) - ASCIILETRA, 0);
          PiezaSimple.Add(objPiezaSimple);
        end
      end
      else // Es una variable, no una función
        begin
          objPiezaSimple := TPieza_Simple.Create(ESVARIABLE, 0, '0', 0, ord(letra) - ASCIILETRA, 0);
          PiezaSimple.Add(objPiezaSimple);
        end
      end;
    end;
    Inc(cont);
  end;
  if armanumero then
  begin
    objPiezaSimple := TPieza_Simple.Create(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0, 0);
    PiezaSimple.Add(objPiezaSimple);
  end
end;
```

JavaScript

```
    //Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
    this.Generar_Piezas_Simples = function(expresion)
    {
      var longExpresion = expresion.length;
      var NumeroPiezaSimple = 0;

      //Variables requeridas para armar un número
      var parteentera = 0;
      var partedecimal = 0;
      var divide = 1;
      var entero = true;
      var armanumero = false;

      for (var cont = 0; cont < longExpresion; cont++) //Va de letra en letra de la expresión
      {
        var letra = expresion.charAt(cont);
        if (letra == '.')  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
          entero = false;
        else if (letra >= '0' && letra <= '9')  //Si es un número, entonces lo va armando
        {
          armanumero = true;
          if (entero)
            parteentera = parteentera * 10 + parseFloat(letra); //La parte entera del número
          else
          {
            divide *= 10;
            partedecimal = partedecimal * 10 + parseFloat(letra); //La parte decimal del número
          }
        }
        else
        {
          if (armanumero) //Si tenía armado un número, entonces crea la pieza ESNUMERO
          {
            objeto = new Pieza_Simple();
            objeto.ConstructorPieza_Simple(this.ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0, 0);
            this.PiezaSimple[NumeroPiezaSimple++] = objeto;
            parteentera = 0;
            partedecimal = 0;
            divide = 1;
            entero = true;
            armanumero = false;
          }

          if (letra == '+' || letra == '-' || letra == '*' || letra == '/' || letra == '^' || letra == '#') { objeto = new
Pieza_Simple(); objeto.ConstructorPieza_Simple(this.ESOPERADOR, 0, letra, 0, 0, 0); this.PiezaSimple[NumeroPiezaSimple++] =
objeto; }
          else if (letra == '(') { objeto = new Pieza_Simple(); objeto.ConstructorPieza_Simple(this.ESPARABRE, 0, '0', 0, 0, 0);
this.PiezaSimple[NumeroPiezaSimple++] = objeto; } //¿Es paréntesis que abre?
          else if (letra == ')') { objeto = new Pieza_Simple(); objeto.ConstructorPieza_Simple(this.ESPARCIERRA, 0, '0', 0, 0,
0); this.PiezaSimple[NumeroPiezaSimple++] = objeto; }//¿Es paréntesis que cierra?
          else if (letra >= 'a' && letra <= 'z') //¿Es variable o función?
          {
            /* Detecta si es una función porque tiene dos letras seguidas */
```

```
            if (cont < longExpresion - 1)
            {
              letra2 = expresion.charAt(cont + 1); /* Chequea si el siguiente carácter es una letra, dado el caso es una función
*/
              if (letra2 >= 'a' && letra2 <= 'z')
              {
                letra3 = expresion.charAt(cont + 2);
                funcionDetectada = 1;  /* Identifica la función */
                for (funcion = 0; funcion <= this.TAMANOFUNCION; funcion += 3)
                {
                  if (letra == this.listaFunciones.charAt(funcion)
                    && letra2 == this.listaFunciones.charAt(funcion + 1)
                    && letra3 == this.listaFunciones.charAt(funcion + 2))
                    break;
                  funcionDetectada++;
                }
                objeto = new Pieza_Simple();
                objeto.ConstructorPieza_Simple(this.ESFUNCION, funcionDetectada, '0', 0, 0, 0);  //Adiciona función a la lista
                this.PiezaSimple[NumeroPiezaSimple++] = objeto;
                cont += 3; /* Mueve tres caracteres  sin(  [s][i][n][(] */
              }
              else /* Es una variable, no una función */
              {
                objeto = new Pieza_Simple();
                objeto.ConstructorPieza_Simple(this.ESVARIABLE, 0, '0', 0, letra.charCodeAt(0) - this.ASCIILETRA, 0);
                this.PiezaSimple[NumeroPiezaSimple++] = objeto;
              }
            }
            else /* Es una variable, no una función */
            {
              objeto = new Pieza_Simple();
              objeto.ConstructorPieza_Simple(this.ESVARIABLE, 0, '0', 0, letra.charCodeAt(0) - this.ASCIILETRA, 0);
              this.PiezaSimple[NumeroPiezaSimple++] = objeto;
            }
          }
        }
        if (armanumero) { objeto = new Pieza_Simple(); objeto.ConstructorPieza_Simple(this.ESNUMERO, 0, '0',
parteentera+partedecimal/divide, 0, 0);  this.PiezaSimple[NumeroPiezaSimple++] = objeto; }
      }
```

PHP

```
   //Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
   function Generar_Piezas_Simples($expresion)
   {
      $longExpresion = strlen($expresion);

      //Variables requeridas para armar un número
      $parteentera = 0;
      $partedecimal = 0;
      $divide = 1;
      $entero = true;
      $armanumero = false;

      for ($cont = 0; $cont < $longExpresion; $cont++) //Va de letra en letra de la expresión
      {
         $letra = $expresion{$cont};

         if ($letra == '.')  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
          $entero = false;
         else if ($letra >= '0' && $letra <= '9')  //Si es un número, entonces lo va armando
         {
          $armanumero = true;
          if ($entero)
             $parteentera = $parteentera * 10 + ord($letra) - $this->ASCIINUMERO; //La parte entera del número
          else
          {
             $divide *= 10;
             $partedecimal = $partedecimal * 10 + ord($letra) - $this->ASCIINUMERO; //La parte decimal del número
          }
         }
         else
         {
          if ($armanumero) //Si tenía armado un número, entonces crea la pieza ESNUMERO
          {
             $objeto = new Pieza_Simple();
             $objeto->ConstructorPiezaSimple($this->ESNUMERO, 0, '0', $parteentera+$partedecimal/$divide, 0);
             $this->PiezaSimple[] = $objeto;
             $parteentera = 0;
             $partedecimal = 0;
             $divide = 1;
             $entero = true;
             $armanumero = false;
          }

          if ($letra == '+' || $letra == '-' || $letra == '*' || $letra == '/' || $letra == '^' || $letra == '#')
          {
             $objeto = new Pieza_Simple();
             $objeto->ConstructorPiezaSimple($this->ESOPERADOR, 0, $letra, 0, 0);
```

```
            $this->PiezaSimple[] = $objeto;
         }
         else if ($letra == '(')
         {
            $objeto = new Pieza_Simple();
            $objeto->ConstructorPiezaSimple($this->ESPARABRE, 0, '0', 0, 0); //¿Es paréntesis que abre?
            $this->PiezaSimple[] = $objeto;
         }
         else if ($letra == ')')
         {
            $objeto = new Pieza_Simple();
            $objeto->ConstructorPiezaSimple($this->ESPARCIERRA, 0, '0', 0, 0);//¿Es paréntesis que cierra?
            $this->PiezaSimple[] = $objeto;
         }
         else if ($letra >= 'a' && $letra <= 'z') //¿Es variable o función?
         {
            /* Detecta si es una función porque tiene dos letras seguidas */
            if ($cont < $longExpresion - 1)
            {
               $letra2 = $expresion{$cont + 1}; /* Chequea si el siguiente carácter es una letra, dado el caso es una función
*/
               if ($letra2 >= 'a' && $letra2 <= 'z')
               {
                  $letra3 = $expresion{$cont + 2};
                  $funcionDetectada = 1;  /* Identifica la función */
                  for ($funcion = 0; $funcion <= $this->TAMANOFUNCION; $funcion += 3)
                  {
                   if ($letra == $this->listaFunciones{$funcion}
                         && $letra2 == $this->listaFunciones{$funcion + 1}
                         && $letra3 == $this->listaFunciones{$funcion + 2})
                      break;
                   $funcionDetectada++;
                  }
                  $objeto = new Pieza_Simple();
                  $objeto->ConstructorPiezaSimple($this->ESFUNCION, $funcionDetectada, '0', 0, 0);  //Adiciona función a la
lista
                  $this->PiezaSimple[] = $objeto;
                  $cont += 3; /* Mueve tres caracteres  sin(  [s][i][n][(] */
               }
               else /* Es una variable, no una función */
               {
                  $objeto = new Pieza_Simple();
                  $objeto->ConstructorPiezaSimple($this->ESVARIABLE, 0, '0', 0, ord($letra) - $this->ASCIILETRA);
                  $this->PiezaSimple[] = $objeto;
               }
            }
            else /* Es una variable, no una función */
            {
               $objeto = new Pieza_Simple();
               $objeto->ConstructorPiezaSimple($this->ESVARIABLE, 0, '0', 0, ord($letra) - $this->ASCIILETRA);
               $this->PiezaSimple[] = $objeto;
            }
          }
        }
      }
      if ($armanumero)
      {
         $objeto = new Pieza_Simple();
         $objeto->ConstructorPiezaSimple($this->ESNUMERO, 0, '0', $parteentera+$partedecimal/$divide, 0);
         $this->PiezaSimple[] = $objeto;
      }
   }
```

¿Qué es cada Nodo en PiezaSimple?

Se ha generado un ArrayList, ¿y en qué consiste cada nodo de ese ArrayList? Se responde con la siguiente clase

Java

```java
public class Pieza_Simple
{
  private int tipo; //Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
  private int funcion; //Que función es seno/coseno/tangente/sqrt
  private char operador; // +, -, *, /, ^
  private double numero; //Número real de la expresión
  private int variableAlgebra; //Variable de la expresión
  private int acumula;  //Indice de la microexpresión

  public final int getTipo() {  return this.tipo;  }
  public final int getFuncion() { return this.funcion; }
  public final char getOperador() { return this.operador; }
  public final double getNumero() { return this.numero; }
  public final int getVariable() { return this.variableAlgebra; }
  public final int getAcumula() { return this.acumula; }
  public final void setAcumula(int acumula) { this.tipo = 7; this.acumula = acumula; }

  public Pieza_Simple(int tipo, int funcion, char operador, double numero, int variable)
  {
    this.tipo = tipo;
    this.funcion = funcion;
    this.operador = operador;
    this.variableAlgebra = variable;
    this.acumula = 0;
    this.numero = numero;
  }
}
```

C#

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EvaluadorCS
{
  class Pieza_Simple
  {
    private int tipo; //Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
    private int funcion; //Que función es seno/coseno/tangente/sqrt
    private char operador; // +, -, *, /, ^
    private double numero; //Número real de la expresión
    private int variableAlgebra; //Variable de la expresión
    private int acumula;  //Indice de la microexpresión

    public int getTipo() { return this.tipo; }
    public int getFuncion() { return this.funcion; }
    public char getOperador() { return this.operador; }
    public double getNumero() { return this.numero; }
    public int getVariable() { return this.variableAlgebra; }
    public int getAcumula() { return this.acumula; }
    public void setAcumula(int acumula) { this.tipo = 7; this.acumula = acumula; }

    public Pieza_Simple(int tipo, int funcion, char operador, double numero, int variable, int acumula)
    {
      this.tipo = tipo;
      this.funcion = funcion;
      this.operador = operador;
      this.variableAlgebra = variable;
      this.acumula = acumula;
      this.numero = numero;
    }
  }
}
```

Visual Basic .NET

```vb
Public Class Pieza_Simple
    'Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
    Private tipo As Integer

    'Que función es seno/coseno/tangente/sqrt
    Private funcion As Integer

    ' +, -, *, /, ^
    Private operador As Char

    'Número real de la expresión
    Private numero As Double
```

```
'Variable de la expresión
Private variableAlgebra As Integer

'Indice de la microexpresión
Private acumula As Integer

Public Function getTipo() As Integer
    Return Me.tipo
End Function
Public Function getFuncion() As Integer
    Return Me.funcion
End Function
Public Function getOperador() As Char
    Return Me.operador
End Function
Public Function getNumero() As Double
    Return Me.numero
End Function
Public Function getVariable() As Integer
    Return Me.variableAlgebra
End Function
Public Function getAcumula() As Integer
    Return Me.acumula
End Function
Public Sub setAcumula(acumula As Integer)
    Me.tipo = 7
    Me.acumula = acumula
End Sub

    Public Sub New(tipo As Integer, funcion As Integer, operador As Char, numero As Double, variable As Integer, acumula As
Integer)
        Me.tipo = tipo
        Me.funcion = funcion
        Me.operador = operador
        Me.variableAlgebra = variable
        Me.acumula = acumula
        Me.numero = numero
    End Sub
End Class
```

C++

```
#include "Pieza_Simple.h"

Pieza_Simple::Pieza_Simple(int tipo, int funcion, char operador, double numero, int variable)
{
  this->tipo = tipo;
  this->funcion = funcion;
  this->operador = operador;
  this->variableAlgebra = variable;
  this->acumula = acumula;
  this->numero = numero;
}

int Pieza_Simple::getTipo() {  return this->tipo;  }
int Pieza_Simple::getFuncion() { return this->funcion; }
char Pieza_Simple::getOperador() { return this->operador; }
double Pieza_Simple::getNumero() { return this->numero; }
int Pieza_Simple::getVariable() { return this->variableAlgebra; }
int Pieza_Simple::getAcumula() { return this->acumula; }
void Pieza_Simple::setAcumula(int acumula) { this->tipo = 7; this->acumula = acumula; }
```

Object Pascal

```
unit Pieza_Simple;

interface
type
  TPieza_Simple = class
  private
     tipo: integer; //Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
     funcion: integer; //Que función es seno/coseno/tangente/sqrt
     operador: char; // +, -, *, /, ^
     numero: double; //Número real de la expresión
     variableAlgebra: integer; //Variable de la expresión
     acumula: integer;  //Indice de la microexpresión
   public
    Constructor Create(tipo: integer; funcion: integer; operador: char; numero: double; variable: integer; acumula: integer);
    function getTipo(): integer;
    function getFuncion(): integer;
    function getOperador(): char;
    function getNumero(): double;
    function getVariable(): integer;
    function getAcumula(): integer;
    procedure setAcumula(acumula: integer);
   end;

implementation
```

```
Constructor TPieza_Simple.Create(tipo: integer; funcion: integer; operador: char; numero: double; variable: integer; acumula:
integer);
begin
  self.tipo := tipo;
  self.funcion := funcion;
  self.operador := operador;
  self.variableAlgebra := variable;
  self.acumula := acumula;
  self.numero := numero;
end;

function TPieza_Simple.getTipo(): integer;
begin
  getTipo := tipo;
end;

function TPieza_Simple.getFuncion(): integer;
begin
  getFuncion := funcion;
end;

function TPieza_Simple.getOperador(): char;
begin
  getOperador := operador;
end;

function TPieza_Simple.getNumero(): double;
begin
  getNumero := numero;
end;

function TPieza_Simple.getVariable(): integer;
begin
  getVariable := variableAlgebra;
end;

function TPieza_Simple.getAcumula(): integer;
begin
  getAcumula := acumula;
end;

procedure TPieza_Simple.setAcumula(acumula: integer);
begin
  self.tipo := 7;
  self.acumula := acumula;
end;

end.
```

JavaScript

```
function Pieza_Simple()
{
    this.tipo; //Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
    this.funcion; //Que función es seno/coseno/tangente/sqrt
    this.operador; // +, -, *, /, ^
    this.numero; //Número real de la expresión
    this.variableAlgebra; //Variable de la expresión
    this.acumula;  //Indice de la microexpresión

    this.getTipo = function()
    {
      return this.tipo;
    }

    this.getFuncion = function()
    {
      return this.funcion;
    }

    this.getOperador = function()
    {
      return this.operador;
    }

    this.getNumero = function()
    {
       return this.numero;
    }

    this.getVariable = function()
    {
      return this.variableAlgebra;
    }

    this.getAcumula = function()
    {
      return this.acumula;
    }
```

```
    this.setAcumula = function(acumula)
    {
      this.tipo = 7;
      this.acumula = acumula;
    }

    this.ConstructorPieza_Simple = function(tipo, funcion, operador, numero, variable)
    {
      this.tipo = tipo;
      this.funcion = funcion;
      this.operador = operador;
      this.variableAlgebra = variable;
      this.acumula = 0;
      this.numero = numero;
    }
}
```

PHP

```
<?php
class Pieza_Simple
{
      var $tipo; //Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
      var $funcion; //Que función es seno/coseno/tangente/sqrt
      var $operador; // +, -, *, /, ^
      var $numero; //Número real de la expresión
      var $variableAlgebra; //Variable de la expresión
      var $acumula;  //Indice de la microexpresión

      public function getTipo() { return $this->tipo;}
      public function getFuncion() { return $this->funcion; }
      public function getOperador() {     return $this->operador; }
      public function getNumero() { return $this->numero;  }
      public function getVariable() {     return $this->variableAlgebra;      }
      public function getAcumula() { return $this->acumula; }
      public function setAcumula($acumula) { $this->tipo = 7; $this->acumula = $acumula; }

      public function ConstructorPiezaSimple($tipo, $funcion, $operador, $numero, $variable)
      {
            $this->tipo = $tipo;
            $this->funcion = $funcion;
            $this->operador = $operador;
            $this->variableAlgebra = $variable;
            $this->acumula = 0;
            $this->numero = $numero;
      }
}
?>
```

Usando el ArrayList PiezaSimple para generar el ArrayList PiezaEjecuta

Terminado PiezaSimple, el siguiente paso del análisis es convertirla a una serie de operaciones simples del tipo:

Acumulador = Numero/Variable/Acumulador Operador Numero/Variable/Acumulador

Acumulador = Función (Acumulador)

Cómo se va abreviando la ecuación	**Acumulador encontrado**
(5.8 - (x^2+y^3) * (cos(7.41 / k) + sen(3.901 + j) * 6.51))	[0] = 3.901 + j
(5.8 - (x^2+y^3) * (cos(7.41 / k) + sen([0]) * 6.51))	[1] = sen ([0])
(5.8 - (x^2+y^3) * (cos(7.41 / k) + [1] * 6.51))	[2] = 7.41 / k
(5.8 - (x^2+y^3) * (cos([2]) + [1] * 6.51))	[3] = cos([2])
(5.8 - (x^2+y^3) * ([3] + [1] * 6.51))	[4] = [1] * 6.51
(5.8 - (x^2+y^3) * ([3] + [4]))	[5] = [3] + [4]
(5.8 - (x^2+y^3) * ([5]))	[6] = [5] + 0
(5.8 - (x^2+y^3) * [6])	[7] = x ^ 2
(5.8 - ([7]+y^3) * [6])	[8] = y ^ 3
(5.8 - ([7]+[8]) * [6])	[9] = [7] + [8]
(5.8 - ([9]) * [6])	[10] = [9] + 0
(5.8 - [10] * [6])	[11] = [10] * [6]
(5.8 - [11])	[12] = 5.8 - [11]
([12])	[13] = [12] + 0

Evaluando desde [0] hasta [13] se obtiene el valor numérico de la expresión en forma muy rápida.

Se genera entonces una nueva estructura ArrayList llamado **PiezaEjecuta** que tiene esos acumuladores de [0] a [13].

A tener en consideración:

1. Toda expresión se le adiciona al inicio un paréntesis que abre y al final un paréntesis que cierra.
2. Un paréntesis interno genera un acumulador adicional, por ejemplo, ([12]) genera a [13] = [12] + 0
3. Construye el nuevo ArrayList buscando los paréntesis más internos hasta los más externos.
4. Primero busca el operador # (para resolver los menos unarios), luego el ^ (potencia), luego * y /, y por último + y –

Java

```
//Toma las piezas simples y las convierte en piezas de ejecución de funciones
//Acumula = función (operando(número/variable/acumula))
private void Generar_Piezas_Ejecucion()
{
  int cont = PiezaSimple.size()-1;
  Contador_Acumula = 0;
  do
  {
    if (PiezaSimple.get(cont).getTipo() == ESPARABRE || PiezaSimple.get(cont).getTipo() == ESFUNCION)
    {
      Generar_Piezas_Operador('#', '#', cont);  //Primero evalúa los menos unarios
      Generar_Piezas_Operador('^', '^', cont);  //Luego evalúa las potencias
      Generar_Piezas_Operador('*', '/', cont);  //Luego evalúa multiplicar y dividir
      Generar_Piezas_Operador('+', '-', cont);  //Finalmente evalúa sumar y restar

      //Crea pieza de ejecución
      PiezaEjecuta.add(new Pieza_Ejecuta(PiezaSimple.get(cont).getFuncion(),
          PiezaSimple.get(cont + 1).getTipo(), PiezaSimple.get(cont + 1).getNumero(),
          PiezaSimple.get(cont + 1).getVariable(), PiezaSimple.get(cont + 1).getAcumula(),
          '+', ESNUMERO, 0, 0, 0));

      //La pieza pasa a ser de tipo Acumulador
```

```
        PiezaSimple.get(cont + 1).setAcumula(Contador_Acumula++);

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

  //Toma las piezas simples y las convierte en piezas de ejecución
  //Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
  private void Generar_Piezas_Operador(char operA, char operB, int inicio)
  {
    int cont = inicio + 1;
    do
    {
      if (PiezaSimple.get(cont).getTipo() == ESOPERADOR && (PiezaSimple.get(cont).getOperador() == operA ||
PiezaSimple.get(cont).getOperador() == operB))
      {
        //Crea pieza de ejecución
        PiezaEjecuta.add(new Pieza_Ejecuta(0,
            PiezaSimple.get(cont - 1).getTipo(),
            PiezaSimple.get(cont - 1).getNumero(), PiezaSimple.get(cont - 1).getVariable(), PiezaSimple.get(cont -
1).getAcumula(),
            PiezaSimple.get(cont).getOperador(),
            PiezaSimple.get(cont + 1).getTipo(),
            PiezaSimple.get(cont + 1).getNumero(), PiezaSimple.get(cont + 1).getVariable(), PiezaSimple.get(cont +
1).getAcumula()));

        //Elimina la pieza del operador y la siguiente
        PiezaSimple.remove(cont);
        PiezaSimple.remove(cont);

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

        //Cambia la pieza anterior por pieza acumula
        PiezaSimple.get(cont).setAcumula(Contador_Acumula++);
      }
      cont++;
    } while (cont < PiezaSimple.size() && PiezaSimple.get(cont).getTipo() != ESPARCIERRA);
  }
```

C#

```
  //Toma las piezas simples y las convierte en piezas de ejecución de funciones
  //Acumula = función (operando(número/variable/acumula))
  private void Generar_Piezas_Ejecucion()
  {
    int cont = PiezaSimple.Count()-1;
    Contador_Acumula = 0;
    do
    {
      if (PiezaSimple[cont].getTipo() == ESPARABRE || PiezaSimple[cont].getTipo() == ESFUNCION)
      {
        Generar_Piezas_Operador('#', '#', cont);  //Primero evalúa los menos unarios
        Generar_Piezas_Operador('^', '^', cont);  //Luego evalúa las potencias
        Generar_Piezas_Operador('*', '/', cont);  //Luego evalúa multiplicar y dividir
        Generar_Piezas_Operador('+', '-', cont);  //Finalmente evalúa sumar y restar

        //Crea pieza de ejecución
        PiezaEjecuta.Add(new Pieza_Ejecuta(PiezaSimple[cont].getFuncion(),
           PiezaSimple[cont + 1].getTipo(), PiezaSimple[cont + 1].getNumero(), PiezaSimple[cont + 1].getVariable(),
PiezaSimple[cont + 1].getAcumula(),
           '+', ESNUMERO, 0, 0, 0));

        //La pieza pasa a ser de tipo Acumulador
        PiezaSimple[cont + 1].setAcumula(Contador_Acumula++);

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

  //Toma las piezas simples y las convierte en piezas de ejecución
  //Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
  private void Generar_Piezas_Operador(char operA, char operB, int inicio)
  {
    int cont = inicio + 1;
    do
    {
      if (PiezaSimple[cont].getTipo() == ESOPERADOR && (PiezaSimple[cont].getOperador() == operA ||
PiezaSimple[cont].getOperador() == operB))
      {
        //Crea pieza de ejecución
        PiezaEjecuta.Add(new Pieza_Ejecuta(0,
```

```
            PiezaSimple[cont - 1].getTipo(),
            PiezaSimple[cont - 1].getNumero(), PiezaSimple[cont - 1].getVariable(), PiezaSimple[cont - 1].getAcumula(),
            PiezaSimple[cont].getOperador(),
            PiezaSimple[cont + 1].getTipo(),
            PiezaSimple[cont + 1].getNumero(), PiezaSimple[cont + 1].getVariable(), PiezaSimple[cont + 1].getAcumula()));

        //Elimina la pieza del operador y la siguiente
        PiezaSimple.RemoveAt(cont);
        PiezaSimple.RemoveAt(cont);

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

        //Cambia la pieza anterior por pieza acumula
        PiezaSimple[cont].setAcumula(Contador_Acumula++);
      }
      cont++;
    } while (cont < PiezaSimple.Count() && PiezaSimple[cont].getTipo() != ESPARCIERRA);
  }
```

Visual Basic .NET

```
  'Toma las piezas simples y las convierte en piezas de ejecución de funciones
  'Acumula = función (operando(número/variable/acumula))
  Private Sub Generar_Piezas_Ejecucion()
    Dim cont As Integer = PiezaSimple.Count() - 1
    Contador_Acumula = 0
    Do
      If PiezaSimple(cont).getTipo() = ESPARABRE OrElse PiezaSimple(cont).getTipo() = ESFUNCION Then
        Generar_Piezas_Operador("#", "#", cont)
        'Primero evalúa los menos unarios
        Generar_Piezas_Operador("^", "^", cont)
        'Luego evalúa las potencias
        Generar_Piezas_Operador("*", "/", cont)
        'Luego evalúa multiplicar y dividir
        Generar_Piezas_Operador("+", "-", cont)
        'Finalmente evalúa sumar y restar
        'Crea pieza de ejecución
        PiezaEjecuta.Add(New Pieza_Ejecuta(PiezaSimple(cont).getFuncion(), PiezaSimple(cont + 1).getTipo(), PiezaSimple(cont +
1).getNumero(), PiezaSimple(cont + 1).getVariable(), PiezaSimple(cont + 1).getAcumula(), "+", _
          ESNUMERO, 0, 0, 0))

        'La pieza pasa a ser de tipo Acumulador
        PiezaSimple(cont + 1).setTipo(ESACUMULA)
        PiezaSimple(cont + 1).setAcumula(Contador_Acumula)
        Contador_Acumula = Contador_Acumula + 1

        'Quita el paréntesis/función que abre y el que cierra, dejando el centro
        PiezaSimple.RemoveAt(cont)
        PiezaSimple.RemoveAt(cont + 1)
      End If
      cont -= 1
    Loop While cont >= 0
  End Sub

  'Toma las piezas simples y las convierte en piezas de ejecución
  'Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
  Private Sub Generar_Piezas_Operador(operA As Char, operB As Char, inicio As Integer)
    Dim cont As Integer = inicio + 1
    Do
      If PiezaSimple(cont).getTipo() = ESOPERADOR AndAlso (PiezaSimple(cont).getOperador() = operA OrElse
PiezaSimple(cont).getOperador() = operB) Then
        'Crea pieza de ejecución
        PiezaEjecuta.Add(New Pieza_Ejecuta(0, PiezaSimple(cont - 1).getTipo(), PiezaSimple(cont - 1).getNumero(),
PiezaSimple(cont - 1).getVariable(), PiezaSimple(cont - 1).getAcumula(), PiezaSimple(cont).getOperador(), _
          PiezaSimple(cont + 1).getTipo(), PiezaSimple(cont + 1).getNumero(), PiezaSimple(cont + 1).getVariable(),
PiezaSimple(cont + 1).getAcumula()))

        'Elimina la pieza del operador y la siguiente
        PiezaSimple.RemoveAt(cont)
        PiezaSimple.RemoveAt(cont)

        'Retorna el contador en uno para tomar la siguiente operación
        cont -= 1

        'Cambia la pieza anterior por pieza acumula
        PiezaSimple(cont).setTipo(ESACUMULA)
        PiezaSimple(cont).setAcumula(Contador_Acumula)
        Contador_Acumula = Contador_Acumula + 1
      End If
      cont += 1
    Loop While cont < PiezaSimple.Count() AndAlso PiezaSimple(cont).getTipo() <> ESPARCIERRA
  End Sub
```

C++

```
//Toma las piezas simples y las convierte en piezas de ejecución de funciones
//Acumula = función (operando(número/variable/acumula))
void Evaluar::Generar_Piezas_Ejecucion()
{
```

```
  int cont = PiezaSimple.size()-1;
  Contador_Acumula = 0;
  do
  {
    if (PiezaSimple[cont].getTipo() == ESPARABRE || PiezaSimple[cont].getTipo() == ESFUNCION)
    {
      Generar_Piezas_Operador('#', '#', cont);  //Primero evalúa los menos unarios
      Generar_Piezas_Operador('^', '^', cont);  //Luego las potencias
      Generar_Piezas_Operador('*', '/', cont);  //Luego evalúa multiplicar y dividir
      Generar_Piezas_Operador('+', '-', cont);  //Finalmente evalúa sumar y restar

      //Crea pieza de ejecución
      Pieza_Ejecuta objeto(PiezaSimple[cont].getFuncion(),
                           PiezaSimple[cont + 1].getTipo(), PiezaSimple[cont + 1].getNumero(), PiezaSimple[cont +
1].getVariable(), PiezaSimple[cont + 1].getAcumula(),
                           '+', ESNUMERO, 0, 0, 0);
      PiezaEjecuta.push_back(objeto);

      //La pieza pasa a ser de tipo Acumulador
      PiezaSimple[cont + 1].setAcumula(Contador_Acumula++);

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

//Toma las piezas simples y las convierte en piezas de ejecución
//Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
void Evaluar::Generar_Piezas_Operador(char operA, char operB, int inicio)
{
  int cont = inicio + 1;
  do
  {
    if (PiezaSimple[cont].getTipo() == ESOPERADOR && (PiezaSimple[cont].getOperador() == operA ||
PiezaSimple[cont].getOperador() == operB))
    {
      //Crea pieza de ejecución
      Pieza_Ejecuta objeto(0,
                PiezaSimple[cont - 1].getTipo(),
                PiezaSimple[cont - 1].getNumero(), PiezaSimple[cont - 1].getVariable(), PiezaSimple[cont - 1].getAcumula(),
                PiezaSimple[cont].getOperador(),
                PiezaSimple[cont + 1].getTipo(),
                PiezaSimple[cont + 1].getNumero(), PiezaSimple[cont + 1].getVariable(), PiezaSimple[cont + 1].getAcumula());
      PiezaEjecuta.push_back(objeto);

      //Elimina la pieza del operador y la siguiente
      PiezaSimple.erase(PiezaSimple.begin() + cont);
      PiezaSimple.erase(PiezaSimple.begin() + cont);

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

      //Cambia la pieza anterior por pieza acumula
      PiezaSimple[cont].setAcumula(Contador_Acumula++);
    }
    cont++;
  } while (cont < PiezaSimple.size() && PiezaSimple[cont].getTipo() != ESPARCIERRA);
}
```

Object Pascal

```
//Toma las piezas simples y las convierte en piezas de ejecución de funciones
//Acumula = función (operando(número/variable/acumula))
procedure TEvaluar.Generar_Piezas_Ejecucion();
var
  cont: integer;
begin
  cont := PiezaSimple.Count-1;
  Contador_Acumula := 0;
  repeat
    if ((PiezaSimple[cont] as TPieza_Simple).getTipo() = ESPARABRE) or ((PiezaSimple[cont] as TPieza_Simple).getTipo() =
ESFUNCION) then
    begin
        Generar_Piezas_Operador('#', '#', cont);  //Primero evalúa los menos unarios
        Generar_Piezas_Operador('^', '^', cont);  //Luego evalúa las potencias
        Generar_Piezas_Operador('*', '/', cont);  //Luego evalúa multiplicar y dividir
        Generar_Piezas_Operador('+', '-', cont);  //Finalmente evalúa sumar y restar

        //Crea pieza de ejecución
        objPiezaEjecuta := TPieza_Ejecuta.Create((PiezaSimple[cont] as TPieza_Simple).getFuncion(),
                    (PiezaSimple[cont + 1] as TPieza_Simple).getTipo(), (PiezaSimple[cont + 1] as TPieza_Simple).getNumero(),
(PiezaSimple[cont + 1] as TPieza_Simple).getVariable(), (PiezaSimple[cont + 1] as TPieza_Simple).getAcumula(),
                    '+', ESNUMERO, 0, 0, 0);
        PiezaEjecuta.Add(objPiezaEjecuta);

        //La pieza pasa a ser de tipo Acumulador
        (PiezaSimple[cont + 1] as TPieza_Simple).setTipo(ESACUMULA);
```

```
        (PiezaSimple[cont + 1] as TPieza_Simple).setAcumula(Contador_Acumula);
        Inc(Contador_Acumula);

        //Quita el paréntesis/función que abre y el que cierra, dejando el centro
        PiezaSimple.Delete(cont);
        PiezaSimple.Delete(cont + 1);
    end;
    cont := cont - 1;
  until cont < 0;
end;

//Toma las piezas simples y las convierte en piezas de ejecución
//Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
procedure TEvaluar.Generar_Piezas_Operador(operA: char; operB: char; inicio: integer);
var
  cont: integer;
begin
  cont := inicio + 1;
  repeat
    if ((PiezaSimple[cont] as TPieza_Simple).getTipo() = ESOPERADOR) AND ( ((PiezaSimple[cont] as TPieza_Simple).getOperador() =
operA) OR ((PiezaSimple[cont] as TPieza_Simple).getOperador() = operB) ) then
    begin
      //Crea pieza de ejecución
      objPiezaEjecuta := TPieza_Ejecuta.Create(0,
                         (PiezaSimple[cont - 1] as TPieza_Simple).getTipo(),
                         (PiezaSimple[cont - 1] as TPieza_Simple).getNumero(), (PiezaSimple[cont - 1] as
TPieza_Simple).getVariable(), (PiezaSimple[cont - 1] as TPieza_Simple).getAcumula(),
                         (PiezaSimple[cont] as TPieza_Simple).getOperador(),
                         (PiezaSimple[cont + 1] as TPieza_Simple).getTipo(),
                         (PiezaSimple[cont + 1] as TPieza_Simple).getNumero(), (PiezaSimple[cont + 1] as
TPieza_Simple).getVariable(), (PiezaSimple[cont + 1] as TPieza_Simple).getAcumula());
      PiezaEjecuta.Add(objPiezaEjecuta);

      //Elimina la pieza del operador y la siguiente
      PiezaSimple.Delete(cont);
      PiezaSimple.Delete(cont);

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

      //Cambia la pieza anterior por pieza acumula
      (PiezaSimple[cont] as TPieza_Simple).setTipo(ESACUMULA);
      (PiezaSimple[cont] as TPieza_Simple).setAcumula(Contador_Acumula);
      Inc(Contador_Acumula);
    end;
    Inc(cont);
  until (cont >= PiezaSimple.Count) or ((PiezaSimple[cont] as TPieza_Simple).getTipo() = ESPARCIERRA);
end;
```

JavaScript

```
    //Toma las piezas simples y las convierte en piezas de ejecución de funciones
    //Acumula = función (operando(número/variable/acumula))
    this.Generar_Piezas_Ejecucion = function()
    {
      var cont = this.PiezaSimple.length - 1;
      this.Contador_Acumula = 0;
      do
      {
        if (this.PiezaSimple[cont].getTipo() == this.ESPARABRE || this.PiezaSimple[cont].getTipo() == this.ESFUNCION)
        {
          this.Generar_Piezas_Operador("#", "#", cont);  //Primero evalúa las potencias
          this.Generar_Piezas_Operador("^", "^", cont);  //Primero evalúa las potencias
          this.Generar_Piezas_Operador("*", "/", cont);  //Luego evalúa multiplicar y dividir
          this.Generar_Piezas_Operador("+", "-", cont);  //Finalmente evalúa sumar y restar

          //Crea pieza de ejecución
          objeto = new Pieza_Ejecuta();
          objeto.ConstructorPieza_Ejecuta(this.PiezaSimple[cont].getFuncion(),
              this.PiezaSimple[cont + 1].getTipo(), this.PiezaSimple[cont + 1].getNumero(), this.PiezaSimple[cont +
1].getVariable(), this.PiezaSimple[cont + 1].getAcumula(),
              '+', this.ESNUMERO, 0, 0, 0);
          this.PiezaEjecuta[this.Contador_Acumula] = objeto;

          //La pieza pasa a ser de tipo Acumulador
          this.PiezaSimple[cont + 1].setAcumula(this.Contador_Acumula++);

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

    //Toma las piezas simples y las convierte en piezas de ejecución
    //Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
    this.Generar_Piezas_Operador = function(operA, operB, inicio)
    {
      var cont = inicio + 1;
```

```
      do
      {
        if ((this.PiezaSimple[cont].getTipo() == this.ESOPERADOR) && (this.PiezaSimple[cont].getOperador() == operA ||
this.PiezaSimple[cont].getOperador() == operB))
        {
          //Crea pieza de ejecución
          objeto = new Pieza_Ejecuta();
          objeto.ConstructorPieza_Ejecuta(0,
              this.PiezaSimple[cont - 1].getTipo(),
              this.PiezaSimple[cont - 1].getNumero(), this.PiezaSimple[cont - 1].getVariable(), this.PiezaSimple[cont -
1].getAcumula(),
              this.PiezaSimple[cont].getOperador(),
              this.PiezaSimple[cont + 1].getTipo(),
              this.PiezaSimple[cont + 1].getNumero(), this.PiezaSimple[cont + 1].getVariable(), this.PiezaSimple[cont +
1].getAcumula());
          this.PiezaEjecuta[this.Contador_Acumula] = objeto;

          //Elimina la pieza del operador y la siguiente
          this.PiezaSimple.splice(cont, 1);
          this.PiezaSimple.splice(cont, 1);

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

          //Cambia la pieza anterior por pieza acumula
          this.PiezaSimple[cont].setAcumula(this.Contador_Acumula++);
        }
        cont++;
      } while (cont < this.PiezaSimple.length && this.PiezaSimple[cont].getTipo() != this.ESPARCIERRA);
    }
```

PHP

```
   //Toma las piezas simples y las convierte en piezas de ejecución de funciones
   //Acumula = función (operando(número/variable/acumula))
   function Generar_Piezas_Ejecucion()
   {
      $cont = sizeof($this->PiezaSimple)-1;
      $this->Contador_Acumula = 0;
      do
      {
         if ($this->PiezaSimple[$cont]->getTipo() == $this->ESPARABRE || $this->PiezaSimple[$cont]->getTipo() == $this-
>ESFUNCION)
         {
          $this->Generar_Piezas_Operador('#', '#', $cont);  //Primero evalúa los menos unarios
          $this->Generar_Piezas_Operador('^', '^', $cont);  //Luego evalúa las potencias
          $this->Generar_Piezas_Operador('*', '/', $cont);  //Luego evalúa multiplicar y dividir
          $this->Generar_Piezas_Operador('+', '-', $cont);  //Finalmente evalúa sumar y restar

          //Crea pieza de ejecución
          $objeto = new Pieza_Ejecuta();
          $objeto->ConstructorPiezaEjecuta($this->PiezaSimple[$cont]->getFuncion(),
                $this->PiezaSimple[$cont + 1]->getTipo(), $this->PiezaSimple[$cont + 1]->getNumero(), $this->PiezaSimple[$cont +
1]->getVariable(), $this->PiezaSimple[$cont + 1]->getAcumula(),
                '+', $this->ESNUMERO, 0, 0, 0);
          $this->PiezaEjecuta[] = $objeto;

          //La pieza pasa a ser de tipo Acumulador
          $this->PiezaSimple[$cont + 1]->setAcumula($this->Contador_Acumula++);

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

   //Toma las piezas simples y las convierte en piezas de ejecución
   //Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
   function Generar_Piezas_Operador($operA, $operB, $inicio)
   {
      $cont = $inicio + 1;
      do
      {
         if ($this->PiezaSimple[$cont]->getTipo() == $this->ESOPERADOR && ($this->PiezaSimple[$cont]->getOperador() == $operA ||
$this->PiezaSimple[$cont]->getOperador() == $operB))
         {
          //Crea pieza de ejecución
          $objeto = new Pieza_Ejecuta();
          $objeto->ConstructorPiezaEjecuta(0,
                $this->PiezaSimple[$cont - 1]->getTipo(),
                $this->PiezaSimple[$cont - 1]->getNumero(), $this->PiezaSimple[$cont - 1]->getVariable(), $this-
>PiezaSimple[$cont - 1]->getAcumula(),
                $this->PiezaSimple[$cont]->getOperador(),
                $this->PiezaSimple[$cont + 1]->getTipo(),
                $this->PiezaSimple[$cont + 1]->getNumero(), $this->PiezaSimple[$cont + 1]->getVariable(), $this-
>PiezaSimple[$cont + 1]->getAcumula());
          $this->PiezaEjecuta[] = $objeto;
```

```
          //Elimina la pieza del operador y la siguiente
          unset($this->PiezaSimple[$cont]); $this->PiezaSimple = array_values($this->PiezaSimple);
          unset($this->PiezaSimple[$cont]); $this->PiezaSimple = array_values($this->PiezaSimple);

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

          //Cambia la pieza anterior por pieza acumula
          $this->PiezaSimple[$cont]->setAcumula($this->Contador_Acumula++);
        }
        $cont++;
    } while ($cont < sizeof($this->PiezaSimple) && $this->PiezaSimple[$cont]->getTipo() != $this->ESPARCIERRA);
}
```

¿Qué es cada Nodo en PiezaEjecuta?

Se ha generado un ArrayList, ¿y en qué consiste cada nodo de ese ArrayList? Se responde con la siguiente clase

Java

```
public class Pieza_Ejecuta
{
  private double valorPieza;

  private int funcion;

  private int tipo_operandoA;
  private double numeroA;
  private int variableA;
  private int acumulaA;

  private char operador;

  private int tipo_operandoB;
  private double numeroB;
  private int variableB;
  private int acumulaB;

  public final double getValorPieza() { return this.valorPieza; }
  public final void setValorPieza(double valor) { this.valorPieza = valor; }
  public final int getFuncion() { return this.funcion; }
  public final int getTipoOperA() { return this.tipo_operandoA; }
  public final double getNumeroA() { return this.numeroA; }
  public final int getVariableA() { return this.variableA; }
  public final int getAcumulaA() { return this.acumulaA; }
  public final char getOperador() { return this.operador; }
  public final int getTipoOperB() { return this.tipo_operandoB; }
  public final double getNumeroB() { return this.numeroB; }
  public final int getVariableB() { return this.variableB; }
  public final int getAcumulaB() { return this.acumulaB; }

  public Pieza_Ejecuta(int funcion, int tipo_operandoA, double numeroA, int variableA, int acumulaA, char operador, int
tipo_operandoB, double numeroB, int variableB, int acumulaB)
  {
    this.valorPieza = 0;

    this.funcion = funcion;

    this.tipo_operandoA = tipo_operandoA;
    this.numeroA = numeroA;
    this.variableA = variableA;
    this.acumulaA = acumulaA;

    this.operador = operador;

    this.tipo_operandoB = tipo_operandoB;
    this.numeroB = numeroB;
    this.variableB = variableB;
    this.acumulaB = acumulaB;
  }
}
```

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EvaluadorCS
{
  class Pieza_Ejecuta
  {
    private double valorPieza;  //Almacena el calculo de la operación. Es Acumula
```

```
    private int funcion;  // ¿Es una función? 0 no lo es. 1 es seno, 3 es coseno, ....

    private int tipo_operandoA; //Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
    private double numeroA;
    private int variableA;
    private int acumulaA;

    private char operador; // +, -, *, /, ^

    private int tipo_operandoB; //Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
    private double numeroB;
    private int variableB;
    private int acumulaB;

    public double getValorPieza() { return this.valorPieza; }
    public void setValorPieza(double valor) { this.valorPieza = valor; }
    public int getFuncion() { return this.funcion; }
    public int getTipoOperA() { return this.tipo_operandoA; }
    public double getNumeroA() { return this.numeroA; }
    public int getVariableA() { return this.variableA; }
    public int getAcumulaA() { return this.acumulaA; }
    public char getOperador() { return this.operador; }
    public int getTipoOperB() { return this.tipo_operandoB; }
    public double getNumeroB() { return this.numeroB; }
    public int getVariableB() { return this.variableB; }
    public int getAcumulaB() { return this.acumulaB; }

    public Pieza_Ejecuta(int funcion, int tipo_operandoA, double numeroA, int variableA, int acumulaA, char operador, int
tipo_operandoB, double numeroB, int variableB, int acumulaB)
    {
      this.valorPieza = 0;

      this.funcion = funcion;

      this.tipo_operandoA = tipo_operandoA;
      this.numeroA = numeroA;
      this.variableA = variableA;
      this.acumulaA = acumulaA;

      this.operador = operador;

      this.tipo_operandoB = tipo_operandoB;
      this.numeroB = numeroB;
      this.variableB = variableB;
      this.acumulaB = acumulaB;
    }
  }
}
```

Visual Basic .NET

```
Public Class Pieza_Ejecuta
    'Almacena el calculo de la operación. Es Acumula
    Private valorPieza As Double

    ' ¿Es una función? 0 no lo es. 1 es seno, 3 es coseno, ....
    Private funcion As Integer

    'Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
    Private tipo_operandoA As Integer
    Private numeroA As Double
    Private variableA As Integer
    Private acumulaA As Integer

    ' +, -, *, /, ^
    Private operador As Char

    'Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
    Private tipo_operandoB As Integer
    Private numeroB As Double
    Private variableB As Integer
    Private acumulaB As Integer

    Public Function getValorPieza() As Double
        Return Me.valorPieza
    End Function
    Public Sub setValorPieza(valor As Double)
        Me.valorPieza = valor
    End Sub
    Public Function getFuncion() As Integer
        Return Me.funcion
    End Function
    Public Function getTipoOperA() As Integer
        Return Me.tipo_operandoA
    End Function
    Public Function getNumeroA() As Double
        Return Me.numeroA
    End Function
```

```
    Public Function getVariableA() As Integer
        Return Me.variableA
    End Function
    Public Function getAcumulaA() As Integer
        Return Me.acumulaA
    End Function
    Public Function getOperador() As Char
        Return Me.operador
    End Function
    Public Function getTipoOperB() As Integer
        Return Me.tipo_operandoB
    End Function
    Public Function getNumeroB() As Double
        Return Me.numeroB
    End Function
    Public Function getVariableB() As Integer
        Return Me.variableB
    End Function
    Public Function getAcumulaB() As Integer
        Return Me.acumulaB
    End Function

    Public Sub New(funcion As Integer, tipo_operandoA As Integer, numeroA As Double, variableA As Integer, acumulaA As Integer,
operador As Char, _
        tipo_operandoB As Integer, numeroB As Double, variableB As Integer, acumulaB As Integer)
        Me.valorPieza = 0

        Me.funcion = funcion

        Me.tipo_operandoA = tipo_operandoA
        Me.numeroA = numeroA
        Me.variableA = variableA
        Me.acumulaA = acumulaA

        Me.operador = operador

        Me.tipo_operandoB = tipo_operandoB
        Me.numeroB = numeroB
        Me.variableB = variableB
        Me.acumulaB = acumulaB
    End Sub
End Class
```

C++

```
#include "Pieza_Ejecuta.h"

double Pieza_Ejecuta::getValorPieza() { return this->valorPieza; }
void Pieza_Ejecuta::setValorPieza(double valor) { this->valorPieza = valor; }
int Pieza_Ejecuta::getFuncion() { return this->funcion; }
int Pieza_Ejecuta::getTipoOperA() { return this->tipo_operandoA; }
double Pieza_Ejecuta::getNumeroA() { return this->numeroA; }
int Pieza_Ejecuta::getVariableA() { return this->variableA; }
int Pieza_Ejecuta::getAcumulaA() { return this->acumulaA; }
char Pieza_Ejecuta::getOperador() { return this->operador; }
int Pieza_Ejecuta::getTipoOperB() { return this->tipo_operandoB; }
double Pieza_Ejecuta::getNumeroB() { return this->numeroB; }
int Pieza_Ejecuta::getVariableB() { return this->variableB; }
int Pieza_Ejecuta::getAcumulaB() { return this->acumulaB; }

Pieza_Ejecuta::Pieza_Ejecuta(int funcion, int tipo_operandoA, double numeroA, int variableA, int acumulaA, char operador, int
tipo_operandoB, double numeroB, int variableB, int acumulaB)
{
  this->valorPieza = 0;

  this->funcion = funcion;

  this->tipo_operandoA = tipo_operandoA;
  this->numeroA = numeroA;
  this->variableA = variableA;
  this->acumulaA = acumulaA;

  this->operador = operador;

  this->tipo_operandoB = tipo_operandoB;
  this->numeroB = numeroB;
  this->variableB = variableB;
  this->acumulaB = acumulaB;
}
```

Object Pascal

```
unit Pieza_Ejecuta;

interface
type
  TPieza_Ejecuta = class
  private
        valorPieza: double;  //Almacena el calculo de la operación. Es Acumula
```

```
        funcion: integer;  // ¿Es una función? 0 no lo es. 1 es seno, 3 es coseno, ....

        tipo_operandoA: integer; //Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
        numeroA: double;
        variableA: integer;
        acumulaA: integer;

        operador: char; // +, -, *, /, ^

        tipo_operandoB: integer; //Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
        numeroB: double;
        variableB: integer;
        acumulaB: integer;
   public
        Constructor Create(funcion: integer; tipo_operandoA: integer; numeroA: double; variableA: integer; acumulaA: integer;
operador: char; tipo_operandoB: integer; numeroB: double; variableB: integer; acumulaB: integer);
        function getValorPieza(): double;
        procedure setValorPieza(valor: double);
        function getFuncion(): integer;
        function getTipoOperA(): integer;
        function getNumeroA(): double;
        function getVariableA(): integer;
        function getAcumulaA(): integer;
        function getOperador(): char;
        function getTipoOperB(): integer;
        function getNumeroB(): double;
        function getVariableB(): integer;
        function getAcumulaB(): integer;
   end;
implementation

Constructor TPieza_Ejecuta.Create(funcion: integer; tipo_operandoA: integer; numeroA: double; variableA: integer; acumulaA:
integer; operador: char; tipo_operandoB: integer; numeroB: double; variableB: integer; acumulaB: integer);
begin
    self.valorPieza := 0;

    self.funcion := funcion;

    self.tipo_operandoA := tipo_operandoA;
    self.numeroA := numeroA;
    self.variableA := variableA;
    self.acumulaA := acumulaA;

    self.operador := operador;

    self.tipo_operandoB := tipo_operandoB;
    self.numeroB := numeroB;
    self.variableB := variableB;
    self.acumulaB := acumulaB;
end;

function TPieza_Ejecuta.getValorPieza(): double;
begin
     getValorPieza := valorPieza;
end;

procedure TPieza_Ejecuta.setValorPieza(valor: double);
begin
     self.valorPieza := valor;
end;

function TPieza_Ejecuta.getFuncion(): integer;
begin
    getFuncion := funcion;
end;

function TPieza_Ejecuta.getTipoOperA(): integer;
begin
    getTipoOperA := tipo_operandoA;
end;

function TPieza_Ejecuta.getNumeroA(): double;
begin
    getNumeroA := numeroA;
end;

function TPieza_Ejecuta.getVariableA(): integer;
begin
  getVariableA := variableA;
end;

function TPieza_Ejecuta.getAcumulaA(): integer;
begin
  getAcumulaA := acumulaA;
end;

function TPieza_Ejecuta.getOperador(): char;
begin
  getOperador := operador;
end;

function TPieza_Ejecuta.getTipoOperB(): integer;
```

```
begin
    getTipoOperB := tipo_operandoB;
end;

function TPieza_Ejecuta.getNumeroB(): double;
begin
    getNumeroB := numeroB;
end;

function TPieza_Ejecuta.getVariableB(): integer;
begin
    getVariableB := variableB;
end;

function TPieza_Ejecuta.getAcumulaB(): integer;
begin
    getAcumulaB := acumulaB;
end;
end.
```

JavaScript

```
function Pieza_Ejecuta()
{
    this.valorPieza;

    this.funcion;

    this.tipo_operandoA;
    this.numeroA;
    this.variableA;
    this.acumulaA;

    this.operador;

    this.tipo_operandoB;
    this.numeroB;
    this.variableB;
    this.acumulaB;

    this.getValorPieza = function()
    {
      return this.valorPieza;
    }

    this.setValorPieza = function(valor)
    {
      this.valorPieza = valor;
    }

    this.getFuncion = function()
    {
      return this.funcion;
    }

    this.getTipoOperA = function()
    {
      return this.tipo_operandoA;
    }

    this.getNumeroA = function()
    {
      return this.numeroA;
    }

    this.getVariableA = function()
    {
      return this.variableA;
    }

    this.getAcumulaA = function()
    {
      return this.acumulaA;
    }

    this.getOperador = function()
    {
      return this.operador;
    }

    this.getTipoOperB = function()
    {
      return this.tipo_operandoB;
    }

    this.getNumeroB = function()
    {
      return this.numeroB;
    }
```

```
    this.getVariableB = function()
    {
      return this.variableB;
    }

    this.getAcumulaB = function()
    {
      return this.acumulaB;
    }

    this.ConstructorPieza_Ejecuta = function(funcion, tipo_operandoA, numeroA, variableA, acumulaA, operador, tipo_operandoB,
numeroB, variableB, acumulaB)
    {
      this.valorPieza = 0;

      this.funcion = funcion;

      this.tipo_operandoA = tipo_operandoA;
      this.numeroA = numeroA;
      this.variableA = variableA;
      this.acumulaA = acumulaA;

      this.operador = operador;

      this.tipo_operandoB = tipo_operandoB;
      this.numeroB = numeroB;
      this.variableB = variableB;
      this.acumulaB = acumulaB;
    }
}
```

PHP

```
<?php
class Pieza_Ejecuta
{
      var $valorPieza;

      var $funcion;

      var $tipo_operandoA;
      var $numeroA;
      var $variableA;
      var $acumulaA;

      var $operador;

      var $tipo_operandoB;
      var $numeroB;
      var $variableB;
      var $acumulaB;

      public function getValorPieza() { return $this->valorPieza;      }
      public function setValorPieza($valor) { $this->valorPieza = $valor;    }
      public function getFuncion() { return $this->funcion; }
      public function getTipoOperA() { return $this->tipo_operandoA; }
      public function getNumeroA() { return $this->numeroA; }
      public function getVariableA() { return $this->variableA; }
      public function getAcumulaA() {     return $this->acumulaA; }
      public function getOperador() {     return $this->operador; }
      public function getTipoOperB() { return $this->tipo_operandoB; }
      public function getNumeroB() { return $this->numeroB; }
      public function getVariableB() { return $this->variableB;}
      public function getAcumulaB() {     return $this->acumulaB; }

      public function ConstructorPiezaEjecuta($funcion, $tipo_operandoA, $numeroA, $variableA, $acumulaA, $operador,
$tipo_operandoB, $numeroB, $variableB, $acumulaB)
      {
            $this->valorPieza = 0;

            $this->funcion = $funcion;

            $this->tipo_operandoA = $tipo_operandoA;
            $this->numeroA = $numeroA;
            $this->variableA = $variableA;
            $this->acumulaA = $acumulaA;

            $this->operador = $operador;

            $this->tipo_operandoB = $tipo_operandoB;
            $this->numeroB = $numeroB;
            $this->variableB = $variableB;
            $this->acumulaB = $acumulaB;
      }
}
?>
```

El método Leer_Variables()

Java

```
// Da valor a las variables que tendrá la expresión algebraica
public final void ValorVariable(char variableAlgebra, double valor)
{
  VariableAlgebra[variableAlgebra - ASCIILETRA] = valor;
}
```

C#

```
// Da valor a las variables que tendrá la expresión algebraica
public void ValorVariable(char variableAlg, double valor)
{
   VariableAlgebra[variableAlg - ASCIILETRA] = valor;
}
```

Visual Basic .NET

```
' Da valor a las variables que tendrá la expresión algebraica
Public Sub ValorVariable(variableAlg As Char, valor As Double)
  VariableAlgebra(Asc(variableAlg) - ASCIILETRA) = valor
End Sub
```

C++

```
// Da valor a las variables que tendrá la expresión algebraica
void Evaluar::ValorVariable(char variableAlg, double valor)
{
  VariableAlgebra[variableAlg - ASCIILETRA] = valor;
}
```

Object Pascal

```
// Da valor a las variables que tendrá la expresión algebraica
procedure TEvaluar.ValorVariable(variableAlgebra: char; valor: double);
begin
  self.VariableAlgebra[ord(variableAlgebra) - self.ASCIILETRA] := valor;
end;
```

JavaScript

```
    // Da valor a las variables que tendrá la expresión algebraica
    this.ValorVariable = function(varAlgebra, valor)
    {
      this.VariableAlgebra[varAlgebra.charCodeAt(0) - this.ASCIILETRA] = valor;
    }
```

PHP

```
   // Da valor a las variables que tendrá la expresión algebraica
   function ValorVariable($variableAlg, $valor)
   {
      $this->VariableAlgebra[ord($variableAlg) - $this->ASCIILETRA] = $valor;
   }
```

El método Evalúa_Expresión()

La expresión ya está analizada y ahora es un arraylist de Pieza_Ejecuta. Las variables han sido leídas. Luego viene la ejecución para dar con el valor. Este es el método crítico en velocidad para múltiples evaluaciones.

Java

```
  //Calcula la expresión convertida en piezas de ejecución
  public final double Calcular()
  {
    double valorA=0, valorB=0;

    for (Pieza_Ejecuta aPiezaEjecuta : PiezaEjecuta)
    {
      switch (aPiezaEjecuta.getTipoOperA())
      {
        case 5: valorA = aPiezaEjecuta.getNumeroA(); break; //¿Es un número?
        case 6: valorA = VariableAlgebra[aPiezaEjecuta.getVariableA()]; break;  //¿Es una variable?
        case 7: valorA = PiezaEjecuta.get(aPiezaEjecuta.getAcumulaA()).getValorPieza();  break; //¿Es una expresión anterior?
      }
      if (Double.isNaN(valorA) || Double.isInfinite(valorA)) return valorA;

      switch (aPiezaEjecuta.getFuncion())
      {
        case 0:
          switch (aPiezaEjecuta.getTipoOperB()) {
            case 5:  valorB = aPiezaEjecuta.getNumeroB();  break; //¿Es un número?
            case 6:  valorB = VariableAlgebra[aPiezaEjecuta.getVariableB()];  break;  //¿Es una variable?
            case 7:  valorB = PiezaEjecuta.get(aPiezaEjecuta.getAcumulaB()).getValorPieza();  break; //¿Es una expresión
anterior?
          }
          if (Double.isNaN(valorB) || Double.isInfinite(valorB)) return valorB;

          switch (aPiezaEjecuta.getOperador())
          {
            case '#': aPiezaEjecuta.setValorPieza(valorA * valorB);  break;
            case '+': aPiezaEjecuta.setValorPieza(valorA + valorB);  break;
            case '-': aPiezaEjecuta.setValorPieza(valorA - valorB);  break;
            case '*': aPiezaEjecuta.setValorPieza(valorA * valorB);  break;
            case '/': aPiezaEjecuta.setValorPieza(valorA / valorB);  break;
            case '^': aPiezaEjecuta.setValorPieza(Math.pow(valorA, valorB));  break;
          }
          break;
        case 1:
        case 2: aPiezaEjecuta.setValorPieza(Math.sin(valorA));  break;
        case 3: aPiezaEjecuta.setValorPieza(Math.cos(valorA));  break;
        case 4: aPiezaEjecuta.setValorPieza(Math.tan(valorA));  break;
        case 5: aPiezaEjecuta.setValorPieza(Math.abs(valorA));  break;
        case 6: aPiezaEjecuta.setValorPieza(Math.asin(valorA)); break;
        case 7: aPiezaEjecuta.setValorPieza(Math.acos(valorA)); break;
        case 8: aPiezaEjecuta.setValorPieza(Math.atan(valorA)); break;
        case 9: aPiezaEjecuta.setValorPieza(Math.log(valorA));  break;
        case 10:aPiezaEjecuta.setValorPieza(Math.ceil(valorA)); break;
        case 11:aPiezaEjecuta.setValorPieza(Math.exp(valorA));  break;
        case 12:aPiezaEjecuta.setValorPieza(Math.sqrt(valorA)); break;
        case 13:aPiezaEjecuta.setValorPieza(Math.pow(valorA, 1 / 3));  break;
      }
    }
    return PiezaEjecuta.get(PiezaEjecuta.size() - 1).getValorPieza();
  }
```

C#

```
  //Calcula la expresión convertida en piezas de ejecución
  public double Calcular()
  {
    double valorA=0, valorB=0;
    int totalPiezaEjecuta = PiezaEjecuta.Count();

    for (int cont = 0; cont < totalPiezaEjecuta; cont++)
    {
      switch (PiezaEjecuta[cont].getTipoOperA())
      {
        case 5: valorA = PiezaEjecuta[cont].getNumeroA(); break; //¿Es un número?
        case 6: valorA = VariableAlgebra[PiezaEjecuta[cont].getVariableA()]; break;  //¿Es una variable?
        case 7: valorA = PiezaEjecuta[PiezaEjecuta[cont].getAcumulaA()].getValorPieza(); break; //¿Es una expresión anterior?
      }
      if (Double.IsNaN(valorA) || Double.IsInfinity(valorA)) return valorA;

      switch (PiezaEjecuta[cont].getFuncion())
      {
        case 0:
          switch (PiezaEjecuta[cont].getTipoOperB())
          {
            case 5: valorB = PiezaEjecuta[cont].getNumeroB(); break; //¿Es un número?
            case 6: valorB = VariableAlgebra[PiezaEjecuta[cont].getVariableB()]; break;  //¿Es una variable?
            case 7: valorB = PiezaEjecuta[PiezaEjecuta[cont].getAcumulaB()].getValorPieza(); break; //¿Es una expresión
anterior?
```

```
          }
          if (Double.IsNaN(valorB) || Double.IsInfinity(valorB)) return valorB;

          switch (PiezaEjecuta[cont].getOperador())
          {
            case '#': PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
            case '+': PiezaEjecuta[cont].setValorPieza(valorA + valorB); break;
            case '-': PiezaEjecuta[cont].setValorPieza(valorA - valorB); break;
            case '*': PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
            case '/': PiezaEjecuta[cont].setValorPieza(valorA / valorB); break;
            case '^': PiezaEjecuta[cont].setValorPieza(Math.Pow(valorA, valorB)); break;
          }
        break;
        case 1:
        case 2: PiezaEjecuta[cont].setValorPieza(Math.Sin(valorA)); break;
        case 3: PiezaEjecuta[cont].setValorPieza(Math.Cos(valorA)); break;
        case 4: PiezaEjecuta[cont].setValorPieza(Math.Tan(valorA)); break;
        case 5: PiezaEjecuta[cont].setValorPieza(Math.Abs(valorA)); break;
        case 6: PiezaEjecuta[cont].setValorPieza(Math.Asin(valorA)); break;
        case 7: PiezaEjecuta[cont].setValorPieza(Math.Acos(valorA)); break;
        case 8: PiezaEjecuta[cont].setValorPieza(Math.Atan(valorA)); break;
        case 9: PiezaEjecuta[cont].setValorPieza(Math.Log(valorA)); break;
        case 10: PiezaEjecuta[cont].setValorPieza(Math.Ceiling(valorA)); break;
        case 11: PiezaEjecuta[cont].setValorPieza(Math.Exp(valorA)); break;
        case 12: PiezaEjecuta[cont].setValorPieza(Math.Sqrt(valorA)); break;
        case 13: PiezaEjecuta[cont].setValorPieza(Math.Pow(valorA, 0.333333333333)); break;
      }
    }
    return PiezaEjecuta[totalPiezaEjecuta - 1].getValorPieza();
  }
```

Visual Basic .NET

```
  'Calcula la expresión convertida en piezas de ejecución
   Public Function Calcular() As Double
       Dim valorA As Double = 0, valorB As Double = 0
       Dim totalPiezaEjecuta As Integer = PiezaEjecuta.Count()

       For cont As Integer = 0 To totalPiezaEjecuta - 1
           Select Case PiezaEjecuta(cont).getTipoOperA()
               Case 5
                   valorA = PiezaEjecuta(cont).getNumeroA()
                   Exit Select
                   '¿Es un número?
               Case 6
                   valorA = VariableAlgebra(PiezaEjecuta(cont).getVariableA())
                   Exit Select
                   '¿Es una variable?
               Case 7
                   valorA = PiezaEjecuta(PiezaEjecuta(cont).getAcumulaA()).getValorPieza()
                   Exit Select
                   '¿Es una expresión anterior?
           End Select
           If [Double].IsNaN(valorA) OrElse [Double].IsInfinity(valorA) Then
               Return valorA
           End If

           Select Case PiezaEjecuta(cont).getFuncion()
               Case 0
                   Select Case PiezaEjecuta(cont).getTipoOperB()
                       Case 5
                           valorB = PiezaEjecuta(cont).getNumeroB()
                           Exit Select
                           '¿Es un número?
                       Case 6
                           valorB = VariableAlgebra(PiezaEjecuta(cont).getVariableB())
                           Exit Select
                           '¿Es una variable?
                       Case 7
                           valorB = PiezaEjecuta(PiezaEjecuta(cont).getAcumulaB()).getValorPieza()
                           Exit Select
                           '¿Es una expresión anterior?
                   End Select
                   If [Double].IsNaN(valorB) OrElse [Double].IsInfinity(valorB) Then
                       Return valorB
                   End If

                   Select Case PiezaEjecuta(cont).getOperador()
                       Case "#"
                           PiezaEjecuta(cont).setValorPieza(valorA * valorB)
                           Exit Select
                       Case "+"
                           PiezaEjecuta(cont).setValorPieza(valorA + valorB)
                           Exit Select
                       Case "-"
                           PiezaEjecuta(cont).setValorPieza(valorA - valorB)
                           Exit Select
                       Case "*"
                           PiezaEjecuta(cont).setValorPieza(valorA * valorB)
                           Exit Select
```

```
                Case "/"
                    PiezaEjecuta(cont).setValorPieza(valorA / valorB)
                    Exit Select
                Case "^"
                    PiezaEjecuta(cont).setValorPieza(Math.Pow(valorA, valorB))
                    Exit Select
            End Select
            Exit Select
        Case 1, 2
            PiezaEjecuta(cont).setValorPieza(Math.Sin(valorA))
            Exit Select
        Case 3
            PiezaEjecuta(cont).setValorPieza(Math.Cos(valorA))
            Exit Select
        Case 4
            PiezaEjecuta(cont).setValorPieza(Math.Tan(valorA))
            Exit Select
        Case 5
            PiezaEjecuta(cont).setValorPieza(Math.Abs(valorA))
            Exit Select
        Case 6
            PiezaEjecuta(cont).setValorPieza(Math.Asin(valorA))
            Exit Select
        Case 7
            PiezaEjecuta(cont).setValorPieza(Math.Acos(valorA))
            Exit Select
        Case 8
            PiezaEjecuta(cont).setValorPieza(Math.Atan(valorA))
            Exit Select
        Case 9
            PiezaEjecuta(cont).setValorPieza(Math.Log(valorA))
            Exit Select
        Case 10
            PiezaEjecuta(cont).setValorPieza(Math.Ceiling(valorA))
            Exit Select
        Case 11
            PiezaEjecuta(cont).setValorPieza(Math.Exp(valorA))
            Exit Select
        Case 12
            PiezaEjecuta(cont).setValorPieza(Math.Sqrt(valorA))
            Exit Select
        Case 13
            PiezaEjecuta(cont).setValorPieza(Math.Pow(valorA, 0.333333333333))
            Exit Select
    End Select
  Next
  Return PiezaEjecuta(totalPiezaEjecuta - 1).getValorPieza()
End Function
```

C++

```
//Calcula la expresión convertida en piezas de ejecución
double Evaluar::Calcular()
{
  double valorA=0, valorB=0;
  int totalPiezaEjecuta = PiezaEjecuta.size();

  for (int cont = 0; cont < totalPiezaEjecuta; cont++)
  {
    switch (PiezaEjecuta[cont].getTipoOperA())
    {
      case 5: valorA = PiezaEjecuta[cont].getNumeroA(); break; //¿Es un número?
      case 6: valorA = VariableAlgebra[PiezaEjecuta[cont].getVariableA()]; break;  //¿Es una variable?
      case 7: valorA = PiezaEjecuta[PiezaEjecuta[cont].getAcumulaA()].getValorPieza(); break; //¿Es una expresión anterior?
    }
    if (_isnan(valorA) || !_finite(valorA)) return valorA;

    switch (PiezaEjecuta[cont].getFuncion())
    {
      case 0:
        switch (PiezaEjecuta[cont].getTipoOperB())
        {
          case 5: valorB = PiezaEjecuta[cont].getNumeroB(); break; //¿Es un número?
          case 6: valorB = VariableAlgebra[PiezaEjecuta[cont].getVariableB()]; break;  //¿Es una variable?
          case 7: valorB = PiezaEjecuta[PiezaEjecuta[cont].getAcumulaB()].getValorPieza(); break; //¿Es una expresión anterior?
        }
        if (_isnan(valorB) || !_finite(valorB)) return valorB;

        switch (PiezaEjecuta[cont].getOperador())
        {
          case '#': PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
          case '+': PiezaEjecuta[cont].setValorPieza(valorA + valorB); break;
          case '-': PiezaEjecuta[cont].setValorPieza(valorA - valorB); break;
          case '*': PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
          case '/': PiezaEjecuta[cont].setValorPieza(valorA / valorB); break;
          case '^': PiezaEjecuta[cont].setValorPieza(pow(valorA, valorB)); break;
        }
        break;
      case 1:
      case 2: PiezaEjecuta[cont].setValorPieza(sin(valorA)); break;
```

```
      case 3: PiezaEjecuta[cont].setValorPieza(cos(valorA)); break;
      case 4: PiezaEjecuta[cont].setValorPieza(tan(valorA)); break;
      case 5: PiezaEjecuta[cont].setValorPieza(abs(valorA)); break;
      case 6: PiezaEjecuta[cont].setValorPieza(asin(valorA)); break;
      case 7: PiezaEjecuta[cont].setValorPieza(acos(valorA)); break;
      case 8: PiezaEjecuta[cont].setValorPieza(atan(valorA)); break;
      case 9: PiezaEjecuta[cont].setValorPieza(log(valorA)); break;
      case 10: PiezaEjecuta[cont].setValorPieza(ceil(valorA)); break;
      case 11: PiezaEjecuta[cont].setValorPieza(exp(valorA)); break;
      case 12: PiezaEjecuta[cont].setValorPieza(sqrt(valorA)); break;
      case 13: PiezaEjecuta[cont].setValorPieza(pow(valorA, 0.333333333333)); break;
    }
  }
  return PiezaEjecuta[totalPiezaEjecuta - 1].getValorPieza();
}
```

Object Pascal

```
//Calcula la expresión convertida en piezas de ejecución
function TEvaluar.Calcular(): double;
var
  valorA, valorB: double;
  totalPiezaEjecuta, cont: integer;
begin
  valorA := 0;
  valorB := 0;
  totalPiezaEjecuta := PiezaEjecuta.Count;

  try
    for cont := 0 to totalPiezaEjecuta-1 do
    begin
      case (PiezaEjecuta[cont] as TPieza_Ejecuta).getTipoOperA() of
        5: begin valorA := (PiezaEjecuta[cont] as TPieza_Ejecuta).getNumeroA(); end; //¿Es un número?
        6: begin valorA := VariableAlgebra[(PiezaEjecuta[cont] as TPieza_Ejecuta).getVariableA()]; end;  //¿Es una variable?
        7: begin valorA := (PiezaEjecuta[(PiezaEjecuta[cont] as TPieza_Ejecuta).getAcumulaA()] as
TPieza_Ejecuta).getValorPieza(); end; //¿Es una expresión anterior?
      end;

      case (PiezaEjecuta[cont] as TPieza_Ejecuta).getFuncion() of
        0:begin
            if (PiezaEjecuta[cont] as TPieza_Ejecuta).getTipoOperB() = 5 then valorB := (PiezaEjecuta[cont] as
TPieza_Ejecuta).getNumeroB() //¿Es un número?
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getTipoOperB() = 6 then valorB := VariableAlgebra[(PiezaEjecuta[cont]
as TPieza_Ejecuta).getVariableB()] //¿Es una variable?
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getTipoOperB() = 7 then valorB := (PiezaEjecuta[(PiezaEjecuta[cont]
as TPieza_Ejecuta).getAcumulaB()] as TPieza_Ejecuta).getValorPieza(); //¿Es una expresión anterior?

            if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '#' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(valorA * valorB)
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '+' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(valorA + valorB)
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '-' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(valorA - valorB)
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '*' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(valorA * valorB)
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '/' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(valorA / valorB)
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '^' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(power(valorA, valorB));
          end;
        1, 2: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(sin(valorA)); end;
        3: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(cos(valorA)); end;
        4: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(tan(valorA)); end;
        5: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(abs(valorA)); end;
        6: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(arcsin(valorA)); end;
        7: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(arccos(valorA)); end;
        8: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(arctan(valorA)); end;
        9: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(ln(valorA)); end;
        10: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(ceil(valorA)); end;
        11: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(exp(valorA)); end;
        12: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(sqrt(valorA)); end;
        13: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(power(valorA, 0.333333333333)); end;
      end;
    end;
  except //Captura el error matemático
      on EMathError do
      begin
        Result := NaN;
        Exit;
      end;
  end;
  Calcular := (PiezaEjecuta[totalPiezaEjecuta - 1] as TPieza_Ejecuta).getValorPieza();
end;
```

JavaScript

```
    //Calcula la expresión convertida en piezas de ejecución
```

```
    this.Calcular = function()
    {
      var valorA=0, valorB=0;
      var totalPiezaEjecuta = this.PiezaEjecuta.length;

      for (var cont = 0; cont < totalPiezaEjecuta; cont++)
      {
        switch (this.PiezaEjecuta[cont].getTipoOperA())
        {
          case 5: valorA = this.PiezaEjecuta[cont].getNumeroA(); break; //¿Es un número?
          case 6: valorA = this.VariableAlgebra[this.PiezaEjecuta[cont].getVariableA()]; break;  //¿Es una variable?
          case 7: valorA = this.PiezaEjecuta[this.PiezaEjecuta[cont].getAcumulaA()].getValorPieza(); break; //¿Es una expresión
anterior?
        }
        if (isNaN(valorA) || !isFinite(valorA)) return valorA;

        switch (this.PiezaEjecuta[cont].getFuncion())
        {
          case 0:
            switch (this.PiezaEjecuta[cont].getTipoOperB())
            {
              case 5: valorB = this.PiezaEjecuta[cont].getNumeroB(); break; //¿Es un número?
              case 6: valorB = this.VariableAlgebra[this.PiezaEjecuta[cont].getVariableB()]; break;  //¿Es una variable?
              case 7: valorB = this.PiezaEjecuta[this.PiezaEjecuta[cont].getAcumulaB()].getValorPieza(); break; //¿Es una
expresión anterior?
            }
            if (isNaN(valorB) || !isFinite(valorB)) return valorB;

            switch (this.PiezaEjecuta[cont].getOperador())
            {
              case '#': this.PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
              case '+': this.PiezaEjecuta[cont].setValorPieza(valorA + valorB); break;
              case '-': this.PiezaEjecuta[cont].setValorPieza(valorA - valorB); break;
              case '*': this.PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
              case '/': this.PiezaEjecuta[cont].setValorPieza(valorA / valorB); break;
              case '^': this.PiezaEjecuta[cont].setValorPieza(Math.pow(valorA, valorB)); break;
            }
            break;
          case 1:
          case 2: this.PiezaEjecuta[cont].setValorPieza(Math.sin(valorA)); break;
          case 3: this.PiezaEjecuta[cont].setValorPieza(Math.cos(valorA)); break;
          case 4: this.PiezaEjecuta[cont].setValorPieza(Math.tan(valorA)); break;
          case 5: this.PiezaEjecuta[cont].setValorPieza(Math.abs(valorA)); break;
          case 6: this.PiezaEjecuta[cont].setValorPieza(Math.asin(valorA)); break;
          case 7: this.PiezaEjecuta[cont].setValorPieza(Math.acos(valorA)); break;
          case 8: this.PiezaEjecuta[cont].setValorPieza(Math.atan(valorA)); break;
          case 9: this.PiezaEjecuta[cont].setValorPieza(Math.log(valorA)); break;
          case 10: this.PiezaEjecuta[cont].setValorPieza(Math.ceil(valorA)); break;
          case 11: this.PiezaEjecuta[cont].setValorPieza(Math.exp(valorA)); break;
          case 12: this.PiezaEjecuta[cont].setValorPieza(Math.sqrt(valorA)); break;
          case 13: this.PiezaEjecuta[cont].setValorPieza(Math.pow(valorA, 0.333333333333)); break;
        }
      }
      return this.PiezaEjecuta[totalPiezaEjecuta - 1].getValorPieza();
    }
```

PHP

```
   //Calcula la expresión convertida en piezas de ejecución
   function Calcular()
   {
      $valorA=0;
      $valorB=0;
      $totalPiezaEjecuta = sizeof($this->PiezaEjecuta);

      for ($cont = 0; $cont < $totalPiezaEjecuta; $cont++)
      {
         switch ($this->PiezaEjecuta[$cont]->getTipoOperA())
         {
          case 5: $valorA = $this->PiezaEjecuta[$cont]->getNumeroA(); break; //¿Es un número?
          case 6: $valorA = $this->VariableAlgebra[$this->PiezaEjecuta[$cont]->getVariableA()]; break;  //¿Es una variable?
          case 7: $valorA = $this->PiezaEjecuta[$this->PiezaEjecuta[$cont]->getAcumulaA()]->getValorPieza(); break; //¿Es una
expresión anterior?
         }
         if (is_nan($valorA) || is_infinite($valorA)) return $valorA;

         switch ($this->PiezaEjecuta[$cont]->getFuncion())
         {
          case 0:
            switch ($this->PiezaEjecuta[$cont]->getTipoOperB())
            {
               case 5: $valorB = $this->PiezaEjecuta[$cont]->getNumeroB(); break; //¿Es un número?
               case 6: $valorB = $this->VariableAlgebra[$this->PiezaEjecuta[$cont]->getVariableB()]; break;  //¿Es una
variable?
               case 7: $valorB = $this->PiezaEjecuta[$this->PiezaEjecuta[$cont]->getAcumulaB()]->getValorPieza(); break; //¿Es
una expresión anterior?
            }
            if (is_nan($valorB) || is_infinite($valorB)) return $valorB;

            switch ($this->PiezaEjecuta[$cont]->getOperador())
```

```
                {
                    case '#': $this->PiezaEjecuta[$cont]->setValorPieza($valorA * $valorB); break;
                    case '+': $this->PiezaEjecuta[$cont]->setValorPieza($valorA + $valorB); break;
                    case '-': $this->PiezaEjecuta[$cont]->setValorPieza($valorA - $valorB); break;
                    case '*': $this->PiezaEjecuta[$cont]->setValorPieza($valorA * $valorB); break;
                    case '/': if ($valorB == 0) return NAN; else $this->PiezaEjecuta[$cont]->setValorPieza($valorA / $valorB);
break;
                    case '^': $this->PiezaEjecuta[$cont]->setValorPieza(pow($valorA, $valorB)); break;
                }
                break;
            case 1:
            case 2: $this->PiezaEjecuta[$cont]->setValorPieza(sin($valorA)); break;
            case 3: $this->PiezaEjecuta[$cont]->setValorPieza(cos($valorA)); break;
            case 4: $this->PiezaEjecuta[$cont]->setValorPieza(tan($valorA)); break;
            case 5: $this->PiezaEjecuta[$cont]->setValorPieza(abs($valorA)); break;
            case 6: $this->PiezaEjecuta[$cont]->setValorPieza(asin($valorA)); break;
            case 7: $this->PiezaEjecuta[$cont]->setValorPieza(acos($valorA)); break;
            case 8: $this->PiezaEjecuta[$cont]->setValorPieza(atan($valorA)); break;
            case 9: $this->PiezaEjecuta[$cont]->setValorPieza(log($valorA)); break;
            case 10: $this->PiezaEjecuta[$cont]->setValorPieza(ceil($valorA)); break;
            case 11: $this->PiezaEjecuta[$cont]->setValorPieza(exp($valorA)); break;
            case 12: $this->PiezaEjecuta[$cont]->setValorPieza(sqrt($valorA)); break;
            case 13: $this->PiezaEjecuta[$cont]->setValorPieza(pow($valorA, 0.333333333333)); break;
          }
      }
      return $this->PiezaEjecuta[$totalPiezaEjecuta - 1]->getValorPieza();
   }
```

Un ejemplo de uso del evaluador de expresiones

Para probar el evaluador de expresiones, se presenta a continuación una serie de programas en cada lenguaje de programación en el que se prueba todas las características del evaluador: chequeo de sintaxis, uso del menos unario, paréntesis, funciones y operadores.

Al final de este libro están los anexos con el código completo en cada lenguaje de programación, debe respetar el nombre de los archivos o hacer uso correcto de la funcionalidad de refactorización de su IDE (Integrated Development Environment) favorito.

Java

```
public class Evaluador
{
  public static void main(String[] args)
  {
      Evaluar evaluadorExpresiones = new Evaluar();
      int NumPruebas = 51;

      //En este arreglo de strings guarda las diversas expresiones
      String[] exprAlgebraica = new String[NumPruebas];

      //Aquí se ponen los resultados probados con Excel o WolframAlpha para chequear que el evaluador funciona correctamente
      double[] resultado = new double[NumPruebas];

      //Estas expresiones presentan fallas sintácticas. Sirve para probar el chequeador de sintaxis
      exprAlgebraica[0] =  " (7*8*--3/5)/4  ";
      exprAlgebraica[1] =  " 5.31-(4.6*)+3 ";
      exprAlgebraica[2] =  " 7+3.4- ";
      exprAlgebraica[3] =  "  5-(*9.4/2)+1 ";
      exprAlgebraica[4] =  "  /5.67*12-6+4 ";
      exprAlgebraica[5] =  "  3-(2*4) ) ";
      exprAlgebraica[6] =  "  7+((4-3) ";
      exprAlgebraica[7] =  "  7- 90.87 * (   ) +9.01";
      exprAlgebraica[8] =  "  2+3)-2)*3+((4";
      exprAlgebraica[9] =  " (3-5)7-(1+2)";
      exprAlgebraica[10] =  " 7-2(5-6) + 8";
      exprAlgebraica[11] =  "  3-2..4+1 ";
      exprAlgebraica[12] =  "  2.5+78.23.1-4 ";
      exprAlgebraica[13] =  "  (12-4)y-1 ";
      exprAlgebraica[14] =  "  4-z.1+3 ";
      exprAlgebraica[15] =  " 7-2.p+1 ";
      exprAlgebraica[16] =  "  3x+1";
      exprAlgebraica[17] =  "  x21+4 ";
      exprAlgebraica[18] =  "  7+abrg-8";
      exprAlgebraica[19] =  "  5*alo(78) ";
      exprAlgebraica[20] =  "  5+tr-xc+5  ";
      exprAlgebraica[21] =  "  5-a(7+3)";
      exprAlgebraica[22] =  "  (4-5)(2*x) ";
      exprAlgebraica[23] =  " -.3+7  ";
      exprAlgebraica[24] =  " 3*(.5+4) ";
      exprAlgebraica[25] =  " 7+3.(2+6) ";
      exprAlgebraica[26] =  "  (4+5).7-2 ";
      exprAlgebraica[27] =  " 5.*9+1  ";
      exprAlgebraica[28] =  " (3+2.)*5 ";

      //Estas expresiones son correctas sintácticamente
      exprAlgebraica[29] =  " --5 "; resultado[29] = 5;
      exprAlgebraica[30] =  " -(-(-(-(-(-(-1^(-(-(-(-1)))))))))"; resultado[30] = -1;
      exprAlgebraica[31] =  "  --(--(--(--(--(--(--(--1^(--(--(--(--1)))))))))"; resultado[31] = 1;
      exprAlgebraica[32] =  " --1*-2*-3*(--4*-2/-4/-2)/-3^-1 "; resultado[32] = 18;
      exprAlgebraica[33] =  " -1-(-2-(-3)) "; resultado[33] = -2;
      exprAlgebraica[34] =  " -1^2-(-2^2-(-3^2)) "; resultado[34] = 6;
      exprAlgebraica[35] =  " -1^-2-(-2^-2-(-3^-2)) "; resultado[35] = 0.861111111111111;
      exprAlgebraica[36] =  " -1^-3-(-2^-3-(-3^-3)) "; resultado[36] = -0.912037037037037;
      exprAlgebraica[37] =  " (--1) "; resultado[37] = 1;
      exprAlgebraica[38] =  " --1 "; resultado[38] = 1;
      exprAlgebraica[39] =  " --3--2--4--5--6 "; resultado[39] = 20;
      exprAlgebraica[40] =  " --3^-2--2^-2--4^-2--5^-2--6^-2 "; resultado[40] = -0.269166666666667;
      exprAlgebraica[41] =  " ((((-1)*-1)*-1)*-1) "; resultado[41] = 1;
      exprAlgebraica[42] =  " 0/2/3/4/5/6/-7 "; resultado[42] = 0;
      exprAlgebraica[43] =  " 0/-1 "; resultado[43] = 0;
      exprAlgebraica[44] =  " --2 "; resultado[44] = 2;
      exprAlgebraica[45] =  " --2^2 "; resultado[45] = 4;
      exprAlgebraica[46] =  " -(-2^3) "; resultado[46] = 8;
      exprAlgebraica[47] =  "  -(-2^-3)"; resultado[47] = 0.125;
      exprAlgebraica[48] =  " X-(Y-(Z-(-1*X)*y)*z) "; resultado[48] = 2084;
      exprAlgebraica[49] =  " ASN(-0.67)+ACS(-0.3*-0.11)+ATN(-4.5/-2.3)+EXP(-2)-SQR(9.7315)-RCB(7)+LOG(7.223) + CEI(10.1) ";
resultado[49] = 9.98202019592338;
      exprAlgebraica[50] =  " -SEN(-12.78)+COS(-SEN(7.1)+ABS(-4.09))+ TAN(-3.4*-5.7)+ LOG(9.12-5.89) "; resultado[50] =
0.994984132031446;

      for (int cont=0; cont < exprAlgebraica.length; cont++)
      {
        System.out.println("\n<" + cont + "> Expresion inicial es: [" + exprAlgebraica[cont] + "]");

        //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
        String Transformado = evaluadorExpresiones.TransformaExpresion(exprAlgebraica[cont]);
        System.out.println("Transformada: [" + Transformado + "]");

        //Chequea la sintaxis de la expresión
        int chequeoSintaxis = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        if (chequeoSintaxis == 0)  //Si la sintaxis es correcta
        {
```

```
            //Transforma la expresión para aceptar los menos unarios agregando (0-1)#
            String ExprNegativos = evaluadorExpresiones.ArreglaNegativos(Transformado);
            System.out.println("Negativos unarios: [" + ExprNegativos + "]");

            //Analiza la expresión
            evaluadorExpresiones.Analizar(ExprNegativos);

            //Da valor a las variables
            evaluadorExpresiones.ValorVariable('x', 7);
            evaluadorExpresiones.ValorVariable('y',  13);
            evaluadorExpresiones.ValorVariable('z', 19);

            //Evalúa la expresión para retornar un valor
            double valor = evaluadorExpresiones.Calcular();

            //Compara el valor retornado con el esperado. Si falla el evaluador, este si condicional avisa
            if (Math.abs(valor - resultado[cont])>0.01)
              System.out.println("FALLA EN [" + ExprNegativos + "] Calcula: " + valor + " Esperado: " + resultado[cont]);

            //Si hay un fallo matemático se captura con este si condicional
            if (Double.isNaN(valor) || Double.isInfinite(valor))
              System.out.println("Error matemático");
            else //No hay fallo matemático, se muestra el valor
              System.out.println("Resultado es: " + valor);
          }
          else
            System.out.println("La validación es: " + evaluadorExpresiones.MensajeSintaxis(chequeoSintaxis));
        }
    }
}
```

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EvaluadorCS
{
  class Program
  {
    static void Main(string[] args)
    {
      Evaluar evaluadorExpresiones = new Evaluar();
      int NumPruebas = 51;

      //En este arreglo de strings guarda las diversas expresiones
      String[] exprAlgebraica = new String[NumPruebas];

      //Aquí se ponen los resultados probados con Excel o WolframAlpha para chequear que el evaluador funciona correctamente
      double[] resultado = new double[NumPruebas];

      //Estas expresiones presentan fallas sintácticas. Sirve para probar el chequeador de sintaxis
      exprAlgebraica[0] =  " (7*8*--3/5)/4  ";
      exprAlgebraica[1] =  " 5.31-(4.6*)+3 ";
      exprAlgebraica[2] =  " 7+3.4- ";
      exprAlgebraica[3] =  "  5-(*9.4/2)+1 ";
      exprAlgebraica[4] =  "  /5.67*12-6+4 ";
      exprAlgebraica[5] =  "  3-(2*4) ) ";
      exprAlgebraica[6] =  "  7+((4-3) ";
      exprAlgebraica[7] =  "  7- 90.87 * (   ) +9.01";
      exprAlgebraica[8] =  "  2+3)-2)*3+((4";
      exprAlgebraica[9] =  " (3-5)7-(1+2)";
      exprAlgebraica[10] =  " 7-2(5-6) + 8";
      exprAlgebraica[11] =  "  3-2..4+1 ";
      exprAlgebraica[12] =  "  2.5+78.23.1-4 ";
      exprAlgebraica[13] =  "  (12-4)y-1 ";
      exprAlgebraica[14] =  "  4-z.1+3 ";
      exprAlgebraica[15] =  " 7-2.p+1 ";
      exprAlgebraica[16] =  "  3x+1";
      exprAlgebraica[17] =  "  x21+4 ";
      exprAlgebraica[18] =  "  7+abrg-8";
      exprAlgebraica[19] =  "  5*alo(78) ";
      exprAlgebraica[20] =  "  5+tr-xc+5  ";
      exprAlgebraica[21] =  "  5-a(7+3)";
      exprAlgebraica[22] =  "  (4-5)(2*x) ";
      exprAlgebraica[23] =  " -.3+7  ";
      exprAlgebraica[24] =  " 3*(.5+4) ";
      exprAlgebraica[25] =  " 7+3.(2+6) ";
      exprAlgebraica[26] =  "  (4+5).7-2 ";
      exprAlgebraica[27] =  " 5.*9+1  ";
      exprAlgebraica[28] =  " (3+2.)*5 ";

      //Estas expresiones son correctas sintácticamente
      exprAlgebraica[29] =  " --5 "; resultado[29] = 5;
      exprAlgebraica[30] =  " -(-(-(-(-(-(-1^(-(-(-(-1)))))))))"; resultado[30] = -1;
      exprAlgebraica[31] =  "  --(--(--(--(--(--(--(--1^(--(--(--(--1)))))))))"; resultado[31] = 1;
```

```
      exprAlgebraica[32] = " --1*-2*-3*(--4*-2/-4/-2)/-3^-1 "; resultado[32] = 18;
      exprAlgebraica[33] = " -1-(-2-(-3)) "; resultado[33] = -2;
      exprAlgebraica[34] = " -1^2-(-2^2-(-3^2)) "; resultado[34] = 6;
      exprAlgebraica[35] = " -1^-2-(-2^-2-(-3^-2)) "; resultado[35] = 0.861111111111111;
      exprAlgebraica[36] = " -1^-3-(-2^-3-(-3^-3)) "; resultado[36] = -0.912037037037037;
      exprAlgebraica[37] = " (--1) "; resultado[37] = 1;
      exprAlgebraica[38] = " --1 "; resultado[38] = 1;
      exprAlgebraica[39] = " --3--2--4--5--6 "; resultado[39] = 20;
      exprAlgebraica[40] = " --3^-2--2^-2--4^-2--5^-2--6^-2 "; resultado[40] = -0.269166666666667;
      exprAlgebraica[41] = " ((((-1)*-1)*-1)*-1) "; resultado[41] = 1;
      exprAlgebraica[42] = " 0/2/3/4/5/6/-7 "; resultado[42] = 0;
      exprAlgebraica[43] = " 0/-1 "; resultado[43] = 0;
      exprAlgebraica[44] = " --2 "; resultado[44] = 2;
      exprAlgebraica[45] = " --2^2 "; resultado[45] = 4;
      exprAlgebraica[46] = " -(-2^3) "; resultado[46] = 8;
      exprAlgebraica[47] = "  -(-2^-3)"; resultado[47] = 0.125;
      exprAlgebraica[48] = " X-(Y-(Z-(-1*X)*y)*z) "; resultado[48] = 2084;
      exprAlgebraica[49] = " ASN(-0.67)+ACS(-0.3*-0.11)+ATN(-4.5/-2.3)+EXP(-2)-SQR(9.7315)-RCB(7)+LOG(7.223) + CEI(10.1) ";
resultado[49] = 9.98202019592338;
      exprAlgebraica[50] = " -SEN(-12.78)+COS(-SEN(7.1)+ABS(-4.09))+ TAN(-3.4*-5.7)+ LOG(9.12-5.89) "; resultado[50] =
0.994984132031446;


      for (int cont=0; cont < exprAlgebraica.Length; cont++)
      {
        Console.WriteLine("\n<" + cont + "> Expresion inicial es: [" + exprAlgebraica[cont] + "]");

        //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
        String Transformado = evaluadorExpresiones.TransformaExpresion(exprAlgebraica[cont]);
        Console.WriteLine("Transformada : [" + Transformado + "]");

        //Chequea la sintaxis de la expresión
        int chequeoSintaxis = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        if (chequeoSintaxis == 0)  //Si la sintaxis es correcta
        {
          //Transforma la expresión para aceptar los menos unarios agregando (0-1)#
          String ExprNegativos = evaluadorExpresiones.ArreglaNegativos(Transformado);
          Console.WriteLine("Negativos unarios: [" + ExprNegativos + "]");

          //Analiza la expresión
          evaluadorExpresiones.Analizar(ExprNegativos);

          //Da valor a las variables
          evaluadorExpresiones.ValorVariable('x', 7);
          evaluadorExpresiones.ValorVariable('y',  13);
          evaluadorExpresiones.ValorVariable('z', 19);

          //Evalúa la expresión para retornar un valor
          double valor = evaluadorExpresiones.Calcular();

          //Compara el valor retornado con el esperado. Si falla el evaluador, este condicional avisa
          if (Math.Abs(valor - resultado[cont])>0.01)
            Console.WriteLine("FALLA EN [" + ExprNegativos + "] Calculado: " + valor + " Esperado: " + resultado[cont]);

          //Si hay un fallo matemático se captura con este si condicional
          if (Double.IsNaN(valor) || Double.IsInfinity(valor))
            Console.WriteLine("Error matemático");
          else //No hay fallo matemático, se muestra el valor
            Console.WriteLine("Resultado es: " + valor);
        }
        else
          Console.WriteLine("La validación es: " + evaluadorExpresiones.MensajeSintaxis(chequeoSintaxis));
      }
      Console.ReadKey();
    }
  }
}
```

Visual Basic .NET

```
Module Module1

  Sub Main()
    Dim evaluadorExpresiones As New Evaluar()
    Dim NumPruebas As Integer = 51

    'En este arreglo de strings guarda las diversas expresiones
    Dim exprAlgebraica As [String]() = New [String](NumPruebas - 1) {}

    'Aquí se ponen los resultados probados con Excel o WolframAlpha para chequear que el evaluador funciona correctamente
    Dim resultado As Double() = New Double(NumPruebas - 1) {}

    'Estas expresiones presentan fallas sintácticas. Sirve para probar el chequeador de sintaxis
    exprAlgebraica(0) = " (7*8*--3/5)/4  "
    exprAlgebraica(1) = " 5.31-(4.6*)+3 "
    exprAlgebraica(2) = " 7+3.4- "
    exprAlgebraica(3) = "  5-(*9.4/2)+1 "
    exprAlgebraica(4) = "  /5.67*12-6+4 "
    exprAlgebraica(5) = "  3-(2*4) ) "
```

```
    exprAlgebraica(6) = "  7+((4-3) "
    exprAlgebraica(7) = "  7- 90.87 * (     ) +9.01"
    exprAlgebraica(8) = "  2+3)-2)*3+((4"
    exprAlgebraica(9) = " (3-5)7-(1+2)"
    exprAlgebraica(10) = " 7-2(5-6) + 8"
    exprAlgebraica(11) = "  3-2..4+1 "
    exprAlgebraica(12) = "  2.5+78.23.1-4 "
    exprAlgebraica(13) = "  (12-4)y-1 "
    exprAlgebraica(14) = "  4-z.1+3 "
    exprAlgebraica(15) = " 7-2.p+1 "
    exprAlgebraica(16) = "  3x+1"
    exprAlgebraica(17) = "  x21+4 "
    exprAlgebraica(18) = "  7+abrg-8"
    exprAlgebraica(19) = "  5*alo(78) "
    exprAlgebraica(20) = "  5+tr-xc+5  "
    exprAlgebraica(21) = "  5-a(7+3)"
    exprAlgebraica(22) = "  (4-5)(2*x) "
    exprAlgebraica(23) = " -.3+7  "
    exprAlgebraica(24) = " 3*(.5+4) "
    exprAlgebraica(25) = " 7+3.(2+6) "
    exprAlgebraica(26) = "  (4+5).7-2 "
    exprAlgebraica(27) = " 5.*9+1  "
    exprAlgebraica(28) = " (3+2.)*5 "

    'Estas expresiones son correctas sintácticamente
    exprAlgebraica(29) = " --5 "
    resultado(29) = 5
    exprAlgebraica(30) = " -(-(-(-(-(-(-1^(-(-(-(-1)))))))))) "
    resultado(30) = -1
    exprAlgebraica(31) = "  --(--(--(--(--(--(--1^(--(--(--(--1)))))))))) "
    resultado(31) = 1
    exprAlgebraica(32) = " --1*-2*-3*(--4*-2/-4/-2)/-3^-1 "
    resultado(32) = 18
    exprAlgebraica(33) = " -1-(-2-(-3)) "
    resultado(33) = -2
    exprAlgebraica(34) = " -1^2-(-2^2-(-3^2)) "
    resultado(34) = 6
    exprAlgebraica(35) = " -1^-2-(-2^-2-(-3^-2)) "
    resultado(35) = 0.861111111111111
    exprAlgebraica(36) = " -1^-3-(-2^-3-(-3^-3)) "
    resultado(36) = -0.912037037037037
    exprAlgebraica(37) = " (--1) "
    resultado(37) = 1
    exprAlgebraica(38) = " --1 "
    resultado(38) = 1
    exprAlgebraica(39) = " --3--2--4--5--6 "
    resultado(39) = 20
    exprAlgebraica(40) = " --3^-2--2^-2--4^-2--5^-2--6^-2 "
    resultado(40) = -0.269166666666667
    exprAlgebraica(41) = " ((((-1)*-1)*-1)*-1) "
    resultado(41) = 1
    exprAlgebraica(42) = " 0/2/3/4/5/6/-7 "
    resultado(42) = 0
    exprAlgebraica(43) = " 0/-1 "
    resultado(43) = 0
    exprAlgebraica(44) = " --2 "
    resultado(44) = 2
    exprAlgebraica(45) = " --2^2 "
    resultado(45) = 4
    exprAlgebraica(46) = " -(-2^3) "
    resultado(46) = 8
    exprAlgebraica(47) = "  -(-2^-3)"
    resultado(47) = 0.125
    exprAlgebraica(48) = " X-(Y-(Z-(-1*X)*y)*z) "
    resultado(48) = 2084
    exprAlgebraica(49) = " ASN(-0.67)+ACS(-0.3*-0.11)+ATN(-4.5/-2.3)+EXP(-2)-SQR(9.7315)-RCB(7)+LOG(7.223) + CEI(10.1) "
    resultado(49) = 9.98202019592338
    exprAlgebraica(50) = " -SEN(-12.78)+COS(-SEN(7.1)+ABS(-4.09))+ TAN(-3.4*-5.7)+ LOG(9.12-5.89) "
    resultado(50) = 0.994984132031446


    For cont As Integer = 0 To exprAlgebraica.Length - 1
      Console.WriteLine((vbLf & "<" & cont & "> Expresion inicial es: [") + exprAlgebraica(cont) & "]")

      'Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
      Dim Transformado As [String] = evaluadorExpresiones.TransformaExpresion(exprAlgebraica(cont))
      Console.WriteLine("Transformada : [" & Transformado & "]")

      'Chequea la sintaxis de la expresión
      Dim chequeoSintaxis As Integer = evaluadorExpresiones.EvaluaSintaxis(Transformado)
      If chequeoSintaxis = 0 Then   'Si la sintaxis es correcta

        'Transforma la expresión para aceptar los menos unarios agregando (0-1)#
        Dim ExprNegativos As [String] = evaluadorExpresiones.ArreglaNegativos(Transformado)
        Console.WriteLine("Negativos unarios: [" & ExprNegativos & "]")

        'Analiza la expresión
        evaluadorExpresiones.Analizar(ExprNegativos)

        'Da valor a las variables
        evaluadorExpresiones.ValorVariable("x", 7)
        evaluadorExpresiones.ValorVariable("y", 13)
```

```
        evaluadorExpresiones.ValorVariable("z", 19)

        'Evalúa la expresión para retornar un valor
        Dim valor As Double = evaluadorExpresiones.Calcular()

        'Compara el valor retornado con el esperado. Si falla el evaluador, este condicional avisa
        If Math.Abs(valor - resultado(cont)) > 0.01 Then
          Console.WriteLine("FALLA EN [" & ExprNegativos & "] Calculado: " & valor & " Esperado: " & resultado(cont))
        End If

        'Si hay un fallo matemático se captura con este si condicional
        If [Double].IsNaN(valor) OrElse [Double].IsInfinity(valor) Then
          Console.WriteLine("Error matemático")
        Else 'No hay fallo matemático, se muestra el valor
          Console.WriteLine("Resultado es: " & valor)
        End If
      Else
        Console.WriteLine("La validación es: " & evaluadorExpresiones.MensajeSintaxis(chequeoSintaxis))
      End If
    Next
    Console.ReadKey()
  End Sub

End Module
```

C++

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "Evaluar.h"

int main(void);

int main()
{
      Evaluar evaluadorExpresiones;

      //En este arreglo de strings guarda las diversas expresiones
      char exprAlgebraica[51][200];

      //Aquí se ponen los resultados probados con Excel o WolframAlpha para chequear que el evaluador funciona correctamente
      double resultado[51];

      //Estas expresiones presentan fallas sintácticas. Sirve para probar el chequeador de sintaxis
      strcpy(exprAlgebraica[0], " (7*8*--3/5)/4  ");
      strcpy(exprAlgebraica[1], " 5.31-(4.6*)+3 ");
      strcpy(exprAlgebraica[2], " 7+3.4- ");
      strcpy(exprAlgebraica[3], "  5-(*9.4/2)+1 ");
      strcpy(exprAlgebraica[4],  "  /5.67*12-6+4 ");
      strcpy(exprAlgebraica[5],  "  3-(2*4) ) ");
      strcpy(exprAlgebraica[6],  "  7+((4-3) ");
      strcpy(exprAlgebraica[7],  "  7- 90.87 * (   ) +9.01");
      strcpy(exprAlgebraica[8],  "  2+3)-2)*3+((4");
      strcpy(exprAlgebraica[9],  " (3-5)7-(1+2)");
      strcpy(exprAlgebraica[10],  " 7-2(5-6) + 8");
      strcpy(exprAlgebraica[11],  "  3-2..4+1 ");
      strcpy(exprAlgebraica[12],  "  2.5+78.23.1-4 ");
      strcpy(exprAlgebraica[13],  "  (12-4)y-1 ");
      strcpy(exprAlgebraica[14],  "  4-z.1+3 ");
      strcpy(exprAlgebraica[15],  " 7-2.p+1 ");
      strcpy(exprAlgebraica[16],  "  3x+1");
      strcpy(exprAlgebraica[17],  "  x21+4 ");
      strcpy(exprAlgebraica[18],  "  7+abrg-8");
      strcpy(exprAlgebraica[19],  "  5*alo(78) ");
      strcpy(exprAlgebraica[20],  "  5+tr-xc+5  ");
      strcpy(exprAlgebraica[21], "  5-a(7+3)");
      strcpy(exprAlgebraica[22], "  (4-5)(2*x) ");
      strcpy(exprAlgebraica[23], " -.3+7  ");
      strcpy(exprAlgebraica[24], " 3*(.5+4) ");
      strcpy(exprAlgebraica[25], " 7+3.(2+6) ");
      strcpy(exprAlgebraica[26], "  (4+5).7-2 ");
      strcpy(exprAlgebraica[27], " 5.*9+1  ");
      strcpy(exprAlgebraica[28], " (3+2.)*5 ");

      //Estas expresiones son correctas sintácticamente
      strcpy(exprAlgebraica[29], " --5 "); resultado[29] = 5;
      strcpy(exprAlgebraica[30], " -(-(-(-(-(-(-1^(-(-(-(-1)))))))))) "); resultado[30] = -1;
      strcpy(exprAlgebraica[31], "  --(--(--(--(--(--(--1^(--(--(--(--1)))))))))) "); resultado[31] = 1;
      strcpy(exprAlgebraica[32], " --1*-2*-3*(--4*-2/-4/-2)/-3^-1 "); resultado[32] = 18;
      strcpy(exprAlgebraica[33], " -1-(-2-(-3)) "); resultado[33] = -2;
      strcpy(exprAlgebraica[34], " -1^2-(-2^2-(-3^2)) "); resultado[34] = 6;
      strcpy(exprAlgebraica[35], " -1^-2-(-2^-2-(-3^-2)) "); resultado[35] = 0.861111111111111;
      strcpy(exprAlgebraica[36], " -1^-3-(-2^-3-(-3^-3)) "); resultado[36] = -0.912037037037037;
      strcpy(exprAlgebraica[37], " (--1) "); resultado[37] = 1;
      strcpy(exprAlgebraica[38], " --1 "); resultado[38] = 1;
      strcpy(exprAlgebraica[39], " --3--2--4--5--6 "); resultado[39] = 20;
      strcpy(exprAlgebraica[40], " --3^-2--2^-2--4^-2--5^-2--6^-2 "); resultado[40] = -0.269166666666667;
      strcpy(exprAlgebraica[41], " ((((-1)*-1)*-1)*-1) "); resultado[41] = 1;
      strcpy(exprAlgebraica[42], " 0/2/3/4/5/6/-7 "); resultado[42] = 0;
```

```
      strcpy(exprAlgebraica[43], " 0/-1 "); resultado[43] = 0;
      strcpy(exprAlgebraica[44], " --2 "); resultado[44] = 2;
      strcpy(exprAlgebraica[45], " --2^2 "); resultado[45] = 4;
      strcpy(exprAlgebraica[46], " -(-2^3) "); resultado[46] = 8;
      strcpy(exprAlgebraica[47], "  -(-2^-3)"); resultado[47] = 0.125;
      strcpy(exprAlgebraica[48], " X-(Y-(Z-(-1*X)*y)*z) "); resultado[48] = 2084;
      strcpy(exprAlgebraica[49], " ASN(-0.67)+ACS(-0.3*-0.11)+ATN(-4.5/-2.3)+EXP(-2)-SQR(9.7315)-RCB(7)+LOG(7.223) + CEI(10.1)
"); resultado[49] = 9.98202019592338;
      strcpy(exprAlgebraica[50], " -SEN(-12.78)+COS(-SEN(7.1)+ABS(-4.09))+ TAN(-3.4*-5.7)+ LOG(9.12-5.89) "); resultado[50] =
0.994984132031446;

      char Transformado[200], ExprNegativos[200], Mensaje[200];

      for (int cont=0; cont < 51; cont++)
      {
        for (int inicializa=0; inicializa<200; inicializa++) { Transformado[inicializa] = '\0'; ExprNegativos[inicializa] =
'\0'; Mensaje[inicializa] = '\0'; }

        printf("\n<%d>Expresión inicial es : [%s]\n", cont, exprAlgebraica[cont]);

        //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
        evaluadorExpresiones.TransformaExpresion(Transformado, exprAlgebraica[cont]);
        printf("Transformada: [%s]\n", Transformado);

        //Chequea la sintaxis de la expresión
        int chequeoSintaxis = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        if (chequeoSintaxis == 0) //Si la sintaxis es correcta
        {
          //Transforma la expresión para aceptar los menos unarios agregando (0-1)#
          if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) == -1)
          {
            printf("Fallo en reserva de memoria para convertir negativos\n");
            exit(0);
          }
          printf("Negativos unarios: [%s]\n", ExprNegativos);

          //Analiza la expresión
          evaluadorExpresiones.Analizar(ExprNegativos);

          //Da valor a las variables
          evaluadorExpresiones.ValorVariable('x', 7);
          evaluadorExpresiones.ValorVariable('y', 13);
          evaluadorExpresiones.ValorVariable('z', 19);

          //Evalúa la expresión para retornar un valor
          double valor = evaluadorExpresiones.Calcular();

          //Compara el valor retornado con el esperado. Si falla el evaluador, este si condicional avisa
          if (abs(valor - resultado[cont])>0.01)
            printf("FALLA EN [%s] Calculado: %f Esperado: %f\n", ExprNegativos, valor, resultado[cont]);

          //Si hay un fallo matemático se captura con este si condicional
          if (_isnan(valor) || !_finite(valor))
            printf("Error matemático\n");
          else //No hay fallo matemático, se muestra el valor
            printf("Resultado es: %f\n", valor);
        }
        else
        {
          evaluadorExpresiones.MensajeSintaxis(chequeoSintaxis, Mensaje);
          printf("La validación es: %s\n", Mensaje);
        }
      }
  getchar();
  return 0;
}
```

Object Pascal

```
procedure TfrmFunciones.btnEvaluarClick(Sender: TObject);
var
  evaluadorExpresiones: TEvaluar;
  exprAlgebraica: array[0..50] of string; //En este arreglo de strings guarda las diversas expresiones
  resultado: array[0..50] of double; //Aquí se ponen los resultados probados con Excel o WolframAlpha para chequear que el
evaluador funciona correctamente
  cont, chequeoSintaxis: integer;
  valor: double;
  Transformado, ExprNegativos: string;
begin
  evaluadorExpresiones :=  TEvaluar.Create;

  //Estas expresiones presentan fallas sintácticas. Sirve para probar el chequeador de sintaxis
  exprAlgebraica[0] :=  ' (7*8*--3/5)/4  ';
  exprAlgebraica[1] :=  ' 5.31-(4.6*)+3 ';
  exprAlgebraica[2] :=  ' 7+3.4- ';
  exprAlgebraica[3] :=  '  5-(*9.4/2)+1 ';
  exprAlgebraica[4] :=  '  /5.67*12-6+4 ';
  exprAlgebraica[5] :=  '  3-(2*4) ) ';
  exprAlgebraica[6] :=  '  7+((4-3) ';
  exprAlgebraica[7] :=  '  7- 90.87 * (     ) +9.01';
```

```
  exprAlgebraica[8] :=  '  2+3)-2)*3+((4';
  exprAlgebraica[9] :=  ' (3-5)7-(1+2)';
  exprAlgebraica[10] :=  ' 7-2(5-6) + 8';
  exprAlgebraica[11] :=  '  3-2..4+1 ';
  exprAlgebraica[12] :=  '  2.5+78.23.1-4 ';
  exprAlgebraica[13] :=  '  (12-4)y-1 ';
  exprAlgebraica[14] :=  '  4-z.1+3 ';
  exprAlgebraica[15] :=  ' 7-2.p+1 ';
  exprAlgebraica[16] :=  '  3x+1';
  exprAlgebraica[17] :=  '  x21+4 ';
  exprAlgebraica[18] :=  '  7+abrg-8';
  exprAlgebraica[19] :=  '  5*alo(78) ';
  exprAlgebraica[20] :=  '  5+tr-xc+5  ';
  exprAlgebraica[21] :=  '  5-a(7+3)';
  exprAlgebraica[22] :=  '  (4-5)(2*x) ';
  exprAlgebraica[23] :=  ' -.3+7  ';
  exprAlgebraica[24] :=  ' 3*(.5+4) ';
  exprAlgebraica[25] :=  ' 7+3.(2+6) ';
  exprAlgebraica[26] :=  '  (4+5).7-2 ';
  exprAlgebraica[27] :=  ' 5.*9+1  ';
  exprAlgebraica[28] :=  ' (3+2.)*5 ';

  //Estas expresiones son correctas sintácticamente
  exprAlgebraica[29] :=  ' --5 '; resultado[29] := 5;
  exprAlgebraica[30] :=  ' -(-(-(-(-(-(-1^(-(-(-(-1)))))))))))'; resultado[30] := -1;
  exprAlgebraica[31] :=  '  --(--(--(--(--(--(--1^(--(--(--(--1)))))))))))'; resultado[31] := 1;
  exprAlgebraica[32] :=  ' --1*-2*-3*(--4*-2/-4/-2)/-3^-1 '; resultado[32] := 18;
  exprAlgebraica[33] :=  ' -1-(-2-(-3)) '; resultado[33] := -2;
  exprAlgebraica[34] :=  ' -1^2-(-2^2-(-3^2)) '; resultado[34] := 6;
  exprAlgebraica[35] :=  ' -1^-2-(-2^-2-(-3^-2)) '; resultado[35] := 0.861111111111111;
  exprAlgebraica[36] :=  ' -1^-3-(-2^-3-(-3^-3)) '; resultado[36] := -0.912037037037037;
  exprAlgebraica[37] :=  ' (--1) '; resultado[37] := 1;
  exprAlgebraica[38] :=  ' --1 '; resultado[38] := 1;
  exprAlgebraica[39] :=  ' --3--2--4--5--6 '; resultado[39] := 20;
  exprAlgebraica[40] :=  ' --3^-2--2^-2--4^-2--5^-2--6^-2 '; resultado[40] := -0.269166666666667;
  exprAlgebraica[41] :=  ' ((((-1)*-1)*-1)*-1) '; resultado[41] := 1;
  exprAlgebraica[42] :=  ' 0/2/3/4/5/6/-7 '; resultado[42] := 0;
  exprAlgebraica[43] :=  ' 0/-1 '; resultado[43] := 0;
  exprAlgebraica[44] :=  ' --2 '; resultado[44] := 2;
  exprAlgebraica[45] :=  ' --2^2 '; resultado[45] := 4;
  exprAlgebraica[46] :=  ' -(-2^3) '; resultado[46] := 8;
  exprAlgebraica[47] :=  '  -(-2^-3)'; resultado[47] := 0.125;
  exprAlgebraica[48] :=  ' X-(Y-(Z-(-1*X)*y)*z) '; resultado[48] := 2084;
  exprAlgebraica[49] :=  ' ASN(-0.67)+ACS(-0.3*-0.11)+ATN(-4.5/-2.3)+EXP(-2)-SQR(9.7315)-RCB(7)+LOG(7.223) + CEI(10.1) ';
resultado[49] := 9.98202019592338;
  exprAlgebraica[50] :=  ' -SEN(-12.78)+COS(-SEN(7.1)+ABS(-4.09))+ TAN(-3.4*-5.7)+ LOG(9.12-5.89) '; resultado[50] :=
0.994984132031446;


  for cont := 0 to 50 do
  begin
    lstConsola.Items.Add(' ');
    lstConsola.Items.Add('Expresión inicial es   : [' + exprAlgebraica[cont] + ']');

    //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
    Transformado := evaluadorExpresiones.TransformaExpresion(exprAlgebraica[cont]);
    lstConsola.Items.Add('Transformada: [' + Transformado + ']');

    //Chequea la sintaxis de la expresión
    chequeoSintaxis := evaluadorExpresiones.EvaluaSintaxis(Transformado);
    if chequeoSintaxis = 0 then   //Si la sintaxis es correcta
    begin
      //Transforma la expresión para aceptar los menos unarios agregando (0-1)#
      ExprNegativos := evaluadorExpresiones.ArreglaNegativos(Transformado);
      lstConsola.Items.Add('Negativos unarios: [' + ExprNegativos + ']');

      //Analiza la expresión
      evaluadorExpresiones.Analizar(ExprNegativos);

      //Da valor a las variables
      evaluadorExpresiones.ValorVariable('x', 7);
      evaluadorExpresiones.ValorVariable('y', 13);
      evaluadorExpresiones.ValorVariable('z', 19);

      //Evalúa la expresión para retornar un valor
      valor := evaluadorExpresiones.Calcular();

      //Compara el valor retornado con el esperado. Si falla el evaluador, este si condicional avisa
      if (abs(valor - resultado[cont])>0.01) then
        lstConsola.Items.Add('FALLA EN [' + ExprNegativos + '] Calculado: ' + floattostr(valor) + ' Esperado: ' +
floattostr(resultado[cont]));

      //Si hay un fallo matemático se captura con este si condicional
      if isNaN(valor) then
        lstConsola.Items.Add('error matemático')
      else //No hay fallo matemático, se muestra el valor
        lstConsola.Items.Add('Resultado es: ' + floattostr(valor));
    end
    else
      lstConsola.Items.Add('La validación es: ' + evaluadorExpresiones.MensajeSintaxis(chequeoSintaxis));
  end;
end;
```

JavaScript

```
<html>
  <body>
  <script type="text/javascript" src="Evaluar.js"> </script>
  <script type="text/javascript">
      evaluadorExpresiones = new Evaluar();

      //En este arreglo de strings guarda las diversas expresiones
      var exprAlgebraica = new Array()

      //Aquí se ponen los resultados probados con Excel o WolframAlpha para chequear que el evaluador funciona correctamente
      var resultado = new Array()

      //Estas expresiones presentan fallas sintácticas. Sirve para probar el chequeador de sintaxis
      exprAlgebraica[0] =  " (7*8*--3/5)/4  ";
      exprAlgebraica[1] =  " 5.31-(4.6*)+3 ";
      exprAlgebraica[2] =  " 7+3.4- ";
      exprAlgebraica[3] =  "  5-(*9.4/2)+1 ";
      exprAlgebraica[4] =  "  /5.67*12-6+4 ";
      exprAlgebraica[5] =  "  3-(2*4) ) ";
      exprAlgebraica[6] =  "  7+((4-3) ";
      exprAlgebraica[7] =  "  7- 90.87 * (   ) +9.01";
      exprAlgebraica[8] =  "  2+3)-2)*3+((4";
      exprAlgebraica[9] =  " (3-5)7-(1+2)";
      exprAlgebraica[10] =  " 7-2(5-6) + 8";
      exprAlgebraica[11] =  "  3-2..4+1 ";
      exprAlgebraica[12] =  "  2.5+78.23.1-4 ";
      exprAlgebraica[13] =  "  (12-4)y-1 ";
      exprAlgebraica[14] =  "  4-z.1+3 ";
      exprAlgebraica[15] =  " 7-2.p+1 ";
      exprAlgebraica[16] =  "  3x+1";
      exprAlgebraica[17] =  "  x21+4 ";
      exprAlgebraica[18] =  "  7+abrg-8";
      exprAlgebraica[19] =  "  5*alo(78) ";
      exprAlgebraica[20] =  "  5+tr-xc+5  ";
      exprAlgebraica[21] =  "  5-a(7+3)";
      exprAlgebraica[22] =  "  (4-5)(2*x) ";
      exprAlgebraica[23] =  " -.3+7  ";
      exprAlgebraica[24] =  " 3*(.5+4) ";
      exprAlgebraica[25] =  " 7+3.(2+6) ";
      exprAlgebraica[26] =  "  (4+5).7-2 ";
      exprAlgebraica[27] =  " 5.*9+1  ";
      exprAlgebraica[28] =  " (3+2.)*5 ";

      //Estas expresiones son correctas sintácticamente
      exprAlgebraica[29] =  " --5 "; resultado[29] = 5;
      exprAlgebraica[30] =  " -(-(-(-(-(-(-1^(-(-(-(-1)))))))))"; resultado[30] = -1;
      exprAlgebraica[31] =  "  --(--(--(--(--(--(--1^(--(--(--(--1)))))))))"; resultado[31] = 1;
      exprAlgebraica[32] =  " --1*-2*-3*(--4*-2/-4/-2)/-3^-1 "; resultado[32] = 18;
      exprAlgebraica[33] =  " -1-(-2-(-3)) "; resultado[33] = -2;
      exprAlgebraica[34] =  " -1^2-(-2^2-(-3^2)) "; resultado[34] = 6;
      exprAlgebraica[35] =  " -1^-2-(-2^-2-(-3^-2)) "; resultado[35] = 0.861111111111111;
      exprAlgebraica[36] =  " -1^-3-(-2^-3-(-3^-3)) "; resultado[36] = -0.912037037037037;
      exprAlgebraica[37] =  " (--1) "; resultado[37] = 1;
      exprAlgebraica[38] =  " --1 "; resultado[38] = 1;
      exprAlgebraica[39] =  " --3--2--4--5--6 "; resultado[39] = 20;
      exprAlgebraica[40] =  " --3^-2--2^-2--4^-2--5^-2--6^-2 "; resultado[40] = -0.269166666666667;
      exprAlgebraica[41] =  " ((((-1)*-1)*-1)*-1) "; resultado[41] = 1;
      exprAlgebraica[42] =  " 0/2/3/4/5/6/-7 "; resultado[42] = 0;
      exprAlgebraica[43] =  " 0/-1 "; resultado[43] = 0;
      exprAlgebraica[44] =  " --2 "; resultado[44] = 2;
      exprAlgebraica[45] =  " --2^2 "; resultado[45] = 4;
      exprAlgebraica[46] =  " -(-2^3) "; resultado[46] = 8;
      exprAlgebraica[47] =  "  -(-2^-3)"; resultado[47] = 0.125;
      exprAlgebraica[48] =  " X-(Y-(Z-(-1*X)*y)*z) "; resultado[48] = 2084;
      exprAlgebraica[49] =  " ASN(-0.67)+ACS(-0.3*-0.11)+ATN(-4.5/-2.3)+EXP(-2)-SQR(9.7315)-RCB(7)+LOG(7.223) + CEI(10.1) ";
resultado[49] = 9.98202019592338;
      exprAlgebraica[50] =  " -SEN(-12.78)+COS(-SEN(7.1)+ABS(-4.09))+ TAN(-3.4*-5.7)+ LOG(9.12-5.89) "; resultado[50] =
0.994984132031446;


      for (var cont=0; cont < exprAlgebraica.length; cont++)
      {
        document.write("<br><br><" + cont + "> Expresion inicial es: [" + exprAlgebraica[cont] + "]");

        //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
        var Transformado = evaluadorExpresiones.TransformaExpresion(exprAlgebraica[cont]);
        document.write("<br>Transformada: [" + Transformado + "]");

        //Chequea la sintaxis de la expresión
        var chequeoSintaxis = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        if (chequeoSintaxis == 0)  //Si la sintaxis es correcta
        {
          //Transforma la expresión para aceptar los menos unarios agregando (0-1)#
          var ExprNegativos = evaluadorExpresiones.ArreglaNegativos(Transformado);
          document.write("<br>Negativos unarios: [" + ExprNegativos + "]");
```

```
            //Analiza la expresión
            evaluadorExpresiones.Analizar(ExprNegativos);

            //Da valor a las variables
            evaluadorExpresiones.ValorVariable('x', 7);
            evaluadorExpresiones.ValorVariable('y',  13);
            evaluadorExpresiones.ValorVariable('z', 19);

            //Evalúa la expresión para retornar un valor
            var valor = evaluadorExpresiones.Calcular();

            //Compara el valor retornado con el esperado. Si falla el evaluador, este si condicional avisa
            if (Math.abs(valor - resultado[cont])>0.01)
              document.write("<br>FALLA EN [" + ExprNegativos + "] Calculado: " + valor + " Esperado: " + resultado[cont]);

            //Si hay un fallo matemático se captura con este si condicional
            if (isNaN(valor) || !isFinite(valor))
              document.write("<br>Error matemático");
            else //No hay fallo matemático, se muestra el valor
              document.write("<br>Resultado es: " + valor);
          }
          else
            document.write("<br>La validación es: " + evaluadorExpresiones.MensajeSintaxis(chequeoSintaxis));
        }
      </script>
    </body>
</html>
```

PHP

```
<?php
include("Evaluar.php");
    $evaluadorExpresiones = new Evaluar();

    //Estas expresiones presentan fallas sintácticas. Sirve para probar el chequeador de sintaxis
    $exprAlgebraica[0] =  " (7*8*--3/5)/4  ";
    $exprAlgebraica[1] =  " 5.31-(4.6*)+3 ";
    $exprAlgebraica[2] =  " 7+3.4- ";
    $exprAlgebraica[3] =  "  5-(*9.4/2)+1 ";
    $exprAlgebraica[4] =  "  /5.67*12-6+4 ";
    $exprAlgebraica[5] =  "  3-(2*4) ) ";
    $exprAlgebraica[6] =  "  7+((4-3) ";
    $exprAlgebraica[7] =  "  7- 90.87 * (   ) +9.01";
    $exprAlgebraica[8] =  "  2+3)-2)*3+((4";
    $exprAlgebraica[9] =  " (3-5)7-(1+2)";
    $exprAlgebraica[10] =  " 7-2(5-6) + 8";
    $exprAlgebraica[11] =  "  3-2..4+1 ";
    $exprAlgebraica[12] =  "  2.5+78.23.1-4 ";
    $exprAlgebraica[13] =  "  (12-4)y-1 ";
    $exprAlgebraica[14] =  "  4-z.1+3 ";
    $exprAlgebraica[15] =  " 7-2.p+1 ";
    $exprAlgebraica[16] =  "  3x+1";
    $exprAlgebraica[17] =  "  x21+4 ";
    $exprAlgebraica[18] =  "  7+abrg-8";
    $exprAlgebraica[19] =  "  5*alo(78) ";
    $exprAlgebraica[20] =  "  5+tr-xc+5  ";
    $exprAlgebraica[21] =  "  5-a(7+3)";
    $exprAlgebraica[22] =  "  (4-5)(2*x) ";
    $exprAlgebraica[23] =  " -.3+7  ";
    $exprAlgebraica[24] =  " 3*(.5+4) ";
    $exprAlgebraica[25] =  " 7+3.(2+6) ";
    $exprAlgebraica[26] =  "  (4+5).7-2 ";
    $exprAlgebraica[27] =  " 5.*9+1  ";
    $exprAlgebraica[28] =  " (3+2.)*5 ";

    //Estas expresiones son correctas sintácticamente
    $exprAlgebraica[29] =  " --5 "; $resultado[29] = 5;
    $exprAlgebraica[30] =  " -(-(-(-(-(-(-1^(-(-(-(-1)))))))))"; $resultado[30] = -1;
    $exprAlgebraica[31] =  "  --(--(--(--(--(--(--1^(--(--(--(--1)))))))))"; $resultado[31] = 1;
    $exprAlgebraica[32] =  " --1*-2*-3*(--4*-2/-4/-2)/-3^-1 "; $resultado[32] = 18;
    $exprAlgebraica[33] =  " -1-(-2-(-3)) "; $resultado[33] = -2;
    $exprAlgebraica[34] =  " -1^2-(-2^2-(-3^2)) "; $resultado[34] = 6;
    $exprAlgebraica[35] =  " -1^-2-(-2^-2-(-3^-2)) "; $resultado[35] = 0.861111111111111;
    $exprAlgebraica[36] =  " -1^-3-(-2^-3-(-3^-3)) "; $resultado[36] = -0.912037037037037;
    $exprAlgebraica[37] =  " (--1) "; $resultado[37] = 1;
    $exprAlgebraica[38] =  " --1 "; $resultado[38] = 1;
    $exprAlgebraica[39] =  " --3--2--4--5--6 "; $resultado[39] = 20;
    $exprAlgebraica[40] =  " --3^-2--2^-2--4^-2--5^-2--6^-2 "; $resultado[40] = -0.269166666666667;
    $exprAlgebraica[41] =  " ((((-1)*-1)*-1)*-1) "; $resultado[41] = 1;
    $exprAlgebraica[42] =  " 0/2/3/4/5/6/-7 "; $resultado[42] = 0;
    $exprAlgebraica[43] =  " 0/-1 "; $resultado[43] = 0;
    $exprAlgebraica[44] =  " --2 "; $resultado[44] = 2;
    $exprAlgebraica[45] =  " --2^2 "; $resultado[45] = 4;
    $exprAlgebraica[46] =  " -(-2^3) "; $resultado[46] = 8;
    $exprAlgebraica[47] =  "  -(-2^-3)"; $resultado[47] = 0.125;
    $exprAlgebraica[48] =  " X-(Y-(Z-(-1*X)*y)*z) "; $resultado[48] = 2084;
    $exprAlgebraica[49] =  " ASN(-0.67)+ACS(-0.3*-0.11)+ATN(-4.5/-2.3)+EXP(-2)-SQR(9.7315)-RCB(7)+LOG(7.223) + CEI(10.1) ";
$resultado[49] = 9.98202019592338;
    $exprAlgebraica[50] =  " -SEN(-12.78)+COS(-SEN(7.1)+ABS(-4.09))+ TAN(-3.4*-5.7)+ LOG(9.12-5.89) "; $resultado[50] =
0.994984132031446;
```

```php
for ($cont=0; $cont < 51; $cont++)
{
  echo "<br><br><" . $cont . "> Expresion inicial es: [" . $exprAlgebraica[$cont] . "]";

  //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
  $Transformado = $evaluadorExpresiones->TransformaExpresion($exprAlgebraica[$cont]);
  echo "<br>Transformada: [" . $Transformado . "]";

  //Chequea la sintaxis de la expresión
  $chequeoSintaxis = $evaluadorExpresiones->EvaluaSintaxis($Transformado);
  if ($chequeoSintaxis == 0)  //Si la sintaxis es correcta
  {
    //Transforma la expresión para aceptar los menos unarios agregando (0-1)#
    $ExprNegativos = $evaluadorExpresiones->ArreglaNegativos($Transformado);
    echo "<br>Negativos unarios: [" . $ExprNegativos . "]";

    //Analiza la expresión
    $evaluadorExpresiones->Analizar($ExprNegativos);

    //Da valor a las variables
    $evaluadorExpresiones->ValorVariable('x', 7);
    $evaluadorExpresiones->ValorVariable('y',  13);
    $evaluadorExpresiones->ValorVariable('z', 19);

    //Evalúa la expresión para retornar un valor
    $valor = $evaluadorExpresiones->Calcular();

    //Compara el valor retornado con el esperado. Si falla el evaluador, este si condicional avisa
    if (abs($valor - $resultado[$cont])>0.01)
        echo "<br>FALLA EN [" . $ExprNegativos . "] Calculado: " . $valor . " Esperado: " . $resultado[$cont];

    //Si hay un fallo matemático se captura con este si condicional
    if (is_nan($valor) || is_infinite($valor))
        echo "<br>Error matemático";
    else  //No hay fallo matemático, se muestra el valor
        echo "<br>Resultado es: " . $valor;
  }
  else
    echo "<br>La validación es: " . $evaluadorExpresiones->MensajeSintaxis($chequeoSintaxis);
}
?>
```

Conclusiones

Resolver un problema en forma algorítmica rara vez se detiene, siempre se encontrará una mejor manera de hacerlo sea por la experiencia del desarrollador, la nueva tecnología disponible o en este caso en particular, nació como idea gracias al desarrollo de un software de algoritmos genéticos. Por otro lado, hay que ser cuidadoso con la palabra mejorar, porque se puede hacer el software con mejor desempeño pero consumiendo más memoria o haciéndolo más complejo y difícil de mantener. Este nuevo desarrollador es mejor en desempeño, con más funcionalidades y menos complejo que la versión anterior. ¿Puede haber una versión nueva? Con certeza diré que sí, me interesa personalmente hacer que el software que tenga mayor velocidad de ejecución.

Invito al lector a que busque una solución más rápida: favor descargue el libro anterior (2012) y observe como nació la versión 1.0 (un algoritmo completamente distinto). Quizás estudiando los dos algoritmos, nazca un tercero mejor.

Igualmente, si se detecta algún problema con el evaluador, favor informarme a enginelife@hotmail.com para poder evaluar el problema y dado el caso dar una solución actualizando este libro.

Anexo 1. Código completo en Java

Para probar el software directamente, se muestra el nombre del archivo y su contenido. Debe compilar.

El código fuente está listo para descarga aquí: http://darwin.50webs.com/Espanol/Evaluador.htm

Pieza_Simple.java

```
public class Pieza_Simple
{
  private int tipo; //Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
  private int funcion; //Que función es seno/coseno/tangente/sqrt
  private char operador; // +, -, *, /, ^
  private double numero; //Número real de la expresión
  private int variableAlgebra; //Variable de la expresión
  private int acumula;  //Indice de la microexpresión

  public final int getTipo() {  return this.tipo;  }
  public final int getFuncion() { return this.funcion; }
  public final char getOperador() { return this.operador; }
  public final double getNumero() { return this.numero; }
  public final int getVariable() { return this.variableAlgebra; }
  public final int getAcumula() { return this.acumula; }
  public final void setAcumula(int acumula) { this.tipo = 7; this.acumula = acumula; }

  public Pieza_Simple(int tipo, int funcion, char operador, double numero, int variable)
  {
    this.tipo = tipo;
    this.funcion = funcion;
    this.operador = operador;
    this.variableAlgebra = variable;
    this.acumula = 0;
    this.numero = numero;
  }
}
```

Pieza_Ejecuta.java

```
public class Pieza_Ejecuta
{
  private double valorPieza;

  private int funcion;

  private int tipo_operandoA;
  private double numeroA;
  private int variableA;
  private int acumulaA;

  private char operador;

  private int tipo_operandoB;
  private double numeroB;
  private int variableB;
  private int acumulaB;

  public final double getValorPieza() { return this.valorPieza; }
  public final void setValorPieza(double valor) { this.valorPieza = valor; }
  public final int getFuncion() { return this.funcion; }
  public final int getTipoOperA() { return this.tipo_operandoA; }
  public final double getNumeroA() { return this.numeroA; }
  public final int getVariableA() { return this.variableA; }
  public final int getAcumulaA() { return this.acumulaA; }
  public final char getOperador() { return this.operador; }
  public final int getTipoOperB() { return this.tipo_operandoB; }
  public final double getNumeroB() { return this.numeroB; }
  public final int getVariableB() { return this.variableB; }
  public final int getAcumulaB() { return this.acumulaB; }

  public Pieza_Ejecuta(int funcion, int tipo_operandoA, double numeroA, int variableA, int acumulaA, char operador, int
tipo_operandoB, double numeroB, int variableB, int acumulaB)
  {
    this.valorPieza = 0;

    this.funcion = funcion;

    this.tipo_operandoA = tipo_operandoA;
    this.numeroA = numeroA;
    this.variableA = variableA;
    this.acumulaA = acumulaA;

    this.operador = operador;

    this.tipo_operandoB = tipo_operandoB;
    this.numeroB = numeroB;
    this.variableB = variableB;
    this.acumulaB = acumulaB;
  }
}
```

Evaluar.java

```
import java.util.ArrayList;

public class Evaluar
{
  /* Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 */
  private static final int ASCIINUMERO = 48;

  /* Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 */
  private static final int ASCIILETRA = 97;

  /* Las funciones que soporta este evaluador */
  private static final int TAMANOFUNCION = 39;
  private static final String listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

  /* 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;

  //Listado de Piezas de análisis
  private ArrayList<Pieza_Simple> PiezaSimple = new ArrayList<Pieza_Simple>();

  //Listado de Piezas de ejecución
  private ArrayList<Pieza_Ejecuta> PiezaEjecuta = new ArrayList<Pieza_Ejecuta>();
  private int Contador_Acumula = 0;

  //Almacena los valores de las 26 diferentes variables que puede tener la expresión algebraica
  private double[] VariableAlgebra = new double[26];

  //Valida la expresión algebraica
  public final int EvaluaSintaxis(String expresion)
  {
    //Hace 25 pruebas de sintaxis
    if (DobleTripleOperadorSeguido(expresion)) return 1;
    if (OperadorParentesisCierra(expresion)) return 2;
    if (ParentesisAbreOperador(expresion)) return 3;
    if (ParentesisDesbalanceados(expresion)) return 4;
    if (ParentesisVacio(expresion)) return 5;
    if (ParentesisBalanceIncorrecto(expresion)) return 6;
    if (ParentesisCierraNumero(expresion)) return 7;
    if (NumeroParentesisAbre(expresion)) return 8;
    if (DoblePuntoNumero(expresion)) return 9;
    if (ParentesisCierraVariable(expresion)) return 10;
    if (VariableluegoPunto(expresion)) return 11;
    if (PuntoluegoVariable(expresion)) return 12;
    if (NumeroAntesVariable(expresion)) return 13;
    if (VariableDespuesNumero(expresion)) return 14;
    if (Chequea4letras(expresion)) return 15;
    if (FuncionInvalida(expresion)) return 16;
    if (VariableInvalida(expresion)) return 17;
    if (VariableParentesisAbre(expresion)) return 18;
    if (ParCierraParAbre(expresion)) return 19;
    if (OperadorPunto(expresion)) return 20;
    if (ParAbrePunto(expresion)) return 21;
    if (PuntoParAbre(expresion)) return 22;
    if (ParCierraPunto(expresion)) return 23;
    if (PuntoOperador(expresion)) return 24;
    if (PuntoParCierra(expresion)) return 25;

    return 0; //No se detectó error de sintaxis
  }

  //Muestra mensaje de error sintáctico
  public final String MensajeSintaxis(int CodigoError)
  {
    switch(CodigoError)
    {
      case 0:  return "No se detectó error sintáctico en las 25 pruebas que se hicieron.";
      case 1:  return "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3";
      case 2:  return "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7";
      case 3:  return "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)";
      case 4:  return "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))";
      case 5:  return "5. Que haya paréntesis vacío. Ejemplo: 2-()*3";
      case 6:  return "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4";
      case 7: return "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)";
      case 8: return "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)";
      case 9: return "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2";
      case 10: return "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1";
      case 11: return "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3";
      case 12: return "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1";
      case 13: return "13. Un número antes de una variable. Ejemplo: 3x+1";
      case 14: return "14. Un número después de una variable. Ejemplo: x21+4";
      case 15: return "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9";
      case 16: return "16. Función inexistente. Ejemplo: 5*alo(78)";
      case 17: return "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5";
      case 18: return "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)";
      case 19: return "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)";
```

```
      case 20: return "20. Después de operador sigue un punto. Ejemplo: -.3+7";
      case 21: return "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)";
      case 22: return "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)";
      case 23: return "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2";
      case 24: return "24. Punto seguido de operador. Ejemplo: 5.*9+1";
      default: return "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5";
    }
  }

  //Retira caracteres inválidos. Pone la expresión entre paréntesis.
  public final String TransformaExpresion(String expr)
  {
    if (expr == null) return "";
    String validos = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
    StringBuilder nuevaExpr = new StringBuilder();
    String expr2 = expr.toLowerCase();
    nuevaExpr.append('(');
    for(int pos = 0; pos < expr2.length(); pos++)
    {
      char letra = expr2.charAt(pos);
      for(int valida = 0; valida < validos.length(); valida++)
        if (letra == validos.charAt(valida))
        {
          nuevaExpr.append(letra);
          break;
        }
    }
    nuevaExpr.append(')');
    return nuevaExpr.toString();
  }

  //1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
  private static boolean DobleTripleOperadorSeguido(String expr)
  {
    for(int pos = 0; pos < expr.length() - 1; pos++)
    {
      char car1 = expr.charAt(pos);   //Extrae un carácter
      char car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

      //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
      if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
        if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^')
          return true;
    }
    for (int pos = 0; pos < expr.length() - 2; pos++)
    {
      char car1 = expr.charAt(pos);   //Extrae un carácter
      char car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter
      char car3 = expr.charAt(pos + 2); //Extrae el siguiente carácter

      //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
      if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
        if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^')
          if (car3 == '+' || car3 == '-' || car3 == '*' || car3 == '/' || car3 == '^')
            return true;
    }
    return false;  //No encontró doble/triple operador seguido
  }

  //2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
  private static boolean OperadorParentesisCierra(String expr)
  {
    for(int pos = 0; pos < expr.length() - 1; pos++)
    {
      char car1 = expr.charAt(pos);   //Extrae un carácter

      //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
      if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
        if (expr.charAt(pos + 1) == ')') return true;
    }
    return false; //No encontró operador seguido de un paréntesis que cierra
  }

  //3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
  private static boolean ParentesisAbreOperador(String expr)
  {
    for(int pos = 0; pos < expr.length() - 1; pos++)
    {
      char car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

      //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
      if (expr.charAt(pos) == '(')
        if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^') return true;
    }
    return false;  //No encontró paréntesis que abre seguido de un operador
  }

  //4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
  private static boolean ParentesisDesbalanceados(String expr)
  {
    int parabre = 0, parcierra = 0;
    for(int pos = 0; pos < expr.length(); pos++)
```

```
    {
      char car1 = expr.charAt(pos);
      if (car1 == '(') parabre++;
      if (car1 == ')') parcierra++;
    }
    return parabre != parcierra;
  }

  //5. Que haya paréntesis vacío o sólo un punto entre paréntesis. Ejemplo: 2-()*3
  private static boolean ParentesisVacio(String expr)
  {
    //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
    for(int pos = 0; pos < expr.length() - 1; pos++)
      if (expr.charAt(pos) == '(' && expr.charAt(pos + 1) == ')') return true;
    return false;
  }

  //6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
  private static boolean ParentesisBalanceIncorrecto(String expr)
  {
    int balance = 0;
    for(int pos = 0; pos < expr.length(); pos++)
    {
      char car1 = expr.charAt(pos);   //Extrae un carácter
      if (car1 == '(') balance++;
      if (car1 == ')') balance--;
      if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
    }
    return false;
  }

  //7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)
  private static boolean ParentesisCierraNumero(String expr)
  {
    for(int pos = 0; pos < expr.length() - 1; pos++)
    {
      char car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

      //Compara si el primer carácter es paréntesis que cierra y el siguiente es número
      if (expr.charAt(pos) == ')')
        if (car2 >= '0' && car2 <= '9') return true;
    }
    return false;
  }

  //8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
  private static boolean NumeroParentesisAbre(String expr)
  {
    for(int pos = 0; pos < expr.length() - 1; pos++)
    {
      char car1 = expr.charAt(pos);   //Extrae un carácter

      //Compara si el primer carácter es número y el siguiente es paréntesis que abre
      if (car1 >= '0' && car1 <= '9')
        if (expr.charAt(pos + 1) == '(') return true;
    }
    return false;
  }

  //9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
  private static boolean DoblePuntoNumero(String expr)
  {
    int totalpuntos = 0;
    for(int pos = 0; pos < expr.length(); pos++)
    {
      char car1 = expr.charAt(pos);   //Extrae un carácter
      if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
      if (car1 == '.') totalpuntos++;
      if (totalpuntos > 1) return true;
    }
    return false;
  }

  //10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
  private static boolean ParentesisCierraVariable(String expr)
  {
    for(int pos = 0; pos < expr.length() - 1; pos++)
      if (expr.charAt(pos) == ')') //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
          return true;
    return false;
  }

  //11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
  private static boolean VariableluegoPunto(String expr)
  {
    for(int pos = 0; pos < expr.length() - 1; pos++)
      if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
        if (expr.charAt(pos + 1) == '.') return true;
    return false;
  }
```

```
  //12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
  private static boolean PuntoluegoVariable(String expr)
  {
    for(int pos = 0; pos < expr.length() - 1; pos++)
      if (expr.charAt(pos) == '.')
        if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
          return true;
    return false;
  }

  //13. Un número antes de una variable. Ejemplo: 3x+1
  //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
  private static boolean NumeroAntesVariable(String expr)
  {
    for(int pos = 0; pos < expr.length() - 1; pos++)
      if (expr.charAt(pos) >= '0' && expr.charAt(pos) <= '9')
        if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
          return true;
    return false;
  }

  //14. Un número después de una variable. Ejemplo: x21+4
  private static boolean VariableDespuesNumero(String expr)
  {
    for(int pos = 0; pos < expr.length() - 1; pos++)
      if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
        if (expr.charAt(pos + 1) >= '0' && expr.charAt(pos + 1) <= '9')
          return true;
    return false;
  }

  //15. Chequea si hay 4 o más letras seguidas
  private static boolean Chequea4letras(String expr)
  {
    for(int pos = 0; pos < expr.length() - 3; pos++)
    {
      char car1 = expr.charAt(pos);
      char car2 = expr.charAt(pos + 1);
      char car3 = expr.charAt(pos + 2);
      char car4 = expr.charAt(pos + 3);

      if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <= 'z')
        return true;
    }
    return false;
  }

  //16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
  private boolean FuncionInvalida(String expr)
  {
    for(int pos = 0; pos < expr.length() - 2; pos++)
    {
      char car1 = expr.charAt(pos);
      char car2 = expr.charAt(pos + 1);
      char car3 = expr.charAt(pos + 2);

      //Si encuentra tres letras seguidas
      if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
      {
        if (pos >= expr.length() - 4) return true; //Hay un error porque no sigue paréntesis
        if (expr.charAt(pos + 3) != '(') return true; //Hay un error porque no hay paréntesis
        if (EsFuncionInvalida(car1, car2, car3)) return true;
      }
    }
    return false;
  }

  //Chequea si las tres letras enviadas son una función
  private static boolean EsFuncionInvalida(char car1, char car2, char car3)
  {
    String listafunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";
    for(int pos = 0; pos <= listafunciones.length() - 3; pos+=3)
    {
      char listfunc1 = listafunciones.charAt(pos);
      char listfunc2 = listafunciones.charAt(pos + 1);
      char listfunc3 = listafunciones.charAt(pos + 2);
      if (car1 == listfunc1 && car2 == listfunc2 && car3 == listfunc3) return false;
    }
    return true;
  }

  //17. Si detecta sólo dos letras seguidas es un error
  private static boolean VariableInvalida(String expr)
  {
    int cuentaletras = 0;
    for(int pos = 0; pos < expr.length(); pos++)
    {
      if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
        cuentaletras++;
      else
      {
```

```
        if (cuentaletras == 2) return true;
        cuentaletras = 0;
      }
    }
    return cuentaletras == 2;
}

//18. Antes de paréntesis que abre hay una letra
private static boolean VariableParentesisAbre(String expr)
{
    int cuentaletras = 0;
    for(int pos = 0; pos < expr.length(); pos++)
    {
      char car1 = expr.charAt(pos);
      if (car1 >= 'a' && car1 <= 'z')
        cuentaletras++;
      else if (car1 == '(' && cuentaletras == 1)
        return true;
      else
        cuentaletras = 0;
    }
    return false;
}

//19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
private static boolean ParCierraParAbre(String expr)
{
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)==')' && expr.charAt(pos+1)=='(')
            return true;
    return false;
  }

  //20. Después de operador sigue un punto. Ejemplo: -.3+7
  private static boolean OperadorPunto(String expr)
  {
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)=='+' || expr.charAt(pos)=='-' || expr.charAt(pos)=='*' || expr.charAt(pos)=='/' ||
expr.charAt(pos)=='^')
                if (expr.charAt(pos+1)=='.')
                  return true;
    return false;
  }

  //21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
  private static boolean ParAbrePunto(String expr)
  {
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)=='(' && expr.charAt(pos+1)=='.')
            return true;
    return false;
  }

  //22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
  private static boolean PuntoParAbre(String expr)
  {
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)=='.' && expr.charAt(pos+1)=='(')
            return true;
    return false;
  }

  //23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
  private static boolean ParCierraPunto(String expr)
  {
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)==')' && expr.charAt(pos+1)=='.')
            return true;
    return false;
  }

  //24. Punto seguido de operador. Ejemplo: 5.*9+1
  private static boolean PuntoOperador(String expr)
  {
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)=='.')
          if (expr.charAt(pos+1)=='+' || expr.charAt(pos+1)=='-' || expr.charAt(pos+1)=='*' || expr.charAt(pos+1)=='/' ||
expr.charAt(pos+1)=='^')
                  return true;
    return false;
  }

  //25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
  private static boolean PuntoParCierra(String expr)
  {
    for (int pos = 0; pos < expr.length()-1; pos++)
        if (expr.charAt(pos)=='.' && expr.charAt(pos+1)==')')
            return true;
    return false;
  }

  /* Convierte una expresión con el menos unario en una expresión válida para el evaluador de expresiones */
```

```
public final String ArreglaNegativos(String expresion)
{
  StringBuilder NuevaExpresion = new StringBuilder();
  StringBuilder NuevaExpresion2 = new StringBuilder();

  //Si detecta un operador y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  for (int pos=0; pos<expresion.length(); pos++)
  {
    char letra1 = expresion.charAt(pos);
    if (letra1=='+' || letra1=='-' || letra1=='*' || letra1=='/' || letra1=='^')
      if (expresion.charAt(pos+1)=='-')
      {
        NuevaExpresion.append(letra1).append("(0-1)#");
        pos++;
        continue;
      }
    NuevaExpresion.append(letra1);
  }

  //Si detecta un paréntesis que abre y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  for (int pos=0; pos<NuevaExpresion.length(); pos++)
  {
    char letra1 = NuevaExpresion.charAt(pos);
    if (letra1=='(')
      if (NuevaExpresion.charAt(pos+1)=='-')
      {
        NuevaExpresion2.append(letra1).append("(0-1)#");
        pos++;
        continue;
      }
    NuevaExpresion2.append(letra1);
  }

  return NuevaExpresion2.toString();
}

//Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
public final void Analizar(String expresion)
{
  PiezaSimple.clear();
  PiezaEjecuta.clear();
  Generar_Piezas_Simples(expresion);
  Generar_Piezas_Ejecucion();
}

//Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
private void Generar_Piezas_Simples(String expresion)
{
  int longExpresion = expresion.length();

  //Variables requeridas para armar un número
  double parteentera = 0;
  double partedecimal = 0;
  double divide = 1;
  boolean entero = true;
  boolean armanumero = false;

  for (int cont = 0; cont < longExpresion; cont++) //Va de letra en letra de la expresión
  {
    char letra = expresion.charAt(cont);
    if (letra == '.')  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
      entero = false;
    else if (letra >= '0' && letra <= '9')  //Si es un número, entonces lo va armando
    {
      armanumero = true;
      if (entero)
        parteentera = parteentera * 10 + letra - ASCIINUMERO; //La parte entera del número
      else
      {
        divide *= 10;
        partedecimal = partedecimal * 10 + letra - ASCIINUMERO; //La parte decimal del número
      }
    }
    else
    {
      if (armanumero) //Si tenía armado un número, entonces crea la pieza ESNUMERO
      {
        PiezaSimple.add(new Pieza_Simple(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0));
        parteentera = 0;
        partedecimal = 0;
        divide = 1;
        entero = true;
        armanumero = false;
      }

      if (letra == '+' || letra == '-' || letra == '*' || letra == '/' || letra == '^' || letra == '#') PiezaSimple.add(new
Pieza_Simple(ESOPERADOR, 0, letra, 0, 0));
      else if (letra == '(') PiezaSimple.add(new Pieza_Simple(ESPARABRE, 0, '0', 0, 0)); //¿Es paréntesis que abre?
      else if (letra == ')') PiezaSimple.add(new Pieza_Simple(ESPARCIERRA, 0, '0', 0, 0));//¿Es paréntesis que cierra?
      else if (letra >= 'a' && letra <= 'z') //¿Es variable o función?
      {
        /* Detecta si es una función porque tiene dos letras seguidas */
```

```
          if (cont < longExpresion - 1)
          {
            char letra2 = expresion.charAt(cont + 1); /* Chequea si el siguiente carácter es una letra, dado el caso es una
función */
            if (letra2 >= 'a' && letra2 <= 'z')
            {
              char letra3 = expresion.charAt(cont + 2);
              int funcionDetectada = 1;  /* Identifica la función */
              for (int funcion = 0; funcion <= TAMANOFUNCION; funcion += 3)
              {
                if (letra == listaFunciones.charAt(funcion)
                    && letra2 == listaFunciones.charAt(funcion + 1)
                    && letra3 == listaFunciones.charAt(funcion + 2))
                  break;
                funcionDetectada++;
              }
              PiezaSimple.add(new Pieza_Simple(ESFUNCION, funcionDetectada, '0', 0, 0));  //Adiciona función a la lista
              cont += 3; /* Mueve tres caracteres  sin(  [s][i][n][(] */
            }
            else /* Es una variable, no una función */
              PiezaSimple.add(new Pieza_Simple(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA));
          }
          else /* Es una variable, no una función */
            PiezaSimple.add(new Pieza_Simple(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA));
        }
      }
    }
    if (armanumero) PiezaSimple.add(new Pieza_Simple(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0));
  }

  //Toma las piezas simples y las convierte en piezas de ejecución de funciones
  //Acumula = función (operando(número/variable/acumula))
  private void Generar_Piezas_Ejecucion()
  {
    int cont = PiezaSimple.size()-1;
    Contador_Acumula = 0;
    do
    {
      if (PiezaSimple.get(cont).getTipo() == ESPARABRE || PiezaSimple.get(cont).getTipo() == ESFUNCION)
      {
        Generar_Piezas_Operador('#', '#', cont);  //Primero evalúa los menos unarios
        Generar_Piezas_Operador('^', '^', cont);  //Luego evalúa las potencias
        Generar_Piezas_Operador('*', '/', cont);  //Luego evalúa multiplicar y dividir
        Generar_Piezas_Operador('+', '-', cont);  //Finalmente evalúa sumar y restar

        //Crea pieza de ejecución
        PiezaEjecuta.add(new Pieza_Ejecuta(PiezaSimple.get(cont).getFuncion(),
            PiezaSimple.get(cont + 1).getTipo(), PiezaSimple.get(cont + 1).getNumero(),
            PiezaSimple.get(cont + 1).getVariable(), PiezaSimple.get(cont + 1).getAcumula(),
            '+', ESNUMERO, 0, 0, 0));

        //La pieza pasa a ser de tipo Acumulador
        PiezaSimple.get(cont + 1).setAcumula(Contador_Acumula++);

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

  //Toma las piezas simples y las convierte en piezas de ejecución
  //Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
  private void Generar_Piezas_Operador(char operA, char operB, int inicio)
  {
    int cont = inicio + 1;
    do
    {
      if (PiezaSimple.get(cont).getTipo() == ESOPERADOR && (PiezaSimple.get(cont).getOperador() == operA ||
PiezaSimple.get(cont).getOperador() == operB))
      {
        //Crea pieza de ejecución
        PiezaEjecuta.add(new Pieza_Ejecuta(0,
            PiezaSimple.get(cont - 1).getTipo(),
            PiezaSimple.get(cont - 1).getNumero(), PiezaSimple.get(cont - 1).getVariable(), PiezaSimple.get(cont -
1).getAcumula(),
            PiezaSimple.get(cont).getOperador(),
            PiezaSimple.get(cont + 1).getTipo(),
            PiezaSimple.get(cont + 1).getNumero(), PiezaSimple.get(cont + 1).getVariable(), PiezaSimple.get(cont +
1).getAcumula()));

        //Elimina la pieza del operador y la siguiente
        PiezaSimple.remove(cont);
        PiezaSimple.remove(cont);

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

        //Cambia la pieza anterior por pieza acumula
        PiezaSimple.get(cont).setAcumula(Contador_Acumula++);
      }
```

```
      cont++;
    } while (cont < PiezaSimple.size() && PiezaSimple.get(cont).getTipo() != ESPARCIERRA);
  }

  //Calcula la expresión convertida en piezas de ejecución
  public final double Calcular()
  {
    double valorA=0, valorB=0;

    for (Pieza_Ejecuta aPiezaEjecuta : PiezaEjecuta)
    {
      switch (aPiezaEjecuta.getTipoOperA())
      {
        case 5: valorA = aPiezaEjecuta.getNumeroA(); break; //¿Es un número?
        case 6: valorA = VariableAlgebra[aPiezaEjecuta.getVariableA()]; break;  //¿Es una variable?
        case 7: valorA = PiezaEjecuta.get(aPiezaEjecuta.getAcumulaA()).getValorPieza();  break; //¿Es una expresión anterior?
      }
      if (Double.isNaN(valorA) || Double.isInfinite(valorA)) return valorA;

      switch (aPiezaEjecuta.getFuncion())
      {
        case 0:
          switch (aPiezaEjecuta.getTipoOperB()) {
            case 5:  valorB = aPiezaEjecuta.getNumeroB();  break; //¿Es un número?
            case 6:  valorB = VariableAlgebra[aPiezaEjecuta.getVariableB()];  break;  //¿Es una variable?
            case 7:  valorB = PiezaEjecuta.get(aPiezaEjecuta.getAcumulaB()).getValorPieza();  break; //¿Es una expresión
anterior?
          }
          if (Double.isNaN(valorB) || Double.isInfinite(valorB)) return valorB;

          switch (aPiezaEjecuta.getOperador())
          {
            case '#': aPiezaEjecuta.setValorPieza(valorA * valorB);  break;
            case '+': aPiezaEjecuta.setValorPieza(valorA + valorB);  break;
            case '-': aPiezaEjecuta.setValorPieza(valorA - valorB);  break;
            case '*': aPiezaEjecuta.setValorPieza(valorA * valorB);  break;
            case '/': aPiezaEjecuta.setValorPieza(valorA / valorB);  break;
            case '^': aPiezaEjecuta.setValorPieza(Math.pow(valorA, valorB));  break;
          }
          break;
        case 1:
        case 2: aPiezaEjecuta.setValorPieza(Math.sin(valorA));  break;
        case 3: aPiezaEjecuta.setValorPieza(Math.cos(valorA));  break;
        case 4: aPiezaEjecuta.setValorPieza(Math.tan(valorA));  break;
        case 5: aPiezaEjecuta.setValorPieza(Math.abs(valorA));  break;
        case 6: aPiezaEjecuta.setValorPieza(Math.asin(valorA)); break;
        case 7: aPiezaEjecuta.setValorPieza(Math.acos(valorA)); break;
        case 8: aPiezaEjecuta.setValorPieza(Math.atan(valorA)); break;
        case 9: aPiezaEjecuta.setValorPieza(Math.log(valorA));  break;
        case 10:aPiezaEjecuta.setValorPieza(Math.ceil(valorA)); break;
        case 11:aPiezaEjecuta.setValorPieza(Math.exp(valorA));  break;
        case 12:aPiezaEjecuta.setValorPieza(Math.sqrt(valorA)); break;
        case 13:aPiezaEjecuta.setValorPieza(Math.pow(valorA, 0.333333333333));  break;
      }
    }
    return PiezaEjecuta.get(PiezaEjecuta.size() - 1).getValorPieza();
  }

  // Da valor a las variables que tendrá la expresión algebraica
  public final void ValorVariable(char variableAlgebra, double valor)
  {
    VariableAlgebra[variableAlgebra - ASCIILETRA] = valor;
  }
}
```

Anexo 2. Código completo en C#

Pieza_Simple.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EvaluadorCS
{
  class Pieza_Simple
  {
    private int tipo; //Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
    private int funcion; //Que función es seno/coseno/tangente/sqrt
    private char operador; // +, -, *, /, ^
    private double numero; //Número real de la expresión
    private int variableAlgebra; //Variable de la expresión
    private int acumula;  //Indice de la microexpresión

    public int getTipo() { return this.tipo; }
    public int getFuncion() { return this.funcion; }
    public char getOperador() { return this.operador; }
    public double getNumero() { return this.numero; }
    public int getVariable() { return this.variableAlgebra; }
    public int getAcumula() { return this.acumula; }
    public void setAcumula(int acumula) { this.tipo = 7; this.acumula = acumula; }

    public Pieza_Simple(int tipo, int funcion, char operador, double numero, int variable, int acumula)
    {
      this.tipo = tipo;
      this.funcion = funcion;
      this.operador = operador;
      this.variableAlgebra = variable;
      this.acumula = acumula;
      this.numero = numero;
    }
  }
}
```

Pieza_Ejecuta.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EvaluadorCS
{
  class Pieza_Ejecuta
  {
    private double valorPieza;  //Almacena el calculo de la operación. Es Acumula

    private int funcion;  // ¿Es una función? 0 no lo es. 1 es seno, 3 es coseno, ....

    private int tipo_operandoA; //Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
    private double numeroA;
    private int variableA;
    private int acumulaA;

    private char operador; // +, -, *, /, ^

    private int tipo_operandoB; //Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
    private double numeroB;
    private int variableB;
    private int acumulaB;

    public double getValorPieza() { return this.valorPieza; }
    public void setValorPieza(double valor) { this.valorPieza = valor; }
    public int getFuncion() { return this.funcion; }
    public int getTipoOperA() { return this.tipo_operandoA; }
    public double getNumeroA() { return this.numeroA; }
    public int getVariableA() { return this.variableA; }
    public int getAcumulaA() { return this.acumulaA; }
    public char getOperador() { return this.operador; }
    public int getTipoOperB() { return this.tipo_operandoB; }
    public double getNumeroB() { return this.numeroB; }
    public int getVariableB() { return this.variableB; }
    public int getAcumulaB() { return this.acumulaB; }

    public Pieza_Ejecuta(int funcion, int tipo_operandoA, double numeroA, int variableA, int acumulaA, char operador, int
tipo_operandoB, double numeroB, int variableB, int acumulaB)
    {
      this.valorPieza = 0;
```

```
      this.funcion = funcion;

      this.tipo_operandoA = tipo_operandoA;
      this.numeroA = numeroA;
      this.variableA = variableA;
      this.acumulaA = acumulaA;

      this.operador = operador;

      this.tipo_operandoB = tipo_operandoB;
      this.numeroB = numeroB;
      this.variableB = variableB;
      this.acumulaB = acumulaB;
    }
  }
}
```

Evaluar.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EvaluadorCS
{
  class Evaluar
  {
  /* Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 */
  private static int ASCIINUMERO = 48;

  /* Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 */
  private static int ASCIILETRA = 97;

  /* Las funciones que soporta este evaluador */
  private static int TAMANOFUNCION = 39;
  private static String listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

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

  //Listado de Piezas de análisis
  private List<Pieza_Simple> PiezaSimple = new List<Pieza_Simple>();

  //Listado de Piezas de ejecución
  private List<Pieza_Ejecuta> PiezaEjecuta = new List<Pieza_Ejecuta>();
  int Contador_Acumula = 0;

  //Almacena los valores de las 26 diferentes variables que puede tener la expresión algebraica
  private double[] VariableAlgebra = new double[26];

  //Valida la expresión algebraica
  public int EvaluaSintaxis(String expresion)
  {
    //Hace 25 pruebas de sintaxis
    if (DobleTripleOperadorSeguido(expresion)) return 1;
    if (OperadorParentesisCierra(expresion)) return 2;
    if (ParentesisAbreOperador(expresion)) return 3;
    if (ParentesisDesbalanceados(expresion)) return 4;
    if (ParentesisVacio(expresion)) return 5;
    if (ParentesisBalanceIncorrecto(expresion)) return 6;
    if (ParentesisCierraNumero(expresion)) return 7;
    if (NumeroParentesisAbre(expresion)) return 8;
    if (DoblePuntoNumero(expresion)) return 9;
    if (ParentesisCierraVariable(expresion)) return 10;
    if (VariableluegoPunto(expresion)) return 11;
    if (PuntoluegoVariable(expresion)) return 12;
    if (NumeroAntesVariable(expresion)) return 13;
    if (VariableDespuesNumero(expresion)) return 14;
    if (Chequea4letras(expresion)) return 15;
    if (FuncionInvalida(expresion)) return 16;
    if (VariableInvalida(expresion)) return 17;
    if (VariableParentesisAbre(expresion)) return 18;
    if (ParCierraParAbre(expresion)) return 19;
    if (OperadorPunto(expresion)) return 20;
    if (ParAbrePunto(expresion)) return 21;
    if (PuntoParAbre(expresion)) return 22;
    if (ParCierraPunto(expresion)) return 23;
    if (PuntoOperador(expresion)) return 24;
    if (PuntoParCierra(expresion)) return 25;
```

```
    return 0; //No se detectó error de sintaxis
  }

  //Muestra mensaje de error sintáctico
  public String MensajeSintaxis(int CodigoError)
  {
    switch(CodigoError)
    {
      case 0:  return "No se detectó error sintáctico en las 21 pruebas que se hicieron.";
      case 1:  return "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3";
      case 2:  return "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7";
      case 3:  return "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)";
      case 4:  return "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))";
      case 5:  return "5. Que haya paréntesis vacío. Ejemplo: 2-()*3";
      case 6:  return "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4";
      case 7: return "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)";
      case 8: return "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)";
      case 9: return "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2";
      case 10: return "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1";
      case 11: return "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3";
      case 12: return "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1";
      case 13: return "13. Un número antes de una variable. Ejemplo: 3x+1";
      case 14: return "14. Un número después de una variable. Ejemplo: x21+4";
      case 15: return "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9";
      case 16: return "16. Función inexistente. Ejemplo: 5*alo(78)";
      case 17: return "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5";
      case 18: return "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)";
      case 19: return "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)";
      case 20: return "20. Después de operador sigue un punto. Ejemplo: -.3+7";
      case 21: return "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)";
      case 22: return "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)";
      case 23: return "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2";
      case 24: return "24. Punto seguido de operador. Ejemplo: 5.*9+1";
      default: return "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5";
    }
  }

  //Retira caracteres inválidos. Pone la expresión entre paréntesis.
  public String TransformaExpresion(String expr)
  {
    if (expr == null) return "";
    String validos = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
    StringBuilder nuevaExpr = new StringBuilder();
    String expr2 = expr.ToLower();
    nuevaExpr.Append('(');
    for(int pos = 0; pos < expr2.Length; pos++)
    {
      char letra = expr2[pos];
      for(int valida = 0; valida < validos.Length; valida++)
        if (letra == validos[valida])
        {
          nuevaExpr.Append(letra);
          break;
        }
    }
    nuevaExpr.Append(')');
    return nuevaExpr.ToString();
  }

  //1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
  private static Boolean DobleTripleOperadorSeguido(String expr)
  {
    for(int pos = 0; pos < expr.Length - 1; pos++)
    {
      char car1 = expr[pos];   //Extrae un carácter
      char car2 = expr[pos + 1]; //Extrae el siguiente carácter

      //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
      if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
        if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^')
          return true;
    }

    for (int pos = 0; pos < expr.Length - 2; pos++)
    {
      char car1 = expr[pos];   //Extrae un carácter
      char car2 = expr[pos + 1]; //Extrae el siguiente carácter
      char car3 = expr[pos + 2]; //Extrae el siguiente carácter


      //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
      if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
        if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^')
          if (car3 == '+' || car3 == '-' || car3 == '*' || car3 == '/' || car3 == '^')
            return true;
    }

    return false;  //No encontró doble/triple operador seguido
  }

  //2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
  private static Boolean OperadorParentesisCierra(String expr)
```

```
{
  for(int pos = 0; pos < expr.Length - 1; pos++)
  {
    char car1 = expr[pos];   //Extrae un carácter

    //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
    if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
      if (expr[pos + 1] == ')') return true;
  }
  return false; //No encontró operador seguido de un paréntesis que cierra
}

//3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
private static Boolean ParentesisAbreOperador(String expr)
{
  for(int pos = 0; pos < expr.Length - 1; pos++)
  {
    char car2 = expr[pos + 1]; //Extrae el siguiente carácter

    //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
    if (expr[pos] == '(')
      if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^') return true;
  }
  return false;  //No encontró paréntesis que abre seguido de un operador
}

//4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
private static Boolean ParentesisDesbalanceados(String expr)
{
  int parabre = 0, parcierra = 0;
  for(int pos = 0; pos < expr.Length; pos++)
  {
    char car1 = expr[pos];
    if (car1 == '(') parabre++;
    if (car1 == ')') parcierra++;
  }
  return parabre != parcierra;
}

//5. Que haya paréntesis vacío. Ejemplo: 2-()*3
private static Boolean ParentesisVacio(String expr)
{
  //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
  for(int pos = 0; pos < expr.Length - 1; pos++)
    if (expr[pos] == '(' && expr[pos + 1] == ')') return true;
  return false;
}

//6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
private static Boolean ParentesisBalanceIncorrecto(String expr)
{
  int balance = 0;
  for(int pos = 0; pos < expr.Length; pos++)
  {
    char car1 = expr[pos];   //Extrae un carácter
    if (car1 == '(') balance++;
    if (car1 == ')') balance--;
    if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
  }
  return false;
}

//7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
private static Boolean ParentesisCierraNumero(String expr)
{
  for(int pos = 0; pos < expr.Length - 1; pos++)
  {
    char car2 = expr[pos + 1]; //Extrae el siguiente carácter

    //Compara si el primer carácter es paréntesis que cierra y el siguiente es número
    if (expr[pos] == ')')
      if (car2 >= '0' && car2 <= '9') return true;
  }
  return false;
}

//8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
private static Boolean NumeroParentesisAbre(String expr)
{
  for(int pos = 0; pos < expr.Length - 1; pos++)
  {
    char car1 = expr[pos];   //Extrae un carácter

    //Compara si el primer carácter es número y el siguiente es paréntesis que abre
    if (car1 >= '0' && car1 <= '9')
      if (expr[pos + 1] == '(') return true;
  }
  return false;
}

//9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
private static Boolean DoblePuntoNumero(String expr)
```

```
{
  int totalpuntos = 0;
  for(int pos = 0; pos < expr.Length; pos++)
  {
    char car1 = expr[pos];   //Extrae un carácter
    if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
    if (car1 == '.') totalpuntos++;
    if (totalpuntos > 1) return true;
  }
  return false;
}

//10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
private static Boolean ParentesisCierraVariable(String expr)
{
  for(int pos = 0; pos < expr.Length - 1; pos++)
    if (expr[pos] == ')') //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
      if (expr[pos + 1] >= 'a' && expr[pos + 1] <= 'z')
        return true;
  return false;
}

//11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
private static Boolean VariableluegoPunto(String expr)
{
  for(int pos = 0; pos < expr.Length - 1; pos++)
    if (expr[pos] >= 'a' && expr[pos] <= 'z')
      if (expr[pos + 1] == '.') return true;
  return false;
}

//12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
private static Boolean PuntoluegoVariable(String expr)
{
  for(int pos = 0; pos < expr.Length - 1; pos++)
    if (expr[pos] == '.')
      if (expr[pos + 1] >= 'a' && expr[pos + 1] <= 'z')
        return true;
  return false;
}

//13. Un número antes de una variable. Ejemplo: 3x+1
//Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en realidad 3*x+1
private static Boolean NumeroAntesVariable(String expr)
{
  for(int pos = 0; pos < expr.Length - 1; pos++)
    if (expr[pos] >= '0' && expr[pos] <= '9')
      if (expr[pos + 1] >= 'a' && expr[pos + 1] <= 'z')
        return true;
  return false;
}

//14. Un número después de una variable. Ejemplo: x21+4
private static Boolean VariableDespuesNumero(String expr)
{
  for(int pos = 0; pos < expr.Length - 1; pos++)
    if (expr[pos] >= 'a' && expr[pos] <= 'z')
      if (expr[pos + 1] >= '0' && expr[pos + 1] <= '9')
        return true;
  return false;
}

//15. Chequea si hay 4 o más letras seguidas
private static Boolean Chequea4letras(String expr)
{
  for(int pos = 0; pos < expr.Length - 3; pos++)
  {
    char car1 = expr[pos];
    char car2 = expr[pos + 1];
    char car3 = expr[pos + 2];
    char car4 = expr[pos + 3];

    if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <= 'z')
      return true;
  }
  return false;
}

//16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
private Boolean FuncionInvalida(String expr)
{
  for(int pos = 0; pos < expr.Length - 2; pos++)
  {
    char car1 = expr[pos];
    char car2 = expr[pos + 1];
    char car3 = expr[pos + 2];

    //Si encuentra tres letras seguidas
    if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
    {
      if (pos >= expr.Length - 4) return true; //Hay un error porque no sigue paréntesis
```

```
        if (expr[pos + 3] != '(') return true; //Hay un error porque no hay paréntesis
        if (FuncionInvalida(car1, car2, car3)) return true;
      }
    }
    return false;
  }

  //Chequea si las tres letras enviadas son una función
  private static Boolean FuncionInvalida(char car1, char car2, char car3)
  {
    String listafunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";
    for(int pos = 0; pos <= listafunciones.Length - 3; pos+=3)
    {
      char listfunc1 = listafunciones[pos];
      char listfunc2 = listafunciones[pos + 1];
      char listfunc3 = listafunciones[pos + 2];
      if (car1 == listfunc1 && car2 == listfunc2 && car3 == listfunc3) return false;
    }
    return true;
  }

  //17. Si detecta sólo dos letras seguidas es un error
  private static Boolean VariableInvalida(String expr)
  {
    int cuentaletras = 0;
    for(int pos = 0; pos < expr.Length; pos++)
    {
      if (expr[pos] >= 'a' && expr[pos] <= 'z')
        cuentaletras++;
      else
      {
        if (cuentaletras == 2) return true;
        cuentaletras = 0;
      }
    }
    return cuentaletras == 2;
  }

  //18. Antes de paréntesis que abre hay una letra
  private static Boolean VariableParentesisAbre(String expr)
  {
    int cuentaletras = 0;
    for(int pos = 0; pos < expr.Length; pos++)
    {
      char car1 = expr[pos];
      if (car1 >= 'a' && car1 <= 'z')
        cuentaletras++;
      else if (car1 == '(' && cuentaletras == 1)
        return true;
      else
        cuentaletras = 0;
    }
    return false;
  }


  //19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
  private static Boolean ParCierraParAbre(String expr)
  {
      for (int pos = 0; pos < expr.Length-1; pos++)
          if (expr[pos]==')' && expr[pos+1]=='(')
              return true;
      return false;
  }

  //20. Después de operador sigue un punto. Ejemplo: -.3+7
  private static Boolean OperadorPunto(String expr)
  {
      for (int pos = 0; pos < expr.Length-1; pos++)
          if (expr[pos]=='+' || expr[pos]=='-' || expr[pos]=='*' || expr[pos]=='/' || expr[pos]=='^')
                  if (expr[pos+1]=='.')
                    return true;
      return false;
  }

  //21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
  private static Boolean ParAbrePunto(String expr)
  {
      for (int pos = 0; pos < expr.Length-1; pos++)
          if (expr[pos]=='(' && expr[pos+1]=='.')
              return true;
      return false;
  }

  //22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
  private static Boolean PuntoParAbre(String expr)
  {
      for (int pos = 0; pos < expr.Length-1; pos++)
          if (expr[pos]=='.' && expr[pos+1]=='(')
              return true;
      return false;
  }
```

```
//23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
private static Boolean ParCierraPunto(String expr)
{
    for (int pos = 0; pos < expr.Length-1; pos++)
        if (expr[pos]==')' && expr[pos+1]=='.')
            return true;
    return false;
}

//24. Punto seguido de operador. Ejemplo: 5.*9+1
private static Boolean PuntoOperador(String expr)
{
    for (int pos = 0; pos < expr.Length-1; pos++)
        if (expr[pos]=='.')
          if (expr[pos+1]=='+' || expr[pos+1]=='-' || expr[pos+1]=='*' || expr[pos+1]=='/' || expr[pos+1]=='^')
                return true;
    return false;
}

//25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
private static Boolean PuntoParCierra(String expr)
{
    for (int pos = 0; pos < expr.Length-1; pos++)
        if (expr[pos]=='.' && expr[pos+1]==')')
            return true;
    return false;
}

/* Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones:
 * 1. Si encuentra un - al inicio le agrega un cero
 * 2. Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
 * 3. Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
 */
public String ArreglaNegativos(String expresion)
{
  StringBuilder NuevaExpresion = new StringBuilder();
  StringBuilder NuevaExpresion2 = new StringBuilder();

  //Si detecta un operador y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  for (int pos=0; pos<expresion.Length; pos++)
  {
    char letra1 = expresion[pos];
    if (letra1=='+' || letra1=='-' || letra1=='*' || letra1=='/' || letra1=='^')
      if (expresion[pos+1]=='-')
      {
        NuevaExpresion.Append(letra1).Append("(0-1)#");
        pos++;
        continue;
      }
    NuevaExpresion.Append(letra1);
  }

  //Si detecta un paréntesis que abre y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  for (int pos=0; pos<NuevaExpresion.Length; pos++)
  {
    char letra1 = NuevaExpresion[pos];
    if (letra1=='(')
      if (NuevaExpresion[pos+1]=='-')
      {
        NuevaExpresion2.Append(letra1).Append("(0-1)#");
        pos++;
        continue;
      }
    NuevaExpresion2.Append(letra1);
  }

  return NuevaExpresion2.ToString();
}


//Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
public void Analizar(String expresion)
{
  PiezaSimple.Clear();
  PiezaEjecuta.Clear();
  Generar_Piezas_Simples(expresion);
  Generar_Piezas_Ejecucion();
}

//Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
private void Generar_Piezas_Simples(String expresion)
{
  int longExpresion = expresion.Length;

  //Variables requeridas para armar un número
  double parteentera = 0;
  double partedecimal = 0;
  double divide = 1;
  bool entero = true;
  bool armanumero = false;
```

```
    for (int cont = 0; cont < longExpresion; cont++) //Va de letra en letra de la expresión
    {
      char letra = expresion[cont];
      if (letra == '.')  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
        entero = false;
      else if (letra >= '0' && letra <= '9')  //Si es un número, entonces lo va armando
      {
        armanumero = true;
        if (entero)
          parteentera = parteentera * 10 + letra - ASCIINUMERO; //La parte entera del número
        else
        {
          divide *= 10;
          partedecimal = partedecimal * 10 + letra - ASCIINUMERO; //La parte decimal del número
        }
      }
      else
      {
        if (armanumero) //Si tenía armado un número, entonces crea la pieza ESNUMERO
        {
          PiezaSimple.Add(new Pieza_Simple(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0, 0));
          parteentera = 0;
          partedecimal = 0;
          divide = 1;
          entero = true;
          armanumero = false;
        }

        if (letra == '+' || letra == '-' || letra == '*' || letra == '/' || letra == '^' || letra == '#') PiezaSimple.Add(new
Pieza_Simple(ESOPERADOR, 0, letra, 0, 0, 0));
        else if (letra == '(') PiezaSimple.Add(new Pieza_Simple(ESPARABRE, 0, '0', 0, 0, 0)); //¿Es paréntesis que abre?
        else if (letra == ')') PiezaSimple.Add(new Pieza_Simple(ESPARCIERRA, 0, '0', 0, 0, 0));//¿Es paréntesis que cierra?
        else if (letra >= 'a' && letra <= 'z') //¿Es variable o función?
        {
          /* Detecta si es una función porque tiene dos letras seguidas */
          if (cont < longExpresion - 1)
          {
            char letra2 = expresion[cont + 1]; /* Chequea si el siguiente carácter es una letra, dado el caso es una función */
            if (letra2 >= 'a' && letra2 <= 'z')
            {
              char letra3 = expresion[cont + 2];
              int funcionDetectada = 1;  /* Identifica la función */
              for (int funcion = 0; funcion <= TAMANOFUNCION; funcion += 3)
              {
                if (letra == listaFunciones[funcion]
                    && letra2 == listaFunciones[funcion + 1]
                    && letra3 == listaFunciones[funcion + 2])
                  break;
                funcionDetectada++;
              }
              PiezaSimple.Add(new Pieza_Simple(ESFUNCION, funcionDetectada, '0', 0, 0, 0));  //Adiciona función a la lista
              cont += 3; /* Mueve tres caracteres  sin(  [s][i][n][(] */
            }
            else /* Es una variable, no una función */
               PiezaSimple.Add(new Pieza_Simple(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA, 0));
          }
          else /* Es una variable, no una función */
            PiezaSimple.Add(new Pieza_Simple(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA, 0));
        }
      }
    }
    if (armanumero) PiezaSimple.Add(new Pieza_Simple(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0, 0));
  }

  //Toma las piezas simples y las convierte en piezas de ejecución de funciones
  //Acumula = función (operando(número/variable/acumula))
  private void Generar_Piezas_Ejecucion()
  {
    int cont = PiezaSimple.Count()-1;
    Contador_Acumula = 0;
    do
    {
      if (PiezaSimple[cont].getTipo() == ESPARABRE || PiezaSimple[cont].getTipo() == ESFUNCION)
      {
        Generar_Piezas_Operador('#', '#', cont);  //Primero evalúa los menos unarios
        Generar_Piezas_Operador('^', '^', cont);  //Luego evalúa las potencias
        Generar_Piezas_Operador('*', '/', cont);  //Luego evalúa multiplicar y dividir
        Generar_Piezas_Operador('+', '-', cont);  //Finalmente evalúa sumar y restar

        //Crea pieza de ejecución
        PiezaEjecuta.Add(new Pieza_Ejecuta(PiezaSimple[cont].getFuncion(),
           PiezaSimple[cont + 1].getTipo(), PiezaSimple[cont + 1].getNumero(), PiezaSimple[cont + 1].getVariable(),
PiezaSimple[cont + 1].getAcumula(),
           '+', ESNUMERO, 0, 0, 0));

        //La pieza pasa a ser de tipo Acumulador
        PiezaSimple[cont + 1].setAcumula(Contador_Acumula++);

        //Quita el paréntesis/función que abre y el que cierra, dejando el centro
        PiezaSimple.RemoveAt(cont);
        PiezaSimple.RemoveAt(cont + 1);
      }
```

```
      cont--;
    }while (cont>=0);
  }

  //Toma las piezas simples y las convierte en piezas de ejecución
  //Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
  private void Generar_Piezas_Operador(char operA, char operB, int inicio)
  {
    int cont = inicio + 1;
    do
    {
      if (PiezaSimple[cont].getTipo() == ESOPERADOR && (PiezaSimple[cont].getOperador() == operA ||
PiezaSimple[cont].getOperador() == operB))
      {
        //Crea pieza de ejecución
        PiezaEjecuta.Add(new Pieza_Ejecuta(0,
           PiezaSimple[cont - 1].getTipo(),
           PiezaSimple[cont - 1].getNumero(), PiezaSimple[cont - 1].getVariable(), PiezaSimple[cont - 1].getAcumula(),
           PiezaSimple[cont].getOperador(),
           PiezaSimple[cont + 1].getTipo(),
           PiezaSimple[cont + 1].getNumero(), PiezaSimple[cont + 1].getVariable(), PiezaSimple[cont + 1].getAcumula()));

        //Elimina la pieza del operador y la siguiente
        PiezaSimple.RemoveAt(cont);
        PiezaSimple.RemoveAt(cont);

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

        //Cambia la pieza anterior por pieza acumula
        PiezaSimple[cont].setAcumula(Contador_Acumula++);
      }
      cont++;
    } while (cont < PiezaSimple.Count() && PiezaSimple[cont].getTipo() != ESPARCIERRA);
  }

  //Calcula la expresión convertida en piezas de ejecución
  public double Calcular()
  {
    double valorA=0, valorB=0;
    int totalPiezaEjecuta = PiezaEjecuta.Count();

    for (int cont = 0; cont < totalPiezaEjecuta; cont++)
    {
      switch (PiezaEjecuta[cont].getTipoOperA())
      {
        case 5: valorA = PiezaEjecuta[cont].getNumeroA(); break; //¿Es un número?
        case 6: valorA = VariableAlgebra[PiezaEjecuta[cont].getVariableA()]; break;  //¿Es una variable?
        case 7: valorA = PiezaEjecuta[PiezaEjecuta[cont].getAcumulaA()].getValorPieza(); break; //¿Es una expresión anterior?
      }
      if (Double.IsNaN(valorA) || Double.IsInfinity(valorA)) return valorA;

      switch (PiezaEjecuta[cont].getFuncion())
      {
        case 0:
          switch (PiezaEjecuta[cont].getTipoOperB())
          {
            case 5: valorB = PiezaEjecuta[cont].getNumeroB(); break; //¿Es un número?
            case 6: valorB = VariableAlgebra[PiezaEjecuta[cont].getVariableB()]; break;  //¿Es una variable?
            case 7: valorB = PiezaEjecuta[PiezaEjecuta[cont].getAcumulaB()].getValorPieza(); break; //¿Es una expresión
anterior?
          }
          if (Double.IsNaN(valorB) || Double.IsInfinity(valorB)) return valorB;

          switch (PiezaEjecuta[cont].getOperador())
          {
            case '#': PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
            case '+': PiezaEjecuta[cont].setValorPieza(valorA + valorB); break;
            case '-': PiezaEjecuta[cont].setValorPieza(valorA - valorB); break;
            case '*': PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
            case '/': PiezaEjecuta[cont].setValorPieza(valorA / valorB); break;
            case '^': PiezaEjecuta[cont].setValorPieza(Math.Pow(valorA, valorB)); break;
          }
        break;
        case 1:
        case 2: PiezaEjecuta[cont].setValorPieza(Math.Sin(valorA)); break;
        case 3: PiezaEjecuta[cont].setValorPieza(Math.Cos(valorA)); break;
        case 4: PiezaEjecuta[cont].setValorPieza(Math.Tan(valorA)); break;
        case 5: PiezaEjecuta[cont].setValorPieza(Math.Abs(valorA)); break;
        case 6: PiezaEjecuta[cont].setValorPieza(Math.Asin(valorA)); break;
        case 7: PiezaEjecuta[cont].setValorPieza(Math.Acos(valorA)); break;
        case 8: PiezaEjecuta[cont].setValorPieza(Math.Atan(valorA)); break;
        case 9: PiezaEjecuta[cont].setValorPieza(Math.Log(valorA)); break;
        case 10: PiezaEjecuta[cont].setValorPieza(Math.Ceiling(valorA)); break;
        case 11: PiezaEjecuta[cont].setValorPieza(Math.Exp(valorA)); break;
        case 12: PiezaEjecuta[cont].setValorPieza(Math.Sqrt(valorA)); break;
        case 13: PiezaEjecuta[cont].setValorPieza(Math.Pow(valorA, 0.333333333333)); break;
      }
    }
    return PiezaEjecuta[totalPiezaEjecuta - 1].getValorPieza();
  }
```

```
  // Da valor a las variables que tendrá la expresión algebraica
  public void ValorVariable(char variableAlg, double valor)
  {
     VariableAlgebra[variableAlg - ASCIILETRA] = valor;
  }
 }
}
```

Anexo 3. Código completo en Visual Basic .NET

Pieza_Simple.vb

```
Public Class Pieza_Simple
    'Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
    Private tipo As Integer

    'Que función es seno/coseno/tangente/sqrt
    Private funcion As Integer

    ' +, -, *, /, ^
    Private operador As Char

    'Número real de la expresión
    Private numero As Double

    'Variable de la expresión
    Private variableAlgebra As Integer

    'Indice de la microexpresión
    Private acumula As Integer

    Public Function getTipo() As Integer
        Return Me.tipo
    End Function
    Public Function getFuncion() As Integer
        Return Me.funcion
    End Function
    Public Function getOperador() As Char
        Return Me.operador
    End Function
    Public Function getNumero() As Double
        Return Me.numero
    End Function
    Public Function getVariable() As Integer
        Return Me.variableAlgebra
    End Function
    Public Function getAcumula() As Integer
        Return Me.acumula
    End Function
    Public Sub setAcumula(acumula As Integer)
        Me.tipo = 7
        Me.acumula = acumula
    End Sub

    Public Sub New(tipo As Integer, funcion As Integer, operador As Char, numero As Double, variable As Integer, acumula As
Integer)
        Me.tipo = tipo
        Me.funcion = funcion
        Me.operador = operador
        Me.variableAlgebra = variable
        Me.acumula = acumula
        Me.numero = numero
    End Sub
End Class
```

Pieza_Ejecuta.vb

```
Public Class Pieza_Ejecuta
    'Almacena el calculo de la operación. Es Acumula
    Private valorPieza As Double

    ' ¿Es una función? 0 no lo es. 1 es seno, 3 es coseno, ....
    Private funcion As Integer

    'Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
    Private tipo_operandoA As Integer
    Private numeroA As Double
    Private variableA As Integer
    Private acumulaA As Integer

    ' +, -, *, /, ^
    Private operador As Char

    'Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
    Private tipo_operandoB As Integer
    Private numeroB As Double
    Private variableB As Integer
    Private acumulaB As Integer

    Public Function getValorPieza() As Double
        Return Me.valorPieza
    End Function
    Public Sub setValorPieza(valor As Double)
        Me.valorPieza = valor
```

```
    End Sub
    Public Function getFuncion() As Integer
        Return Me.funcion
    End Function
    Public Function getTipoOperA() As Integer
        Return Me.tipo_operandoA
    End Function
    Public Function getNumeroA() As Double
        Return Me.numeroA
    End Function
    Public Function getVariableA() As Integer
        Return Me.variableA
    End Function
    Public Function getAcumulaA() As Integer
        Return Me.acumulaA
    End Function
    Public Function getOperador() As Char
        Return Me.operador
    End Function
    Public Function getTipoOperB() As Integer
        Return Me.tipo_operandoB
    End Function
    Public Function getNumeroB() As Double
        Return Me.numeroB
    End Function
    Public Function getVariableB() As Integer
        Return Me.variableB
    End Function
    Public Function getAcumulaB() As Integer
        Return Me.acumulaB
    End Function

    Public Sub New(funcion As Integer, tipo_operandoA As Integer, numeroA As Double, variableA As Integer, acumulaA As Integer,
operador As Char, _
        tipo_operandoB As Integer, numeroB As Double, variableB As Integer, acumulaB As Integer)
        Me.valorPieza = 0

        Me.funcion = funcion

        Me.tipo_operandoA = tipo_operandoA
        Me.numeroA = numeroA
        Me.variableA = variableA
        Me.acumulaA = acumulaA

        Me.operador = operador

        Me.tipo_operandoB = tipo_operandoB
        Me.numeroB = numeroB
        Me.variableB = variableB
        Me.acumulaB = acumulaB
    End Sub
End Class
```

Evaluar.vb

```
Public Class Evaluar
  ' Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7
  Private Shared ASCIINUMERO As Integer = 48

  ' Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1
  Private Shared ASCIILETRA As Integer = 97

  ' Las funciones que soporta este evaluador
  Private Shared TAMANOFUNCION As Integer = 39
  Private Shared listaFunciones As [String] = "sinsencostanabsasnacsatnlogceiexpsqrrcb"

  ' Constantes de los diferentes tipos de datos que tendrán las piezas
  Private Shared ESFUNCION As Integer = 1
  Private Shared ESPARABRE As Integer = 2
  Private Shared ESPARCIERRA As Integer = 3
  Private Shared ESOPERADOR As Integer = 4
  Private Shared ESNUMERO As Integer = 5
  Private Shared ESVARIABLE As Integer = 6

  'Listado de Piezas de análisis
  Private PiezaSimple As New List(Of Pieza_Simple)()

  'Listado de Piezas de ejecución
  Private PiezaEjecuta As New List(Of Pieza_Ejecuta)()
  Private Contador_Acumula As Integer = 0

  'Almacena los valores de las 26 diferentes variables que puede tener la expresión algebraica
  Private VariableAlgebra As Double() = New Double(25) {}

'Valida la expresión algebraica
Public Function EvaluaSintaxis(expresion As [String]) As Integer
  'Hace 25 pruebas de sintaxis
  If DobleTripleOperadorSeguido(expresion) Then
```

```
    Return 1
  End If
  If OperadorParentesisCierra(expresion) Then
    Return 2
  End If
  If ParentesisAbreOperador(expresion) Then
    Return 3
  End If
  If ParentesisDesbalanceados(expresion) Then
    Return 4
  End If
  If ParentesisVacio(expresion) Then
    Return 5
  End If
  If ParentesisBalanceIncorrecto(expresion) Then
    Return 6
  End If
  If ParentesisCierraNumero(expresion) Then
    Return 7
  End If
  If NumeroParentesisAbre(expresion) Then
    Return 8
  End If
  If DoblePuntoNumero(expresion) Then
    Return 9
  End If
  If ParentesisCierraVariable(expresion) Then
    Return 10
  End If
  If VariableluegoPunto(expresion) Then
    Return 11
  End If
  If PuntoluegoVariable(expresion) Then
    Return 12
  End If
  If NumeroAntesVariable(expresion) Then
    Return 13
  End If
  If VariableDespuesNumero(expresion) Then
    Return 14
  End If
  If Chequea4letras(expresion) Then
    Return 15
  End If
  If FuncionInvalida(expresion) Then
    Return 16
  End If
  If VariableInvalida(expresion) Then
    Return 17
  End If
  If VariableParentesisAbre(expresion) Then
    Return 18
  End If
  If ParCierraParAbre(expresion) Then
    Return 19
  End If
  If OperadorPunto(expresion) Then
    Return 20
  End If
  If ParAbrePunto(expresion) Then
    Return 21
  End If
  If PuntoParAbre(expresion) Then
    Return 22
  End If
  If ParCierraPunto(expresion) Then
    Return 23
  End If
  If PuntoOperador(expresion) Then
    Return 24
  End If
  If PuntoParCierra(expresion) Then
    Return 25
  End If

  Return 0
  'No se detectó error de sintaxis
End Function

'Muestra mensaje de error sintáctico
Public Function MensajeSintaxis(CodigoError As Integer) As [String]
  Select Case CodigoError
    Case 0
      Return "No se detectó error sintáctico en las 21 pruebas que se hicieron."
    Case 1
      Return "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3"
    Case 2
      Return "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7"
    Case 3
      Return "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)"
    Case 4
      Return "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))"
```

```
    Case 5
      Return "5. Que haya paréntesis vacío. Ejemplo: 2-()*3"
    Case 6
      Return "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4"
    Case 7
      Return "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)"
    Case 8
      Return "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)"
    Case 9
      Return "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2"
    Case 10
      Return "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1"
    Case 11
      Return "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3"
    Case 12
      Return "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1"
    Case 13
      Return "13. Un número antes de una variable. Ejemplo: 3x+1"
    Case 14
      Return "14. Un número después de una variable. Ejemplo: x21+4"
    Case 15
      Return "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9"
    Case 16
      Return "16. Función inexistente. Ejemplo: 5*alo(78)"
    Case 17
      Return "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5"
    Case 18
      Return "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)"
    Case 19
      Return "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)"
    Case 20
      Return "20. Después de operador sigue un punto. Ejemplo: -.3+7"
    Case 21
      Return "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)"
    Case 22
      Return "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)"
    Case 23
      Return "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2"
    Case 24
      Return "24. Punto seguido de operador. Ejemplo: 5.*9+1"
    Case Else
      Return "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5"
  End Select
End Function

'Retira caracteres inválidos. Pone la expresión entre paréntesis.
Public Function TransformaExpresion(expr As [String]) As [String]
  If expr Is Nothing Then
    Return ""
  End If
  Dim validos As [String] = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()"
  Dim nuevaExpr As New Text.StringBuilder()
  Dim expr2 As [String] = expr.ToLower()

  nuevaExpr.Append("(")
  For pos As Integer = 0 To expr2.Length - 1
    Dim letra As Char = expr2(pos)
    For valida As Integer = 0 To validos.Length - 1
        If letra = validos(valida) Then
            nuevaExpr.Append(letra)
            Exit For
        End If
    Next
  Next
  nuevaExpr.Append(")")
  Return nuevaExpr.ToString()
End Function

  '1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
  Private Shared Function DobleTripleOperadorSeguido(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      Dim car1 As Char = expr(pos)
      'Extrae un carácter
      Dim car2 As Char = expr(pos + 1)
      'Extrae el siguiente carácter
      'Compara si el carácter y el siguiente son operadores, dado el caso retorna true
      If car1 = "+" OrElse car1 = "-" OrElse car1 = "*" OrElse car1 = "/" OrElse car1 = "^" Then
        If car2 = "+" OrElse car2 = "*" OrElse car2 = "/" OrElse car2 = "^" Then
          Return True
        End If
      End If
    Next

    For pos As Integer = 0 To expr.Length - 3
      Dim car1 As Char = expr(pos)
      'Extrae un carácter
      Dim car2 As Char = expr(pos + 1)
      'Extrae el siguiente carácter
      Dim car3 As Char = expr(pos + 2)
      'Extrae el siguiente carácter

      'Compara si el carácter y el siguiente son operadores, dado el caso retorna true
```

```
      If car1 = "+" OrElse car1 = "-" OrElse car1 = "*" OrElse car1 = "/" OrElse car1 = "^" Then
        If car2 = "+" OrElse car2 = "-" OrElse car2 = "*" OrElse car2 = "/" OrElse car2 = "^" Then
          If car3 = "+" OrElse car3 = "-" OrElse car3 = "*" OrElse car3 = "/" OrElse car3 = "^" Then
            Return True
          End If
        End If
      End If
    Next

    Return False
    'No encontró doble/triple operador seguido
  End Function

  '2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
  Private Shared Function OperadorParentesisCierra(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      Dim car1 As Char = expr(pos)
      'Extrae un carácter
      'Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
      If car1 = "+" OrElse car1 = "-" OrElse car1 = "*" OrElse car1 = "/" OrElse car1 = "^" Then
        If expr(pos + 1) = ")" Then
          Return True
        End If
      End If
    Next
    Return False
    'No encontró operador seguido de un paréntesis que cierra
  End Function

  '3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
  Private Shared Function ParentesisAbreOperador(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      Dim car2 As Char = expr(pos + 1)
      'Extrae el siguiente carácter
      'Compara si el primer carácter es paréntesis que abre y el siguiente es operador
      If expr(pos) = "(" Then
        If car2 = "+" OrElse car2 = "*" OrElse car2 = "/" OrElse car2 = "^" Then
          Return True
        End If
      End If
    Next
    Return False
    'No encontró paréntesis que abre seguido de un operador
  End Function

  '4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
  Private Shared Function ParentesisDesbalanceados(expr As [String]) As [Boolean]
    Dim parabre As Integer = 0, parcierra As Integer = 0
    For pos As Integer = 0 To expr.Length - 1
      Dim car1 As Char = expr(pos)
      If car1 = "(" Then
        parabre += 1
      End If
      If car1 = ")" Then
        parcierra += 1
      End If
    Next
    Return parabre <> parcierra
  End Function

  '5. Que haya paréntesis vacío. Ejemplo: 2-()*3
  Private Shared Function ParentesisVacio(expr As [String]) As [Boolean]
    'Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) = "(" AndAlso expr(pos + 1) = ")" Then
        Return True
      End If
    Next
    Return False
  End Function

  '6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
  Private Shared Function ParentesisBalanceIncorrecto(expr As [String]) As [Boolean]
    Dim balance As Integer = 0
    For pos As Integer = 0 To expr.Length - 1
      Dim car1 As Char = expr(pos)
      'Extrae un carácter
      If car1 = "(" Then
        balance += 1
      End If
      If car1 = ")" Then
        balance -= 1
      End If
      If balance < 0 Then
        Return True
        'Si cae por debajo de cero es que el balance es erróneo
      End If
    Next
    Return False
  End Function

  '7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
```

```
  Private Shared Function ParentesisCierraNumero(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      Dim car2 As Char = expr(pos + 1)
      'Extrae el siguiente carácter
      'Compara si el primer carácter es paréntesis que cierra y el siguiente es número
      If expr(pos) = ")" Then
        If car2 >= "0" AndAlso car2 <= "9" Then
          Return True
        End If
      End If
    Next
    Return False
  End Function

  '8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
  Private Shared Function NumeroParentesisAbre(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      Dim car1 As Char = expr(pos)
      'Extrae un carácter
      'Compara si el primer carácter es número y el siguiente es paréntesis que abre
      If car1 >= "0" AndAlso car1 <= "9" Then
        If expr(pos + 1) = "(" Then
          Return True
        End If
      End If
    Next
    Return False
  End Function

  '9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
  Private Shared Function DoblePuntoNumero(expr As [String]) As [Boolean]
    Dim totalpuntos As Integer = 0
    For pos As Integer = 0 To expr.Length - 1
      Dim car1 As Char = expr(pos)
      'Extrae un carácter
      If (car1 < "0" OrElse car1 > "9") AndAlso car1 <> "." Then
        totalpuntos = 0
      End If
      If car1 = "." Then
        totalpuntos += 1
      End If
      If totalpuntos > 1 Then
        Return True
      End If
    Next
    Return False
  End Function

  '10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
  Private Shared Function ParentesisCierraVariable(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) = ")" Then
        'Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
        If expr(pos + 1) >= "a" AndAlso expr(pos + 1) <= "z" Then
          Return True
        End If
      End If
    Next
    Return False
  End Function

  '11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
  Private Shared Function VariableluegoPunto(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) >= "a" AndAlso expr(pos) <= "z" Then
        If expr(pos + 1) = "." Then
          Return True
        End If
      End If
    Next
    Return False
  End Function

  '12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
  Private Shared Function PuntoluegoVariable(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) = "." Then
        If expr(pos + 1) >= "a" AndAlso expr(pos + 1) <= "z" Then
          Return True
        End If
      End If
    Next
    Return False
  End Function

  '13. Un número antes de una variable. Ejemplo: 3x+1
  'Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
  Private Shared Function NumeroAntesVariable(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) >= "0" AndAlso expr(pos) <= "9" Then
        If expr(pos + 1) >= "a" AndAlso expr(pos + 1) <= "z" Then
```

```
          Return True
        End If
      End If
    Next
    Return False
  End Function

  '14. Un número después de una variable. Ejemplo: x21+4
  Private Shared Function VariableDespuesNumero(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) >= "a" AndAlso expr(pos) <= "z" Then
        If expr(pos + 1) >= "0" AndAlso expr(pos + 1) <= "9" Then
          Return True
        End If
      End If
    Next
    Return False
  End Function

  '15. Chequea si hay 4 o más letras seguidas
  Private Shared Function Chequea4letras(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 4
      Dim car1 As Char = expr(pos)
      Dim car2 As Char = expr(pos + 1)
      Dim car3 As Char = expr(pos + 2)
      Dim car4 As Char = expr(pos + 3)

      If car1 >= "a" AndAlso car1 <= "z" AndAlso car2 >= "a" AndAlso car2 <= "z" AndAlso car3 >= "a" AndAlso car3 <= "z" AndAlso
car4 >= "a" AndAlso car4 <= "z" Then
        Return True
      End If
    Next
    Return False
  End Function

  '16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
  Private Function FuncionInvalida(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 3
      Dim car1 As Char = expr(pos)
      Dim car2 As Char = expr(pos + 1)
      Dim car3 As Char = expr(pos + 2)

      'Si encuentra tres letras seguidas
      If car1 >= "a" AndAlso car1 <= "z" AndAlso car2 >= "a" AndAlso car2 <= "z" AndAlso car3 >= "a" AndAlso car3 <= "z" Then
        If pos >= expr.Length - 4 Then
          Return True
        End If
        'Hay un error porque no sigue paréntesis
        If expr(pos + 3) <> "(" Then
          Return True
        End If
        'Hay un error porque no hay paréntesis
        If FuncionInvalida(car1, car2, car3) Then
          Return True
        End If
      End If
    Next
    Return False
  End Function

  'Chequea si las tres letras enviadas son una función
  Private Shared Function FuncionInvalida(car1 As Char, car2 As Char, car3 As Char) As [Boolean]
    Dim listafunciones As [String] = "sinsencostanabsasnacsatnlogceiexpsqrrcb"
    For pos As Integer = 0 To listafunciones.Length - 3 Step 3
      Dim listfunc1 As Char = listafunciones(pos)
      Dim listfunc2 As Char = listafunciones(pos + 1)
      Dim listfunc3 As Char = listafunciones(pos + 2)
      If car1 = listfunc1 AndAlso car2 = listfunc2 AndAlso car3 = listfunc3 Then
        Return False
      End If
    Next
    Return True
  End Function

  '17. Si detecta sólo dos letras seguidas es un error
  Private Shared Function VariableInvalida(expr As [String]) As [Boolean]
    Dim cuentaletras As Integer = 0
    For pos As Integer = 0 To expr.Length - 1
      If expr(pos) >= "a" AndAlso expr(pos) <= "z" Then
        cuentaletras += 1
      Else
        If cuentaletras = 2 Then
          Return True
        End If
        cuentaletras = 0
      End If
    Next
    Return cuentaletras = 2
  End Function

  '18. Antes de paréntesis que abre hay una letra
  Private Shared Function VariableParentesisAbre(expr As [String]) As [Boolean]
```

```
    Dim cuentaletras As Integer = 0
    For pos As Integer = 0 To expr.Length - 1
      Dim car1 As Char = expr(pos)
      If car1 >= "a" AndAlso car1 <= "z" Then
        cuentaletras += 1
      ElseIf car1 = "(" AndAlso cuentaletras = 1 Then
        Return True
      Else
        cuentaletras = 0
      End If
    Next
    Return False
  End Function


  '19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
  Private Shared Function ParCierraParAbre(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) = ")" AndAlso expr(pos + 1) = "(" Then
        Return True
      End If
    Next
    Return False
  End Function

  '20. Después de operador sigue un punto. Ejemplo: -.3+7
  Private Shared Function OperadorPunto(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) = "+" OrElse expr(pos) = "-" OrElse expr(pos) = "*" OrElse expr(pos) = "/" OrElse expr(pos) = "^" Then
        If expr(pos + 1) = "." Then
          Return True
        End If
      End If
    Next
    Return False
  End Function

  '21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
  Private Shared Function ParAbrePunto(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) = "(" AndAlso expr(pos + 1) = "." Then
        Return True
      End If
    Next
    Return False
  End Function

  '22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
  Private Shared Function PuntoParAbre(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) = "." AndAlso expr(pos + 1) = "(" Then
        Return True
      End If
    Next
    Return False
  End Function

  '23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
  Private Shared Function ParCierraPunto(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) = ")" AndAlso expr(pos + 1) = "." Then
        Return True
      End If
    Next
    Return False
  End Function

  '24. Punto seguido de operador. Ejemplo: 5.*9+1
  Private Shared Function PuntoOperador(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) = "." Then
        If expr(pos + 1) = "+" OrElse expr(pos + 1) = "-" OrElse expr(pos + 1) = "*" OrElse expr(pos + 1) = "/" OrElse expr(pos
+ 1) = "^" Then
          Return True
        End If
      End If
    Next
    Return False
  End Function

  '25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
  Private Shared Function PuntoParCierra(expr As [String]) As [Boolean]
    For pos As Integer = 0 To expr.Length - 2
      If expr(pos) = "." AndAlso expr(pos + 1) = ")" Then
        Return True
      End If
    Next
    Return False
  End Function

  ' Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones:
  '   * 1. Si encuentra un - al inicio le agrega un cero
```

```
'    * 2. Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
'    * 3. Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
'

Public Function ArreglaNegativos(expresion As [String]) As [String]
  Dim NuevaExpresion As New Text.StringBuilder()
  Dim NuevaExpresion2 As New Text.StringBuilder()

  'Si detecta un operador y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  For pos As Integer = 0 To expresion.Length - 1
    Dim letra1 As Char = expresion(pos)
    If letra1 = "+" OrElse letra1 = "-" OrElse letra1 = "*" OrElse letra1 = "/" OrElse letra1 = "^" Then
      If expresion(pos + 1) = "-" Then
        NuevaExpresion.Append(letra1).Append("(0-1)#")
        pos += 1
        Continue For
      End If
    End If
    NuevaExpresion.Append(letra1)
  Next

  'Si detecta un paréntesis que abre y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  For pos As Integer = 0 To NuevaExpresion.Length - 1
    Dim letra1 As Char = NuevaExpresion(pos)
    If letra1 = "(" Then
      If NuevaExpresion(pos + 1) = "-" Then
        NuevaExpresion2.Append(letra1).Append("(0-1)#")
        pos += 1
        Continue For
      End If
    End If
    NuevaExpresion2.Append(letra1)
  Next

  Return NuevaExpresion2.ToString()
End Function

'Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
Public Sub Analizar(expresion As [String])
  PiezaSimple.Clear()
  PiezaEjecuta.Clear()
  Generar_Piezas_Simples(expresion)
  Generar_Piezas_Ejecucion()
End Sub

'Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
Private Sub Generar_Piezas_Simples(expresion As [String])
  Dim longExpresion As Integer = expresion.Length

  'Variables requeridas para armar un número
  Dim parteentera As Double = 0
  Dim partedecimal As Double = 0
  Dim divide As Double = 1
  Dim entero As Boolean = True
  Dim armanumero As Boolean = False

  For cont As Integer = 0 To longExpresion - 1
    'Va de letra en letra de la expresión
    Dim letra As Char = expresion(cont)
    If letra = "." Then
      'Si letra es . entonces el resto de digitos leídos son la parte decimal del número
      entero = False
    ElseIf letra >= "0" AndAlso letra <= "9" Then
      'Si es un número, entonces lo va armando
      armanumero = True
      If entero Then
        parteentera = parteentera * 10 + Asc(letra) - ASCIINUMERO
      Else
        'La parte entera del número
        divide *= 10
        'La parte decimal del número
        partedecimal = partedecimal * 10 + Asc(letra) - ASCIINUMERO
      End If
    Else
      If armanumero Then
        'Si tenía armado un número, entonces crea la pieza ESNUMERO
        PiezaSimple.Add(New Pieza_Simple(ESNUMERO, 0, "0", parteentera + partedecimal / divide, 0, 0))
        parteentera = 0
        partedecimal = 0
        divide = 1
        entero = True
        armanumero = False
      End If

      If letra = "+" OrElse letra = "-" OrElse letra = "*" OrElse letra = "/" OrElse letra = "^" OrElse letra = "#" Then
        PiezaSimple.Add(New Pieza_Simple(ESOPERADOR, 0, letra, 0, 0, 0))
      ElseIf letra = "(" Then
        PiezaSimple.Add(New Pieza_Simple(ESPARABRE, 0, "0", 0, 0, 0))
        '¿Es paréntesis que abre?
      ElseIf letra = ")" Then
        PiezaSimple.Add(New Pieza_Simple(ESPARCIERRA, 0, "0", 0, 0, 0))
        '¿Es paréntesis que cierra?
```

```
        ElseIf letra >= "a" AndAlso letra <= "z" Then
          '¿Es variable o función?
          ' Detecta si es una función porque tiene dos letras seguidas

          If cont < longExpresion - 1 Then
            Dim letra2 As Char = expresion(cont + 1)
            ' Chequea si el siguiente carácter es una letra, dado el caso es una función
            If letra2 >= "a" AndAlso letra2 <= "z" Then
              Dim letra3 As Char = expresion(cont + 2)
              Dim funcionDetectada As Integer = 1
              ' Identifica la función
              For funcion As Integer = 0 To TAMANOFUNCION Step 3
                If letra = listaFunciones(funcion) AndAlso letra2 = listaFunciones(funcion + 1) AndAlso letra3 =
listaFunciones(funcion + 2) Then
                  Exit For
                End If
                funcionDetectada += 1
              Next
              PiezaSimple.Add(New Pieza_Simple(ESFUNCION, funcionDetectada, "0", 0, 0, 0))
              'Adiciona función a la lista
              ' Mueve tres caracteres  sin(  [s][i][n][(]
              cont += 3
            Else
              ' Es una variable, no una función
              PiezaSimple.Add(New Pieza_Simple(ESVARIABLE, 0, "0", 0, Asc(letra) - ASCIILETRA, 0))
            End If
          Else
            ' Es una variable, no una función
            PiezaSimple.Add(New Pieza_Simple(ESVARIABLE, 0, "0", 0, Asc(letra) - ASCIILETRA, 0))
          End If
        End If
      End If
    Next
    If armanumero Then
      PiezaSimple.Add(New Pieza_Simple(ESNUMERO, 0, "0", parteentera + partedecimal / divide, 0, 0))
    End If
  End Sub

  'Toma las piezas simples y las convierte en piezas de ejecución de funciones
  'Acumula = función (operando(número/variable/acumula))
  Private Sub Generar_Piezas_Ejecucion()
    Dim cont As Integer = PiezaSimple.Count() - 1
    Contador_Acumula = 0
    Do
      If PiezaSimple(cont).getTipo() = ESPARABRE OrElse PiezaSimple(cont).getTipo() = ESFUNCION Then
        Generar_Piezas_Operador("#", "#", cont)
        'Primero evalúa los menos unarios
        Generar_Piezas_Operador("^", "^", cont)
        'Luego evalúa las potencias
        Generar_Piezas_Operador("*", "/", cont)
        'Luego evalúa multiplicar y dividir
        Generar_Piezas_Operador("+", "-", cont)
        'Finalmente evalúa sumar y restar
        'Crea pieza de ejecución
        PiezaEjecuta.Add(New Pieza_Ejecuta(PiezaSimple(cont).getFuncion(), PiezaSimple(cont + 1).getTipo(), PiezaSimple(cont +
1).getNumero(), PiezaSimple(cont + 1).getVariable(), PiezaSimple(cont + 1).getAcumula(), "+", _
          ESNUMERO, 0, 0, 0))

        'La pieza pasa a ser de tipo Acumulador
        PiezaSimple(cont + 1).setAcumula(Contador_Acumula)
        Contador_Acumula = Contador_Acumula + 1

        'Quita el paréntesis/función que abre y el que cierra, dejando el centro
        PiezaSimple.RemoveAt(cont)
        PiezaSimple.RemoveAt(cont + 1)
      End If
      cont -= 1
    Loop While cont >= 0
  End Sub

  'Toma las piezas simples y las convierte en piezas de ejecución
  'Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
  Private Sub Generar_Piezas_Operador(operA As Char, operB As Char, inicio As Integer)
    Dim cont As Integer = inicio + 1
    Do
      If PiezaSimple(cont).getTipo() = ESOPERADOR AndAlso (PiezaSimple(cont).getOperador() = operA OrElse
PiezaSimple(cont).getOperador() = operB) Then
        'Crea pieza de ejecución
        PiezaEjecuta.Add(New Pieza_Ejecuta(0, PiezaSimple(cont - 1).getTipo(), PiezaSimple(cont - 1).getNumero(),
PiezaSimple(cont - 1).getVariable(), PiezaSimple(cont - 1).getAcumula(), PiezaSimple(cont).getOperador(), _
          PiezaSimple(cont + 1).getTipo(), PiezaSimple(cont + 1).getNumero(), PiezaSimple(cont + 1).getVariable(),
PiezaSimple(cont + 1).getAcumula()))

        'Elimina la pieza del operador y la siguiente
        PiezaSimple.RemoveAt(cont)
        PiezaSimple.RemoveAt(cont)

        'Retorna el contador en uno para tomar la siguiente operación
        cont -= 1

        'Cambia la pieza anterior por pieza acumula
        PiezaSimple(cont).setAcumula(Contador_Acumula)
```

```
        Contador_Acumula = Contador_Acumula + 1
      End If
      cont += 1
    Loop While cont < PiezaSimple.Count() AndAlso PiezaSimple(cont).getTipo() <> ESPARCIERRA
  End Sub

  'Calcula la expresión convertida en piezas de ejecución
  Public Function Calcular() As Double
    Dim valorA As Double = 0, valorB As Double = 0
    Dim totalPiezaEjecuta As Integer = PiezaEjecuta.Count()

    For cont As Integer = 0 To totalPiezaEjecuta - 1
      Select Case PiezaEjecuta(cont).getTipoOperA()
        Case 5
          valorA = PiezaEjecuta(cont).getNumeroA()
          Exit Select
          '¿Es un número?
        Case 6
          valorA = VariableAlgebra(PiezaEjecuta(cont).getVariableA())
          Exit Select
          '¿Es una variable?
        Case 7
          valorA = PiezaEjecuta(PiezaEjecuta(cont).getAcumulaA()).getValorPieza()
          Exit Select
          '¿Es una expresión anterior?
      End Select
      If [Double].IsNaN(valorA) OrElse [Double].IsInfinity(valorA) Then
        Return valorA
      End If

      Select Case PiezaEjecuta(cont).getFuncion()
        Case 0
          Select Case PiezaEjecuta(cont).getTipoOperB()
            Case 5
              valorB = PiezaEjecuta(cont).getNumeroB()
              Exit Select
              '¿Es un número?
            Case 6
              valorB = VariableAlgebra(PiezaEjecuta(cont).getVariableB())
              Exit Select
              '¿Es una variable?
            Case 7
              valorB = PiezaEjecuta(PiezaEjecuta(cont).getAcumulaB()).getValorPieza()
              Exit Select
              '¿Es una expresión anterior?
          End Select
          If [Double].IsNaN(valorB) OrElse [Double].IsInfinity(valorB) Then
            Return valorB
          End If

          Select Case PiezaEjecuta(cont).getOperador()
            Case "#"
              PiezaEjecuta(cont).setValorPieza(valorA * valorB)
              Exit Select
            Case "+"
              PiezaEjecuta(cont).setValorPieza(valorA + valorB)
              Exit Select
            Case "-"
              PiezaEjecuta(cont).setValorPieza(valorA - valorB)
              Exit Select
            Case "*"
              PiezaEjecuta(cont).setValorPieza(valorA * valorB)
              Exit Select
            Case "/"
              PiezaEjecuta(cont).setValorPieza(valorA / valorB)
              Exit Select
            Case "^"
              PiezaEjecuta(cont).setValorPieza(Math.Pow(valorA, valorB))
              Exit Select
          End Select
          Exit Select
        Case 1, 2
          PiezaEjecuta(cont).setValorPieza(Math.Sin(valorA))
          Exit Select
        Case 3
          PiezaEjecuta(cont).setValorPieza(Math.Cos(valorA))
          Exit Select
        Case 4
          PiezaEjecuta(cont).setValorPieza(Math.Tan(valorA))
          Exit Select
        Case 5
          PiezaEjecuta(cont).setValorPieza(Math.Abs(valorA))
          Exit Select
        Case 6
          PiezaEjecuta(cont).setValorPieza(Math.Asin(valorA))
          Exit Select
        Case 7
          PiezaEjecuta(cont).setValorPieza(Math.Acos(valorA))
          Exit Select
        Case 8
          PiezaEjecuta(cont).setValorPieza(Math.Atan(valorA))
          Exit Select
```

```
        Case 9
          PiezaEjecuta(cont).setValorPieza(Math.Log(valorA))
          Exit Select
        Case 10
          PiezaEjecuta(cont).setValorPieza(Math.Ceiling(valorA))
          Exit Select
        Case 11
          PiezaEjecuta(cont).setValorPieza(Math.Exp(valorA))
          Exit Select
        Case 12
          PiezaEjecuta(cont).setValorPieza(Math.Sqrt(valorA))
          Exit Select
        Case 13
          PiezaEjecuta(cont).setValorPieza(Math.Pow(valorA, 0.333333333333))
          Exit Select
      End Select
    Next
    Return PiezaEjecuta(totalPiezaEjecuta - 1).getValorPieza()
  End Function

  ' Da valor a las variables que tendrá la expresión algebraica
  Public Sub ValorVariable(variableAlg As Char, valor As Double)
    VariableAlgebra(Asc(variableAlg) - ASCIILETRA) = valor
  End Sub
End Class
```

Anexo 4. Código completo en C++

Pieza_Simple.h

```
class Pieza_Simple
{
private:
    int tipo; //Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
    int funcion; //Que función es seno/coseno/tangente/sqrt
    char operador; // +, -, *, /, ^
    double numero; //Número real de la expresión
    int variableAlgebra; //Variable de la expresión
    int acumula;  //Indice de la microexpresión

public:
    int getTipo();
    int getFuncion();
    char getOperador();
    double getNumero();
    int getVariable();
    int getAcumula();
    void setAcumula(int acumula);

    Pieza_Simple(int tipo, int funcion, char operador, double numero, int variable);
};
```

Pieza_Simple.cpp

```
#include "Pieza_Simple.h"

Pieza_Simple::Pieza_Simple(int tipo, int funcion, char operador, double numero, int variable)
{
  this->tipo = tipo;
  this->funcion = funcion;
  this->operador = operador;
  this->variableAlgebra = variable;
  this->acumula = acumula;
  this->numero = numero;
}

int Pieza_Simple::getTipo() {  return this->tipo;  }
int Pieza_Simple::getFuncion() { return this->funcion; }
char Pieza_Simple::getOperador() { return this->operador; }
double Pieza_Simple::getNumero() { return this->numero; }
int Pieza_Simple::getVariable() { return this->variableAlgebra; }
int Pieza_Simple::getAcumula() { return this->acumula; }
void Pieza_Simple::setAcumula(int acumula) { this->tipo = 7; this->acumula = acumula; }
```

Pieza_Ejecuta.h

```
class Pieza_Ejecuta
{
private:
      double valorPieza;

      int funcion;

      int tipo_operandoA;
      double numeroA;
      int variableA;
      int acumulaA;

      char operador;

      int tipo_operandoB;
      double numeroB;
      int variableB;
      int acumulaB;

public:
      double getValorPieza();
      void setValorPieza(double valor);
      int getFuncion();
      int getTipoOperA();
      double getNumeroA();
      int getVariableA();
      int getAcumulaA();
      char getOperador();
      int getTipoOperB();
      double getNumeroB();
      int getVariableB();
      int getAcumulaB();
```

```
    Pieza_Ejecuta(int funcion, int tipo_operandoA, double numeroA, int variableA, int acumulaA, char operador, int
tipo_operandoB, double numeroB, int variableB, int acumulaB);
};
```

Pieza_Ejecuta.cpp

```
#include "Pieza_Ejecuta.h"

double Pieza_Ejecuta::getValorPieza() { return this->valorPieza; }
void Pieza_Ejecuta::setValorPieza(double valor) { this->valorPieza = valor; }
int Pieza_Ejecuta::getFuncion() { return this->funcion; }
int Pieza_Ejecuta::getTipoOperA() { return this->tipo_operandoA; }
double Pieza_Ejecuta::getNumeroA() { return this->numeroA; }
int Pieza_Ejecuta::getVariableA() { return this->variableA; }
int Pieza_Ejecuta::getAcumulaA() { return this->acumulaA; }
char Pieza_Ejecuta::getOperador() { return this->operador; }
int Pieza_Ejecuta::getTipoOperB() { return this->tipo_operandoB; }
double Pieza_Ejecuta::getNumeroB() { return this->numeroB; }
int Pieza_Ejecuta::getVariableB() { return this->variableB; }
int Pieza_Ejecuta::getAcumulaB() { return this->acumulaB; }

Pieza_Ejecuta::Pieza_Ejecuta(int funcion, int tipo_operandoA, double numeroA, int variableA, int acumulaA, char operador, int
tipo_operandoB, double numeroB, int variableB, int acumulaB)
{
  this->valorPieza = 0;

  this->funcion = funcion;

  this->tipo_operandoA = tipo_operandoA;
  this->numeroA = numeroA;
  this->variableA = variableA;
  this->acumulaA = acumulaA;

  this->operador = operador;

  this->tipo_operandoB = tipo_operandoB;
  this->numeroB = numeroB;
  this->variableB = variableB;
  this->acumulaB = acumulaB;
}
```

Evaluar.h

```
#include "Pieza_Simple.h"
#include "Pieza_Ejecuta.h"
#include <vector>

class Evaluar
{
private:
    /* Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 */
    static const int ASCIINUMERO = 48;

    /* Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 */
    static const int ASCIILETRA = 97;

    /* Las funciones que soporta este evaluador */
    static const int TAMANOFUNCION = 39;
    static const char *listaFunciones;

    /* Constantes de los diferentes tipos de datos que tendrán las piezas */
    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 Piezas de análisis
    std::vector<Pieza_Simple> PiezaSimple;

    //Listado de Piezas de ejecución
    std::vector<Pieza_Ejecuta> PiezaEjecuta;
    int Contador_Acumula;

    //Almacena los valores de las 26 diferentes variables que puede tener la expresión algebraica
    double VariableAlgebra[26];

    char *expr;
    bool DobleTripleOperadorSeguido(char *);
    bool OperadorParentesisCierra(char *);
    bool ParentesisAbreOperador(char *);
    bool ParentesisDesbalanceados(char *);
    bool ParentesisVacio(char *);
    bool ParentesisBalanceIncorrecto(char *);
    bool ParentesisCierraNumero(char *);
```

```
    bool NumeroParentesisAbre(char *);
    bool DoblePuntoNumero(char *);
    bool ParentesisCierraVariable(char *);
    bool VariableluegoPunto(char *);
    bool PuntoluegoVariable(char *);
    bool NumeroAntesVariable(char *);
    bool VariableDespuesNumero(char *);
    bool Chequea4letras(char *);
    bool FuncionInvalida(char *);
    bool EsFuncionInvalida(char, char, char);
    bool VariableInvalida(char *);
    bool VariableParentesisAbre(char *);
    bool ParCierraParAbre(char *);
    bool OperadorPunto(char *);
    bool ParAbrePunto(char *);
    bool PuntoParAbre(char *);
    bool ParCierraPunto(char *);
    bool PuntoOperador(char *);
    bool PuntoParCierra(char *);

    //Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
    void Generar_Piezas_Simples(char *expresion);

    //Toma las piezas simples y las convierte en piezas de ejecución de funciones
    //Acumula = función (operando(número/variable/acumula))
    void Generar_Piezas_Ejecucion();

    //Toma las piezas simples y las convierte en piezas de ejecución
    //Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
    void Generar_Piezas_Operador(char operA, char operB, int inicio);



public:
    int EvaluaSintaxis(char *);
    void MensajeSintaxis(int CodigoError, char *);

    void TransformaExpresion(char *, char *);
    int ArreglaNegativos(char *, char *);

    //Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
    void Analizar(char *expresion);

    //Calcula la expresión convertida en piezas de ejecución
    double Calcular();

    // Da valor a las variables que tendrá la expresión algebraica
    void ValorVariable(char variableAlgebra, double valor);
};
```

Evaluar.cpp

```
#include "Evaluar.h"
#include <string.h>
#include <math.h>

const char *Evaluar::listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

//Valida la expresión algebraica
int Evaluar::EvaluaSintaxis(char *expresion)
{
    //Hace 25 pruebas de sintaxis
    if (DobleTripleOperadorSeguido(expresion)) return 1;
    if (OperadorParentesisCierra(expresion)) return 2;
    if (ParentesisAbreOperador(expresion)) return 3;
    if (ParentesisDesbalanceados(expresion)) return 4;
    if (ParentesisVacio(expresion)) return 5;
    if (ParentesisBalanceIncorrecto(expresion)) return 6;
    if (ParentesisCierraNumero(expresion)) return 7;
    if (NumeroParentesisAbre(expresion)) return 8;
    if (DoblePuntoNumero(expresion)) return 9;
    if (ParentesisCierraVariable(expresion)) return 10;
    if (VariableluegoPunto(expresion)) return 11;
    if (PuntoluegoVariable(expresion)) return 12;
    if (NumeroAntesVariable(expresion)) return 13;
    if (VariableDespuesNumero(expresion)) return 14;
    if (Chequea4letras(expresion)) return 15;
    if (FuncionInvalida(expresion)) return 16;
    if (VariableInvalida(expresion)) return 17;
    if (VariableParentesisAbre(expresion)) return 18;
    if (ParCierraParAbre(expresion)) return 19;
    if (OperadorPunto(expresion)) return 20;
    if (ParAbrePunto(expresion)) return 21;
    if (PuntoParAbre(expresion)) return 22;
    if (ParCierraPunto(expresion)) return 23;
    if (PuntoOperador(expresion)) return 24;
    if (PuntoParCierra(expresion)) return 25;

    //No se detectó error de sintaxis
```

```
    return 0;
}


//Muestra mensaje de error sintáctico
void Evaluar::MensajeSintaxis(int CodigoError, char *mensaje)
{
    switch(CodigoError)
    {
      case 0:  strcpy(mensaje, "No se detectó error sintáctico en las 25 pruebas que se hicieron."); break;
      case 1:  strcpy(mensaje, "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3"); break;
      case 2:  strcpy(mensaje, "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7"); break;
      case 3:  strcpy(mensaje, "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)"); break;
      case 4:  strcpy(mensaje, "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))"); break;
      case 5:  strcpy(mensaje, "5. Que haya paréntesis vacío. Ejemplo: 2-()*3"); break;
      case 6:  strcpy(mensaje, "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4"); break;
      case 7: strcpy(mensaje, "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)"); break;
      case 8: strcpy(mensaje, "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)"); break;
      case 9: strcpy(mensaje, "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2"); break;
      case 10: strcpy(mensaje, "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1"); break;
      case 11: strcpy(mensaje, "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3"); break;
      case 12: strcpy(mensaje, "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1"); break;
      case 13: strcpy(mensaje, "13. Un número antes de una variable. Ejemplo: 3x+1"); break;
      case 14: strcpy(mensaje, "14. Un número después de una variable. Ejemplo: x21+4"); break;
      case 15: strcpy(mensaje, "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9"); break;
      case 16: strcpy(mensaje, "16. Función inexistente. Ejemplo: 5*alo(78)"); break;
      case 17: strcpy(mensaje, "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5"); break;
      case 18: strcpy(mensaje, "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)"); break;
      case 19: strcpy(mensaje, "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)"); break;
      case 20: strcpy(mensaje, "20. Después de operador sigue un punto. Ejemplo: -.3+7"); break;
      case 21: strcpy(mensaje, "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)"); break;
      case 22: strcpy(mensaje, "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)"); break;
      case 23: strcpy(mensaje, "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2"); break;
      case 24: strcpy(mensaje, "24. Punto seguido de operador. Ejemplo: 5.*9+1"); break;
      default: strcpy(mensaje, "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5"); break;
    }
}

//Retira caracteres inválidos. Pone la expresión entre paréntesis.
void Evaluar::TransformaExpresion(char *nuevaExpr, char *expr)
{
  char validos[50];
  strcpy(validos, "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()");

  //Convierte a minúsculas
  for (int cont=0; cont<strlen(expr); cont++)
     if (expr[cont] >= 'A' && expr[cont] <= 'Z')
         expr[cont] = expr[cont] - 'A' + 'a';
  strcpy(nuevaExpr, "(");
  int posnuevaExpr = 1;
  for(int pos = 0; *(expr+pos); pos++)
  {
    char letra = expr[pos];
    for(int valida = 0; *(validos+valida); valida++)
      if (letra == validos[valida])
      {
        nuevaExpr[posnuevaExpr++] = letra;
        break;
      }
  }
  nuevaExpr[posnuevaExpr]=')';
  nuevaExpr[posnuevaExpr+1]='\0';
}


//1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
bool Evaluar::DobleTripleOperadorSeguido(char *expr)
{
    for (int pos=0; *(expr+pos+1); pos++)
    {
        char car1 = expr[pos];  //Extrae un carácter
        char car2 = expr[pos+1]; //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^')
                return true;
    }

    for (int pos=0; *(expr+pos+2); pos++)
    {
        char car1 = expr[pos];  //Extrae un carácter
        char car2 = expr[pos+1]; //Extrae el siguiente carácter
        char car3 = expr[pos+2]; //Extrae el siguiente carácter

        //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^')
                if (car3 == '+' || car3 == '-' || car3 == '*' || car3 == '/' || car3 == '^')
                    return true;
    }
    return false; //No encontró doble/triple operador seguido
```

```
}

//2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
bool Evaluar::OperadorParentesisCierra(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter

        //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
        if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
            if (expr[pos+1] == ')') return true;
    }
    return false; //No encontró operador seguido de un paréntesis que cierra
}

//3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
bool Evaluar::ParentesisAbreOperador(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
    {
        char car2 = expr[pos+1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
        if (expr[pos] == '(')
            if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^') return true;
    }
    return false; //No encontró paréntesis que abre seguido de un operador
}

//4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
bool Evaluar::ParentesisDesbalanceados(char *expr)
{
    int parabre = 0, parcierra = 0;
    for(int pos = 0; *(expr+pos); pos++)
    {
        char car1 = expr[pos];
        if (car1 == '(') parabre++;
        if (car1 == ')') parcierra++;
    }
    return (parabre != parcierra);
}

//5. Que haya paréntesis vacío. Ejemplo: 2-()*3
bool Evaluar::ParentesisVacio(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] == '(' && expr[pos+1] == ')') return true;
    return false;
}

//6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
bool Evaluar::ParentesisBalanceIncorrecto(char *expr)
{
    int balance = 0;
    for(int pos = 0; *(expr+pos); pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter
        if (car1 == '(') balance++;
        if (car1 == ')') balance--;
        if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
    }
    return false;
}

//7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
bool Evaluar::ParentesisCierraNumero(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
    {
        char car2 = expr[pos+1]; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
        if (expr[pos] == ')')
            if (car2 >= '0' && car2 <= '9') return true;
    }
    return false;
}

//8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
bool Evaluar::NumeroParentesisAbre(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter

        //Compara si el primer carácter es número y el siguiente es paréntesis que abre
        if (car1 >= '0' && car1 <= '9')
            if (expr[pos+1] == '(') return true;
    }
    return false;
}
```

```
//9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
bool Evaluar::DoblePuntoNumero(char *expr)
{
    int totalpuntos = 0;
    for(int pos = 0; *(expr+pos); pos++)
    {
        char car1 = expr[pos];   //Extrae un carácter
        if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
        if (car1 == '.') totalpuntos++;
        if (totalpuntos > 1) return true;
    }
    return false;
}

//10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
bool Evaluar::ParentesisCierraVariable(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] == ')') //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (expr[pos+1] >= 'a' && expr[pos+1] <= 'z') return true;
    return false;
}

//11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
bool Evaluar::VariableluegoPunto(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] >= 'a' && expr[pos] <= 'z')
            if (expr[pos+1] == '.') return true;
    return false;
}

//12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
bool Evaluar::PuntoluegoVariable(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] == '.')
            if (expr[pos+1] >= 'a' && expr[pos+1] <= 'z')
                return true;
    return false;
}

//13. Un número antes de una variable. Ejemplo: 3x+1
//Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
bool Evaluar::NumeroAntesVariable(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] >= '0' && expr[pos] <= '9')
            if (expr[pos+1] >= 'a' && expr[pos+1] <= 'z') return true;
    return false;
}

//14. Un número después de una variable. Ejemplo: x21+4
bool Evaluar::VariableDespuesNumero(char *expr)
{
    for(int pos = 0; *(expr+pos+1); pos++)
        if (expr[pos] >= 'a' && expr[pos] <= 'z')
            if (expr[pos+1] >= '0' && expr[pos+1] <= '9')
                return true;
    return false;
}

//15. Chequea si hay 4 o más letras seguidas
bool Evaluar::Chequea4letras(char *expr)
{
    for(int pos = 0; *(expr+pos+3); pos++)
    {
        char car1 = expr[pos];
        char car2 = expr[pos+1];
        char car3 = expr[pos+2];
        char car4 = expr[pos+3];

        if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <=
'z')
            return true;
    }
    return false;
}

//16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
bool Evaluar::FuncionInvalida(char *expr)
{
    for(int pos = 0; *(expr+pos+2); pos++)
    {
        char car1 = expr[pos];
        char car2 = expr[pos+1];
        char car3 = expr[pos+2];

        //Si encuentra tres letras seguidas
        if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
```

```
        {
            if (pos >= strlen(expr) - 4) return true; //Hay un error porque no sigue paréntesis
            if (*(expr+pos+3) != '(') return true; //Hay un error porque no hay paréntesis
            if (EsFuncionInvalida(car1, car2, car3)) return true;
        }
    }
    return false;
}

//Chequea si las tres letras enviadas son una función
bool Evaluar::EsFuncionInvalida(char car1, char car2, char car3)
{
    char *listafunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";
    for(int pos = 0; *(listafunciones+pos+2); pos+=3)
    {
        char listfunc1 = listafunciones[pos];
        char listfunc2 = listafunciones[pos + 1];
        char listfunc3 = listafunciones[pos + 2];
        if (car1 == listfunc1 && car2 == listfunc2 && car3 == listfunc3) return false;
    }
    return true;
}

//17. Si detecta sólo dos letras seguidas es un error
bool Evaluar::VariableInvalida(char *expr)
{
    int cuentaletras = 0;
    for(int pos = 0; *(expr+pos); pos++)
    {
      char car1 = expr[pos];
      if (car1 >= 'a' && car1 <= 'z')
        cuentaletras++;
      else
      {
        if (cuentaletras == 2) return true;
        cuentaletras = 0;
      }
    }
    return cuentaletras == 2;
}

//18. Antes de paréntesis que abre hay una letra
bool Evaluar::VariableParentesisAbre(char *expr)
{
    int cuentaletras = 0;
    for(int pos = 0; *(expr+pos); pos++)
    {
        char car1 = expr[pos];
        if (car1 >= 'a' && car1 <= 'z')
           cuentaletras++;
        else if (car1 == '(' && cuentaletras == 1)
           return true;
        else
           cuentaletras = 0;
    }
    return false;
}

//19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
bool Evaluar::ParCierraParAbre(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]==')' && expr[pos+1]=='(')
              return true;
      return false;
}

//20. Después de operador sigue un punto. Ejemplo: -.3+7
bool Evaluar::OperadorPunto(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]=='+' || expr[pos]=='-' || expr[pos]=='*' || expr[pos]=='/' || expr[pos]=='^')
                  if (expr[pos+1]=='.')
                    return true;
      return false;
}

//21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
bool Evaluar::ParAbrePunto(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]=='(' && expr[pos+1]=='.')
              return true;
      return false;
}

//22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
bool Evaluar::PuntoParAbre(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]=='.' && expr[pos+1]=='(')
              return true;
```

```
      return false;
}

//23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
bool Evaluar::ParCierraPunto(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]==')' && expr[pos+1]=='.')
              return true;
      return false;
}

//24. Punto seguido de operador. Ejemplo: 5.*9+1
bool Evaluar::PuntoOperador(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]=='.')
            if (expr[pos+1]=='+' || expr[pos+1]=='-' || expr[pos+1]=='*' || expr[pos+1]=='/' || expr[pos+1]=='^')
                  return true;
      return false;
}

//25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
bool Evaluar::PuntoParCierra(char *expr)
{
      for (int pos = 0; pos < *(expr+pos)-1; pos++)
          if (expr[pos]=='.' && expr[pos+1]==')')
              return true;
      return false;
}

/* Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones */
int Evaluar::ArreglaNegativos(char *NuevaExpresion2, char *expresion)
{
  char *NuevaExpresion = (char *) malloc((strlen(expresion)+100)*7);
  if (NuevaExpresion == NULL) return -1;
  NuevaExpresion[0] = '\0';
  int posNuevaExpr = 0;
  int posNuevaExpr2 = 0;

  //Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
  for (int pos=0; pos<strlen(expresion); pos++)
  {
    char letra1 = expresion[pos];
    if (letra1=='+' || letra1=='-' || letra1=='*' || letra1=='/' || letra1=='^')
      if (expresion[pos+1]=='-')
      {
        NuevaExpresion[posNuevaExpr++] = letra1;
        NuevaExpresion[posNuevaExpr++] = '(';
        NuevaExpresion[posNuevaExpr++] = '0';
        NuevaExpresion[posNuevaExpr++] = '-';
        NuevaExpresion[posNuevaExpr++] = '1';
        NuevaExpresion[posNuevaExpr++] = ')';
        NuevaExpresion[posNuevaExpr++] = '#';
        pos++;
        continue;
      }
      NuevaExpresion[posNuevaExpr++] =  letra1;
  }
  NuevaExpresion[posNuevaExpr++] = '\0';

  //Si detecta un paréntesis que abre y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  for (int pos=0; pos<strlen(NuevaExpresion); pos++)
  {
    char letra1 = NuevaExpresion[pos];
    if (letra1=='(')
      if (NuevaExpresion[pos+1]=='-')
      {
        NuevaExpresion2[posNuevaExpr2++] = letra1;
        NuevaExpresion2[posNuevaExpr2++] = '(';
        NuevaExpresion2[posNuevaExpr2++] = '0';
        NuevaExpresion2[posNuevaExpr2++] = '-';
        NuevaExpresion2[posNuevaExpr2++] = '1';
        NuevaExpresion2[posNuevaExpr2++] = ')';
        NuevaExpresion2[posNuevaExpr2++] = '#';
        pos++;
        continue;
      }
      NuevaExpresion2[posNuevaExpr2++] =  letra1;
  }
  NuevaExpresion2[posNuevaExpr2++] = '\0';

  free(NuevaExpresion);
  return 0;
}

//Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
void Evaluar::Analizar(char *expresion)
{
  PiezaSimple.clear();
  PiezaEjecuta.clear();
  Generar_Piezas_Simples(expresion);
```

```
  Generar_Piezas_Ejecucion();
}

//Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
void Evaluar::Generar_Piezas_Simples(char *expresion)
{
  int longExpresion = strlen(expresion);

  //Variables requeridas para armar un número
  double parteentera = 0;
  double partedecimal = 0;
  double divide = 1;
  bool entero = true;
  bool armanumero = false;

  for (int cont = 0; cont < longExpresion; cont++) //Va de letra en letra de la expresión
  {
    char letra = expresion[cont];
    if (letra == '.')  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
      entero = false;
    else if (letra >= '0' && letra <= '9')  //Si es un número, entonces lo va armando
    {
      armanumero = true;
      if (entero)
        parteentera = parteentera * 10 + letra - ASCIINUMERO; //La parte entera del número
      else
      {
        divide *= 10;
        partedecimal = partedecimal * 10 + letra - ASCIINUMERO; //La parte decimal del número
      }
    }
    else
    {
      if (armanumero) //Si tenía armado un número, entonces crea la pieza ESNUMERO
      {
        Pieza_Simple objeto(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0);
        PiezaSimple.push_back(objeto);
        parteentera = 0;
        partedecimal = 0;
        divide = 1;
        entero = true;
        armanumero = false;
      }

      if (letra == '+' || letra == '-' || letra == '*' || letra == '/' || letra == '^' || letra == '#'){ Pieza_Simple
objeto(ESOPERADOR, 0, letra, 0, 0); PiezaSimple.push_back(objeto); }
      else if (letra == '('){ Pieza_Simple objeto(ESPARABRE, 0, '0', 0, 0); PiezaSimple.push_back(objeto); }//¿Es paréntesis que
abre?
      else if (letra == ')'){ Pieza_Simple objeto(ESPARCIERRA, 0, '0', 0, 0); PiezaSimple.push_back(objeto); }//¿Es paréntesis
que cierra?
      else if (letra >= 'a' && letra <= 'z') //¿Es variable o función?
      {
        /* Detecta si es una función porque tiene dos letras seguidas */
        if (cont < longExpresion - 1)
        {
          char letra2 = expresion[cont + 1]; /* Chequea si el siguiente carácter es una letra, dado el caso es una función */
          if (letra2 >= 'a' && letra2 <= 'z')
          {
            char letra3 = expresion[cont + 2];
            int funcionDetectada = 1;  /* Identifica la función */
            for (int funcion = 0; funcion <= TAMANOFUNCION; funcion += 3)
            {
              if (letra == listaFunciones[funcion]
                && letra2 == listaFunciones[funcion + 1]
                && letra3 == listaFunciones[funcion + 2])
                break;
              funcionDetectada++;
            }
            Pieza_Simple objeto(ESFUNCION, funcionDetectada, '0', 0, 0);
            PiezaSimple.push_back(objeto);  //Adiciona función a la lista
            cont += 3; /* Mueve tres caracteres  sin(  [s][i][n][(] */
          }
          else /* Es una variable, no una función */
          {
            Pieza_Simple objeto(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA);
            PiezaSimple.push_back(objeto);
          }
        }
        else /* Es una variable, no una función */
        {
          Pieza_Simple objeto(ESVARIABLE, 0, '0', 0, letra - ASCIILETRA);
          PiezaSimple.push_back(objeto);
        }
      }
    }
  }
  if (armanumero) { Pieza_Simple objeto(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0); PiezaSimple.push_back(objeto);  }
}


//Toma las piezas simples y las convierte en piezas de ejecución de funciones
//Acumula = función (operando(número/variable/acumula))
```

```
void Evaluar::Generar_Piezas_Ejecucion()
{
  int cont = PiezaSimple.size()-1;
  Contador_Acumula = 0;
  do
  {
    if (PiezaSimple[cont].getTipo() == ESPARABRE || PiezaSimple[cont].getTipo() == ESFUNCION)
    {
      Generar_Piezas_Operador('#', '#', cont);  //Primero evalúa los menos unarios
      Generar_Piezas_Operador('^', '^', cont);  //Luego las potencias
      Generar_Piezas_Operador('*', '/', cont);  //Luego evalúa multiplicar y dividir
      Generar_Piezas_Operador('+', '-', cont);  //Finalmente evalúa sumar y restar

      //Crea pieza de ejecución
      Pieza_Ejecuta objeto(PiezaSimple[cont].getFuncion(),
                           PiezaSimple[cont + 1].getTipo(), PiezaSimple[cont + 1].getNumero(), PiezaSimple[cont +
1].getVariable(), PiezaSimple[cont + 1].getAcumula(),
                           '+', ESNUMERO, 0, 0, 0);
      PiezaEjecuta.push_back(objeto);

      //La pieza pasa a ser de tipo Acumulador
      PiezaSimple[cont + 1].setAcumula(Contador_Acumula++);

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

//Toma las piezas simples y las convierte en piezas de ejecución
//Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
void Evaluar::Generar_Piezas_Operador(char operA, char operB, int inicio)
{
  int cont = inicio + 1;
  do
  {
    if (PiezaSimple[cont].getTipo() == ESOPERADOR && (PiezaSimple[cont].getOperador() == operA ||
PiezaSimple[cont].getOperador() == operB))
    {
      //Crea pieza de ejecución
      Pieza_Ejecuta objeto(0,
                PiezaSimple[cont - 1].getTipo(),
                PiezaSimple[cont - 1].getNumero(), PiezaSimple[cont - 1].getVariable(), PiezaSimple[cont - 1].getAcumula(),
                PiezaSimple[cont].getOperador(),
                PiezaSimple[cont + 1].getTipo(),
                PiezaSimple[cont + 1].getNumero(), PiezaSimple[cont + 1].getVariable(), PiezaSimple[cont + 1].getAcumula());
      PiezaEjecuta.push_back(objeto);

      //Elimina la pieza del operador y la siguiente
      PiezaSimple.erase(PiezaSimple.begin() + cont);
      PiezaSimple.erase(PiezaSimple.begin() + cont);

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

      //Cambia la pieza anterior por pieza acumula
      PiezaSimple[cont].setAcumula(Contador_Acumula++);
    }
    cont++;
  } while (cont < PiezaSimple.size() && PiezaSimple[cont].getTipo() != ESPARCIERRA);
}

//Calcula la expresión convertida en piezas de ejecución
double Evaluar::Calcular()
{
  double valorA=0, valorB=0;
  int totalPiezaEjecuta = PiezaEjecuta.size();

  for (int cont = 0; cont < totalPiezaEjecuta; cont++)
  {
    switch (PiezaEjecuta[cont].getTipoOperA())
    {
      case 5: valorA = PiezaEjecuta[cont].getNumeroA(); break; //¿Es un número?
      case 6: valorA = VariableAlgebra[PiezaEjecuta[cont].getVariableA()]; break;  //¿Es una variable?
      case 7: valorA = PiezaEjecuta[PiezaEjecuta[cont].getAcumulaA()].getValorPieza(); break; //¿Es una expresión anterior?
    }
    if (_isnan(valorA) || !_finite(valorA)) return valorA;

    switch (PiezaEjecuta[cont].getFuncion())
    {
      case 0:
        switch (PiezaEjecuta[cont].getTipoOperB())
        {
          case 5: valorB = PiezaEjecuta[cont].getNumeroB(); break; //¿Es un número?
          case 6: valorB = VariableAlgebra[PiezaEjecuta[cont].getVariableB()]; break;  //¿Es una variable?
          case 7: valorB = PiezaEjecuta[PiezaEjecuta[cont].getAcumulaB()].getValorPieza(); break; //¿Es una expresión anterior?
        }
        if (_isnan(valorB) || !_finite(valorB)) return valorB;

        switch (PiezaEjecuta[cont].getOperador())
```

```
        {
          case '#': PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
          case '+': PiezaEjecuta[cont].setValorPieza(valorA + valorB); break;
          case '-': PiezaEjecuta[cont].setValorPieza(valorA - valorB); break;
          case '*': PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
          case '/': PiezaEjecuta[cont].setValorPieza(valorA / valorB); break;
          case '^': PiezaEjecuta[cont].setValorPieza(pow(valorA, valorB)); break;
        }
        break;
      case 1:
      case 2: PiezaEjecuta[cont].setValorPieza(sin(valorA)); break;
      case 3: PiezaEjecuta[cont].setValorPieza(cos(valorA)); break;
      case 4: PiezaEjecuta[cont].setValorPieza(tan(valorA)); break;
      case 5: PiezaEjecuta[cont].setValorPieza(abs(valorA)); break;
      case 6: PiezaEjecuta[cont].setValorPieza(asin(valorA)); break;
      case 7: PiezaEjecuta[cont].setValorPieza(acos(valorA)); break;
      case 8: PiezaEjecuta[cont].setValorPieza(atan(valorA)); break;
      case 9: PiezaEjecuta[cont].setValorPieza(log(valorA)); break;
      case 10: PiezaEjecuta[cont].setValorPieza(ceil(valorA)); break;
      case 11: PiezaEjecuta[cont].setValorPieza(exp(valorA)); break;
      case 12: PiezaEjecuta[cont].setValorPieza(sqrt(valorA)); break;
      case 13: PiezaEjecuta[cont].setValorPieza(pow(valorA, 0.333333333333)); break;
    }
  }
  return PiezaEjecuta[totalPiezaEjecuta - 1].getValorPieza();
}

// Da valor a las variables que tendrá la expresión algebraica
void Evaluar::ValorVariable(char variableAlg, double valor)
{
  VariableAlgebra[variableAlg - ASCIILETRA] = valor;
}
```

Anexo 5. Código completo en Object Pascal

Pieza_Simple.pas

```
unit Pieza_Simple;

interface
type
  TPieza_Simple = class
  private
     tipo: integer; //Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
     funcion: integer; //Que función es seno/coseno/tangente/sqrt
     operador: char; // +, -, *, /, ^
     numero: double; //Número real de la expresión
     variableAlgebra: integer; //Variable de la expresión
     acumula: integer;  //Indice de la microexpresión
   public
    Constructor Create(tipo: integer; funcion: integer; operador: char; numero: double; variable: integer; acumula: integer);
    function getTipo(): integer;
    function getFuncion(): integer;
    function getOperador(): char;
    function getNumero(): double;
    function getVariable(): integer;
    function getAcumula(): integer;
    procedure setAcumula(acumula: integer);
   end;

implementation

Constructor TPieza_Simple.Create(tipo: integer; funcion: integer; operador: char; numero: double; variable: integer; acumula:
integer);
begin
  self.tipo := tipo;
  self.funcion := funcion;
  self.operador := operador;
  self.variableAlgebra := variable;
  self.acumula := acumula;
  self.numero := numero;
end;

function TPieza_Simple.getTipo(): integer;
begin
  getTipo := tipo;
end;

function TPieza_Simple.getFuncion(): integer;
begin
  getFuncion := funcion;
end;

function TPieza_Simple.getOperador(): char;
begin
  getOperador := operador;
end;

function TPieza_Simple.getNumero(): double;
begin
  getNumero := numero;
end;

function TPieza_Simple.getVariable(): integer;
begin
  getVariable := variableAlgebra;
end;

function TPieza_Simple.getAcumula(): integer;
begin
  getAcumula := acumula;
end;

procedure TPieza_Simple.setAcumula(acumula: integer);
begin
  self.tipo := 7;
  self.acumula := acumula;
end;

end.
```

Pieza_Ejecuta.pas

```
unit Pieza_Ejecuta;

interface
type
  TPieza_Ejecuta = class
  private
        valorPieza: double;  //Almacena el calculo de la operación. Es Acumula
```

```
        funcion: integer;  // ¿Es una función? 0 no lo es. 1 es seno, 3 es coseno, ....

        tipo_operandoA: integer; //Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
        numeroA: double;
        variableA: integer;
        acumulaA: integer;

        operador: char; // +, -, *, /, ^

        tipo_operandoB: integer; //Que tipo de operando es: ¿Un número, una variable o una acumulación anterior?
        numeroB: double;
        variableB: integer;
        acumulaB: integer;
   public
        Constructor Create(funcion: integer; tipo_operandoA: integer; numeroA: double; variableA: integer; acumulaA: integer;
operador: char; tipo_operandoB: integer; numeroB: double; variableB: integer; acumulaB: integer);
        function getValorPieza(): double;
        procedure setValorPieza(valor: double);
        function getFuncion(): integer;
        function getTipoOperA(): integer;
        function getNumeroA(): double;
        function getVariableA(): integer;
        function getAcumulaA(): integer;
        function getOperador(): char;
        function getTipoOperB(): integer;
        function getNumeroB(): double;
        function getVariableB(): integer;
        function getAcumulaB(): integer;
   end;
implementation

Constructor TPieza_Ejecuta.Create(funcion: integer; tipo_operandoA: integer; numeroA: double; variableA: integer; acumulaA:
integer; operador: char; tipo_operandoB: integer; numeroB: double; variableB: integer; acumulaB: integer);
begin
    self.valorPieza := 0;

    self.funcion := funcion;

    self.tipo_operandoA := tipo_operandoA;
    self.numeroA := numeroA;
    self.variableA := variableA;
    self.acumulaA := acumulaA;

    self.operador := operador;

    self.tipo_operandoB := tipo_operandoB;
    self.numeroB := numeroB;
    self.variableB := variableB;
    self.acumulaB := acumulaB;
end;

function TPieza_Ejecuta.getValorPieza(): double;
begin
     getValorPieza := valorPieza;
end;

procedure TPieza_Ejecuta.setValorPieza(valor: double);
begin
     self.valorPieza := valor;
end;

function TPieza_Ejecuta.getFuncion(): integer;
begin
    getFuncion := funcion;
end;

function TPieza_Ejecuta.getTipoOperA(): integer;
begin
    getTipoOperA := tipo_operandoA;
end;

function TPieza_Ejecuta.getNumeroA(): double;
begin
    getNumeroA := numeroA;
end;

function TPieza_Ejecuta.getVariableA(): integer;
begin
  getVariableA := variableA;
end;

function TPieza_Ejecuta.getAcumulaA(): integer;
begin
  getAcumulaA := acumulaA;
end;

function TPieza_Ejecuta.getOperador(): char;
begin
  getOperador := operador;
end;
```

```
function TPieza_Ejecuta.getTipoOperB(): integer;
begin
    getTipoOperB := tipo_operandoB;
end;

function TPieza_Ejecuta.getNumeroB(): double;
begin
    getNumeroB := numeroB;
end;

function TPieza_Ejecuta.getVariableB(): integer;
begin
    getVariableB := variableB;
end;

function TPieza_Ejecuta.getAcumulaB(): integer;
begin
    getAcumulaB := acumulaB;
end;
end.
```

Evaluar.pas

```
unit Evaluar;

interface
uses
  //Requerido para el TobjectList
  Contnrs, SysUtils, Math, Pieza_Simple, Pieza_Ejecuta;
type
  TEvaluar = class
  private
    { Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 }
    ASCIINUMERO: integer;

    { Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 }
    ASCIILETRA: integer;

    { Las funciones que soporta este evaluador }
    TAMANOFUNCION: integer;
    listaFunciones: string;

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

    //Listado de Piezas de análisis
    PiezaSimple: TobjectList;
    objPiezaSimple: TPieza_Simple;

    //Listado de Piezas de ejecución
    PiezaEjecuta: TobjectList;
    objPiezaEjecuta: TPieza_Ejecuta;

    Contador_Acumula: integer;

    //Almacena los valores de las 26 diferentes variables que puede tener la expresión algebraica
    VariableAlgebra: array[0..26] of double;

    function DobleTripleOperadorSeguido(expr: string): boolean;
    function OperadorParentesisCierra(expr: string): boolean;
    function ParentesisAbreOperador(expr: string): boolean;
    function ParentesisDesbalanceados(expr: string): boolean;
    function ParentesisVacio(expr: string): boolean;
    function ParentesisBalanceIncorrecto(expr: string): boolean;
    function ParentesisCierraNumero(expr: string): boolean;
    function NumeroParentesisAbre(expr: string): boolean;
    function DoblePuntoNumero(expr: string): boolean;
    function ParentesisCierraVariable(expr: string): boolean;
    function VariableluegoPunto(expr: string): boolean;
    function PuntoluegoVariable(expr: string): boolean;
    function NumeroAntesVariable(expr: string): boolean;
    function VariableDespuesNumero(expr: string): boolean;
    function Chequea4letras(expr: string): boolean;
    function FuncionInvalida(expr: string): boolean;
    function EsFuncionInvalida(car1: char; car2: char; car3: char): boolean;
    function VariableInvalida(expr: string): boolean;
    function VariableParentesisAbre(expr: string): boolean;
    function ParCierraParAbre(expr: string): boolean;
    function OperadorPunto(expr: string): boolean;
    function ParAbrePunto(expr: string): boolean;
    function PuntoParAbre(expr: string): boolean;
    function ParCierraPunto(expr: string): boolean;
    function PuntoOperador(expr: string): boolean;
```

```
    function PuntoParCierra(expr: string): boolean;

    procedure Generar_Piezas_Simples(expresion: string);
    procedure Generar_Piezas_Ejecucion();
    procedure Generar_Piezas_Operador(operA: char; operB: char; inicio: integer);
  public
    Constructor Create();
    function EvaluaSintaxis(expresion: string): integer;
    function MensajeSintaxis(CodigoError: integer): string;
    function ArreglaNegativos(expresion: string): string;
    function TransformaExpresion(expr: string): string;
    procedure Analizar(expresion: string);
    function Calcular(): double;
    procedure ValorVariable(variableAlgebra: char; valor: double);
  end;

implementation

constructor TEvaluar.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.TAMANOFUNCION := 39;
   self.ASCIINUMERO := 48;
   self.ASCIILETRA := 97;
   self.listaFunciones := 'sinsencostanabsasnacsatnlogceiexpsqrrcb';
end;

//Valida la expresión algebraica
function TEvaluar.EvaluaSintaxis(expresion: string): integer;
begin
  //Hace 25 pruebas de sintaxis
  if DobleTripleOperadorSeguido(expresion)then begin Result :=  1; exit; end;
  if OperadorParentesisCierra(expresion)then begin Result :=  2; exit; end;
  if ParentesisAbreOperador(expresion)then begin Result :=  3; exit; end;
  if ParentesisDesbalanceados(expresion)then begin Result :=  4; exit; end;
  if ParentesisVacio(expresion)then begin Result :=  5; exit; end;
  if ParentesisBalanceIncorrecto(expresion)then begin Result :=  6; exit; end;
  if ParentesisCierraNumero(expresion)then begin Result :=  7; exit; end;
  if NumeroParentesisAbre(expresion)then begin Result :=  8; exit; end;
  if DoblePuntoNumero(expresion)then begin Result :=  9; exit; end;
  if ParentesisCierraVariable(expresion)then begin Result :=  10; exit; end;
  if VariableluegoPunto(expresion)then begin Result :=  11; exit; end;
  if PuntoluegoVariable(expresion)then begin Result :=  12; exit; end;
  if NumeroAntesVariable(expresion)then begin Result :=  13; exit; end;
  if VariableDespuesNumero(expresion)then begin Result :=  14; exit; end;
  if Chequea4letras(expresion)then begin Result :=  15; exit; end;
  if FuncionInvalida(expresion)then begin Result :=  16; exit; end;
  if VariableInvalida(expresion)then begin Result :=  17; exit; end;
  if VariableParentesisAbre(expresion)then begin Result :=  18; exit; end;
  if ParCierraParAbre(expresion)then begin Result :=  19; exit; end;
  if OperadorPunto(expresion)then begin Result :=  20; exit; end;
  if ParAbrePunto(expresion)then begin Result :=  21; exit; end;
  if PuntoParAbre(expresion)then begin Result :=  22; exit; end;
  if ParCierraPunto(expresion)then begin Result :=  23; exit; end;
  if PuntoOperador(expresion)then begin Result :=  24; exit; end;
  if PuntoParCierra(expresion)then begin Result :=  25; exit; end;
  Result := 0; //No se detectó error de sintaxis
end;

//Muestra mensaje de error sintáctico
function TEvaluar.MensajeSintaxis(CodigoError: integer): string;
begin
  case CodigoError of
      0: begin  Result := 'No se detectó error sintáctico en las 25 pruebas que se hicieron.'; exit; end;
      1: begin  Result := '1. Dos o más operadores estén seguidos. Ejemplo: begin 2++4, 5-*3'; exit; end;
      2: begin  Result := '2. Un operador seguido de un paréntesis que cierra. Ejemplo: begin 2-(4+)-7'; exit; end;
      3: begin  Result := '3. Un paréntesis que abre seguido de un operador. Ejemplo: begin 2-(*3)'; exit; end;
      4: begin  Result := '4. Que los paréntesis estén desbalanceados. Ejemplo: begin 3-(2*4))'; exit; end;
      5: begin  Result := '5. Que haya paréntesis vacío. Ejemplo: begin 2-()*3'; exit; end;
      6: begin  Result := '6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: begin 2+3)-2*(4'; exit; end;
      7: begin Result := '7. Un paréntesis que cierra y sigue un número. Ejemplo: begin (3-5)7-(1+2)'; exit; end;
      8: begin Result := '8. Un número seguido de un paréntesis que abre. Ejemplo: begin 7-2(5-6)'; exit; end;
      9: begin Result := '9. Doble punto en un número de tipo real. Ejemplo: begin 3-2..4+1  7-6.46.1+2'; exit; end;
     10: begin Result := '10. Un paréntesis que cierra seguido de una variable. Ejemplo: begin (12-4)y-1'; exit; end;
     11: begin Result := '11. Una variable seguida de un punto. Ejemplo: begin 4-z.1+3'; exit; end;
     12: begin Result := '12. Un punto seguido de una variable. Ejemplo: begin 7-2.p+1'; exit; end;
     13: begin Result := '13. Un número antes de una variable. Ejemplo: begin 3x+1'; exit; end;
     14: begin Result := '14. Un número después de una variable. Ejemplo: begin x21+4'; exit; end;
     15: begin Result := '15. Hay 4 o más letras seguidas. Ejemplo: begin 12+ramp+8.9'; exit; end;
     16: begin Result := '16. Función inexistente. Ejemplo: begin 5*alo(78)'; exit; end;
     17: begin Result := '17. Variable inválida (solo pueden tener una letra). Ejemplo: begin 5+tr-xc+5'; exit; end;
     18: begin Result := '18. Variable seguida de paréntesis que abre. Ejemplo: begin 5-a(7+3)'; exit; end;
     19: begin Result := '19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: begin (4-5)(2*x)'; exit;
end;
     20: begin Result := '20. Después de operador sigue un punto. Ejemplo: begin -.3+7'; exit; end;
     21: begin Result := '21. Después de paréntesis que abre sigue un punto. Ejemplo: begin 3*(.5+4)'; exit; end;
```

```
      22: begin Result := '22. Un punto seguido de un paréntesis que abre. Ejemplo: begin 7+3.(2+6)'; exit; end;
      23: begin Result := '23. Paréntesis cierra y sigue punto. Ejemplo: begin (4+5).7-2'; exit; end;
      24: begin Result := '24. Punto seguido de operador. Ejemplo: begin 5.*9+1'; exit; end;
      else begin Result := '25. Punto seguido de paréntesis que cierra. Ejemplo: begin (3+2.)*5'; exit; end;
  end;
end;

//Retira los espacios y caracteres inválidos.
function TEvaluar.TransformaExpresion(expr: string): string;
var
  validos, nuevaExpr, expr2: string;
  pos, valida: integer;
  letra: char;
begin
  validos := 'abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()';
  expr2 := lowercase(expr);
  nuevaExpr := '(';
  for pos := 1 to length(expr2) do
  begin
    letra := expr2[pos];
    for valida := 1 to length(validos) do
    begin
      if letra = validos[valida] then
      begin
          nuevaExpr := nuevaExpr + letra;
          break;
      end;
    end;
  end;
  nuevaExpr := nuevaExpr + ')';
  Result := nuevaExpr;
end;

//1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
function TEvaluar.DobleTripleOperadorSeguido(expr: string): boolean;
var
  pos: integer;
  car1, car2, car3: char;
begin
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];   //Extrae un carácter
    car2 := expr[pos + 1]; //Extrae el siguiente carácter

    //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
    if (car1 = '+') or (car1 = '-') or (car1 = '*') or (car1 = '/') or (car1 = '^') then
      if (car2 = '+') or (car2 = '*') or (car2 = '/') or (car2 = '^') then
      begin
        Result := true;
        exit;
      end;
  end;

  for pos := 1 to length(expr) - 1 do
  begin
    car1 := expr[pos];   //Extrae un carácter
    car2 := expr[pos + 1]; //Extrae el siguiente carácter
    car3 := expr[pos + 2]; //Extrae el siguiente carácter

    //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
    if (car1 = '+') or (car1 = '-') or (car1 = '*') or (car1 = '/') or (car1 = '^') then
      if (car2 = '+') or (car2 = '-') or (car2 = '*') or (car2 = '/') or (car2 = '^') then
        if (car3 = '+') or (car3 = '-') or (car3 = '*') or (car3 = '/') or (car3 = '^') then
        begin
          Result := true;
          exit;
        end;
  end;

  Result := false;  //No encontró doble/triple operador seguido
end;

//2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
function TEvaluar.OperadorParentesisCierra(expr: string): boolean;
var
  pos: integer;
  car1: char;
begin
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];   //Extrae un carácter

    //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
    if (car1 = '+') or (car1 = '-') or (car1 = '*') or (car1 = '/') or (car1 = '^') then
      if (expr[pos + 1] = ')') then
        begin
          Result := true;
          exit;
        end;
  end;
  Result := false; //No encontró operador seguido de un paréntesis que cierra
end;
```

```
//3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
function TEvaluar.ParentesisAbreOperador(expr: string): boolean;
var
  pos: integer;
  car2: char;
begin
  for pos := 1 to length(expr) do
  begin
    car2 := expr[pos + 1]; //Extrae el siguiente carácter

    //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
    if expr[pos] = '(' then
      if (car2 = '+') or (car2 = '*') or (car2 = '/') or (car2 = '^') then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;  //No encontró paréntesis que abre seguido de un operador
end;

//4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
function TEvaluar.ParentesisDesbalanceados(expr: string): boolean;
var
  parabre, parcierra, pos: integer;
  car1: char;
begin
  parabre := 0; parcierra := 0;
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];
    if car1 = '(' then Inc(parabre);
    if car1 = ')' then Inc(parcierra);
  end;
  if parabre <> parcierra then
    Result := true
  else
    Result := false;
end;

//5. Que haya paréntesis vacío. Ejemplo: 2-()*3
function TEvaluar.ParentesisVacio(expr: string): boolean;
var
  pos: integer;
begin
  //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] = '(') and (expr[pos + 1] = ')') then
    begin
      Result := true;
      Exit;
    end;
  end;
  Result := false;
end;

//6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
function TEvaluar.ParentesisBalanceIncorrecto(expr: string): boolean;
var
  balance, pos: integer;
  car1: char;
begin
  balance := 0;
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];   //Extrae un carácter
    if car1 = '(' then Inc(balance);
    if car1 = ')' then Dec(balance);
    if balance < 0 then begin Result := true; Exit; end;
  end;
  Result := false;
end;

//7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
function TEvaluar.ParentesisCierraNumero(expr: string): boolean;
var
  pos: integer;
  car2: char;
begin
  for pos := 1 to length(expr) do
  begin
    car2 := expr[pos + 1]; //Extrae el siguiente carácter

    //Compara si el primer carácter es paréntesis que cierra y el siguiente es número
    if (expr[pos] = ')') then
      if (car2 >= '0') and (car2 <= '9')  then
      begin
        Result := true;
        Exit;
      end;
```

```
  end;
  Result := false;
end;


//8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
function TEvaluar.NumeroParentesisAbre(expr: string): boolean;
var
  pos: integer;
  car1: char;
begin
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];   //Extrae un carácter

    //Compara si el primer carácter es número y el siguiente es paréntesis que abre
    if (car1 >= '0') and (car1 <= '9') then
      if expr[pos + 1] = '(' then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;

//9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
function TEvaluar.DoblePuntoNumero(expr: string): boolean;
var
  totalpuntos, pos: integer;
  car1: char;
begin
  totalpuntos := 0;
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];   //Extrae un carácter
    if ((car1 < '0') or (car1 > '9')) and (car1 <> '.') then totalpuntos := 0;
    if (car1 = '.') then Inc(totalpuntos);
    if (totalpuntos > 1) then
    begin
      Result := true;
      Exit;
    end;
  end;
  Result := false;
end;

//10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
function TEvaluar.ParentesisCierraVariable(expr: string): boolean;
var
  pos: integer;
begin
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] = ')') then //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
      if (expr[pos + 1] >= 'a') and (expr[pos + 1] <= 'z') then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;

//11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
function TEvaluar.VariableluegoPunto(expr: string): boolean;
var
  pos: integer;
begin
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] >= 'a') and (expr[pos] <= 'z') then
      if expr[pos + 1] = '.'then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;

//12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
function TEvaluar.PuntoluegoVariable(expr: string): boolean;
var
  pos: integer;
begin
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] = '.') then
      if (expr[pos + 1] >= 'a') and (expr[pos + 1] <= 'z') then
      begin
        Result := true;
```

```
        Exit;
      end;
  end;
  Result := false;
end;

//13. Un número antes de una variable. Ejemplo: 3x+1
//Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
function TEvaluar.NumeroAntesVariable(expr: string): boolean;
var
  pos: integer;
begin
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] >= '0') and (expr[pos] <= '9') then
      if (expr[pos + 1] >= 'a') and (expr[pos + 1] <= 'z') then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;

//14. Un número después de una variable. Ejemplo: x21+4
function TEvaluar.VariableDespuesNumero(expr: string): boolean;
var
  pos: integer;
begin
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] >= 'a') and (expr[pos] <= 'z') then
      if (expr[pos + 1] >= '0') and (expr[pos + 1] <= '9') then
      begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;

//15. Chequea si hay 4 o más variables seguidas
function TEvaluar.Chequea4letras(expr: string): boolean;
var
  pos: integer;
  car1, car2, car3, car4: char;
begin
  for pos := 1 to length(expr)-3 do
  begin
    car1 := expr[pos];
    car2 := expr[pos + 1];
    car3 := expr[pos + 2];
    car4 := expr[pos + 3];

    if (car1 >= 'a') and (car1 <= 'z') and (car2 >= 'a') and (car2 <= 'z') and (car3 >= 'a') and (car3 <= 'z') and (car4 >= 'a')
and (car4 <= 'z') then
    begin
        Result := true;
        Exit;
      end;
  end;
  Result := false;
end;

//16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
function TEvaluar.FuncionInvalida(expr: string): boolean;
var
  pos: integer;
  car1, car2, car3: char;
begin
  for pos := 1 to length(expr)-2 do
  begin
    car1 := expr[pos];
    car2 := expr[pos + 1];
    car3 := expr[pos + 2];

    //Si encuentra tres letras seguidas
    if (car1 >= 'a') and (car1 <= 'z') and (car2 >= 'a') and (car2 <= 'z') and (car3 >= 'a') and (car3 <= 'z') then
    begin
      if pos >= length(expr) - 4 then begin Result:= true; Exit; end;//Hay un error porque no sigue paréntesis
      if expr[pos + 3] <> '(' then begin Result:= true; Exit; end; //Hay un error porque no hay paréntesis
      if EsFuncionInvalida(car1, car2, car3) then begin Result:= true; Exit; end;
    end;
  end;
  Result := false;
end;

//Chequea si las tres letras enviadas son una función
function TEvaluar.EsFuncionInvalida(car1: char; car2: char; car3: char): boolean;
var
  pos, tamfunciones:integer;
```

```
  listfunc1, listfunc2, listfunc3: char;
  listafunciones: string;
begin
  listafunciones := 'sinsencostanabsasnacsatnlogceiexpsqrrcb';
  tamfunciones := length(listafunciones);
  pos := 1;
  while (pos <= tamfunciones - 2) do
  begin
    listfunc1 := listafunciones[pos];
    listfunc2 := listafunciones[pos + 1];
    listfunc3 := listafunciones[pos + 2];

    if (car1 = listfunc1) and (car2 = listfunc2) and (car3 = listfunc3) then
    begin
      Result := false;
      exit;
    end;

    pos := pos + 3;
  end;
  Result := true;
end;

//17. Si detecta sólo dos letras seguidas es un error
function TEvaluar.VariableInvalida(expr: string): boolean;
var
  cuentaletras, pos: integer;
begin
  cuentaletras := 0;
  for pos := 1 to length(expr) do
  begin
    if (expr[pos] >= 'a') and (expr[pos] <= 'z') then
      Inc(cuentaletras)
    else
    begin
      if cuentaletras = 2 then
      begin
        Result := true;
        exit;
      end;
      cuentaletras := 0;
    end;
  end;
  if cuentaletras = 2 then
    Result := true
  else
    Result := false;
end;

//18. Antes de paréntesis que abre hay una letra
function TEvaluar.VariableParentesisAbre(expr: string): boolean;
var
  pos, cuentaletras: integer;
  car1: char;
begin
  cuentaletras := 0;
  for pos := 1 to length(expr) do
  begin
    car1 := expr[pos];
    if (car1 >= 'a') and (car1 <= 'z') then
      Inc(cuentaletras)
    else if (car1 = '(') and (cuentaletras = 1) then
    begin
      Result := true;
      Exit;
    end
    else
      cuentaletras := 0;
  end;
  Result := false;
end;

//19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
function TEvaluar.ParCierraParAbre(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]=')') and (expr[pos+1]='(') then
          begin
              Result := true;
              Exit;
          end;
      end;
      Result := false;
end;

//20. Después de operador sigue un punto. Ejemplo: -.3+7
function TEvaluar.OperadorPunto(expr: string): boolean;
var
  pos: integer;
```

```
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]='+') or (expr[pos]='-') or (expr[pos]='*') or (expr[pos]='/') or (expr[pos]='^') then
                  if expr[pos+1]='.' then
                  begin
                    Result := true;
                    Exit;
                  end;
      end;
      Result := false;
end;

//21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
function TEvaluar.ParAbrePunto(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]='(') and (expr[pos+1]='.') then
          begin
              Result := true;
              Exit;
          end;
      end;
      Result := false;
end;

//22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
function TEvaluar.PuntoParAbre(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]='.') and (expr[pos+1]='(') then
          begin
              Result := true;
              Exit;
          end;
      end;
      Result := false;
end;

//23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
function TEvaluar.ParCierraPunto(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]=')') and (expr[pos+1]='.') then
          begin
              Result := true;
              Exit;
          end;
      end;
      Result := false;
end;

//24. Punto seguido de operador. Ejemplo: 5.*9+1
function TEvaluar.PuntoOperador(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]='.') then
            if (expr[pos+1]='+') or (expr[pos+1]='-') or (expr[pos+1]='*') or (expr[pos+1]='/') or (expr[pos+1]='^') then
            begin
                    Result := true;
                    Exit;
            end;
      end;
      Result := false;
end;

//25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
function TEvaluar.PuntoParCierra(expr: string): boolean;
var
  pos: integer;
begin
      for pos:=1 to length(expr) - 1 do
      begin
          if (expr[pos]='.') and (expr[pos+1]=')') then
          begin
              Result := true;
              Exit;
          end;
      end;
      Result := false;
```

```
end;


{ Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones:
 1. Si encuentra un - al inicio le agrega un cero
 2. Si detecta un paréntesis que abre y un - entonces pone el 0 adelante
 3. Si detecta un operador y luego un menos, entonces agrega un "(0-1)#" entre esos dos
}
function TEvaluar.ArreglaNegativos(expresion: string): string;
var
  letra1: char;
  NuevaExpresion, NuevaExpresion2: string;
  pos: integer;
begin
  NuevaExpresion := '';
  NuevaExpresion2 := '';

  //Si detecta un operador y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  pos := 1;
  while (pos <= length(expresion)) do
  begin
    letra1 := expresion[pos];
    if (letra1='+') or (letra1='-') or (letra1='*') or (letra1='/') or (letra1='^') then
      if expresion[pos+1]='-' then
      begin
        NuevaExpresion := NuevaExpresion + letra1 + '(0-1)#';
        Inc(pos); Inc(pos);
        continue;
      end;
    NuevaExpresion := NuevaExpresion + letra1;
    Inc(pos);
  end;

  //Si detecta un paréntesis que abre y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  pos := 1;
  while (pos <= length(NuevaExpresion)) do
  begin
    letra1 := NuevaExpresion[pos];
    if letra1='(' then
      if NuevaExpresion[pos+1]='-' then
      begin
        NuevaExpresion2 := NuevaExpresion2 + letra1 + '(0-1)#';
        Inc(pos); Inc(pos);
        continue;
      end;
    NuevaExpresion2 := NuevaExpresion2 + letra1;
    Inc(pos);
  end;

  Result := NuevaExpresion2;
end;



//Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
procedure TEvaluar.Analizar(expresion: string);
begin
  PiezaSimple.Free;
  PiezaEjecuta.Free;
  PiezaSimple := TObjectList.Create;
  PiezaEjecuta := TObjectList.Create;
  Generar_Piezas_Simples(expresion);
  Generar_Piezas_Ejecucion();
end;

//Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
procedure TEvaluar.Generar_Piezas_Simples(expresion: string);
var
  longExpresion, cont, funciondetectada, funcion: integer;
  parteentera, partedecimal, divide: double;
  entero, armanumero: boolean;
  letra, letra2, letra3: char;
begin
  longExpresion := length(expresion);

  //Variables requeridas para armar un número
  parteentera := 0;
  partedecimal := 0;
  divide := 1;
  entero := true;
  armanumero := false;

  cont := 1;
  while cont <= longExpresion do  //Va de letra en letra de la expresión
  begin
    letra := expresion[cont];
    if letra = '.' then  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
    begin
      entero := false
    end
    else if (letra >= '0') and (letra <= '9') then  //Si es un número, entonces lo va armando
    begin
```

```
      armanumero := true;
      if (entero) then
        parteentera := parteentera * 10 + ord(letra) - ASCIINUMERO //La parte entera del número
      else
      begin
        divide := divide * 10;
        partedecimal := partedecimal * 10 + ord(letra) - ASCIINUMERO; //La parte decimal del número
      end;
    end
    else
    begin
      if armanumero = true then //Si tenía armado un número, entonces crea la pieza ESNUMERO
      begin
        objPiezaSimple := TPieza_Simple.Create(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0, 0);
        PiezaSimple.Add(objPiezaSimple);
        parteentera := 0;
        partedecimal := 0;
        divide := 1;
        entero := true;
        armanumero := false;
      end;
      if (letra = '+') or (letra = '-') or (letra = '*') or (letra = '/') or (letra = '^') or (letra = '#') then
      begin
        objPiezaSimple := TPieza_Simple.Create(ESOPERADOR, 0, letra, 0, 0, 0);
        PiezaSimple.Add(objPiezaSimple);
      end
      else if (letra = '(') then
      begin
        objPiezaSimple := TPieza_Simple.Create(ESPARABRE, 0, '0', 0, 0, 0);
        PiezaSimple.Add(objPiezaSimple);
      end
      else if (letra = ')') then
      begin
        objPiezaSimple := TPieza_Simple.Create(ESPARCIERRA, 0, '0', 0, 0, 0);
        PiezaSimple.Add(objPiezaSimple);
      end
      else if (letra >= 'a') and (letra <= 'z') then //¿Es variable o función?
      begin
        // Detecta si es una función porque tiene dos letras seguidas
        if (cont < longExpresion - 1) then
        begin
          letra2 := expresion[cont + 1]; // Chequea si el siguiente carácter es una letra, dado el caso es una función
          if (letra2 >= 'a') and (letra2 <= 'z') then
          begin
            letra3 := expresion[cont + 2];
            funcionDetectada := 1;  // Identifica la función
            funcion := 1;
            while (funcion <= TAMANOFUNCION) do
            begin
              if (letra = listaFunciones[funcion]) and (letra2 = listaFunciones[funcion + 1]) and (letra3 =
listaFunciones[funcion + 2]) then break;
              Inc(funcionDetectada);
              funcion := funcion + 3;
            end;
            objPiezaSimple := TPieza_Simple.Create(ESFUNCION, funcionDetectada, '0', 0, 0, 0); //Adiciona función a la lista
            PiezaSimple.Add(objPiezaSimple);
            cont := cont + 3; // Mueve tres caracteres  sin(  [s][i][n][(]
          end
        else // Es una variable, no una función
        begin
          objPiezaSimple := TPieza_Simple.Create(ESVARIABLE, 0, '0', 0, ord(letra) - ASCIILETRA, 0);
          PiezaSimple.Add(objPiezaSimple);
        end
      end
      else // Es una variable, no una función
        begin
          objPiezaSimple := TPieza_Simple.Create(ESVARIABLE, 0, '0', 0, ord(letra) - ASCIILETRA, 0);
          PiezaSimple.Add(objPiezaSimple);
        end;
    end;
    Inc(cont);
  end;
  if armanumero then
  begin
    objPiezaSimple := TPieza_Simple.Create(ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0, 0);
    PiezaSimple.Add(objPiezaSimple);
  end
end;

//Toma las piezas simples y las convierte en piezas de ejecución de funciones
//Acumula = función (operando(número/variable/acumula))
procedure TEvaluar.Generar_Piezas_Ejecucion();
var
  cont: integer;
begin
  cont := PiezaSimple.Count-1;
  Contador_Acumula := 0;
  repeat
    if ((PiezaSimple[cont] as TPieza_Simple).getTipo() = ESPARABRE) or ((PiezaSimple[cont] as TPieza_Simple).getTipo() =
ESFUNCION) then
    begin
```

```
        Generar_Piezas_Operador('#', '#', cont);  //Primero evalúa los menos unarios
        Generar_Piezas_Operador('^', '^', cont);  //Luego evalúa las potencias
        Generar_Piezas_Operador('*', '/', cont);  //Luego evalúa multiplicar y dividir
        Generar_Piezas_Operador('+', '-', cont);  //Finalmente evalúa sumar y restar

        //Crea pieza de ejecución
        objPiezaEjecuta := TPieza_Ejecuta.Create((PiezaSimple[cont] as TPieza_Simple).getFuncion(),
                        (PiezaSimple[cont + 1] as TPieza_Simple).getTipo(), (PiezaSimple[cont + 1] as TPieza_Simple).getNumero(),
(PiezaSimple[cont + 1] as TPieza_Simple).getVariable(), (PiezaSimple[cont + 1] as TPieza_Simple).getAcumula(),
                        '+', ESNUMERO, 0, 0, 0);
        PiezaEjecuta.Add(objPiezaEjecuta);

        //La pieza pasa a ser de tipo Acumulador
        (PiezaSimple[cont + 1] as TPieza_Simple).setAcumula(Contador_Acumula);
        Inc(Contador_Acumula);

        //Quita el paréntesis/función que abre y el que cierra, dejando el centro
        PiezaSimple.Delete(cont);
        PiezaSimple.Delete(cont + 1);
    end;
    cont := cont - 1;
  until cont < 0;
end;

//Toma las piezas simples y las convierte en piezas de ejecución
//Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
procedure TEvaluar.Generar_Piezas_Operador(operA: char; operB: char; inicio: integer);
var
  cont: integer;
begin
  cont := inicio + 1;
  repeat
    if ((PiezaSimple[cont] as TPieza_Simple).getTipo() = ESOPERADOR) AND ( ((PiezaSimple[cont] as TPieza_Simple).getOperador() =
operA) OR ((PiezaSimple[cont] as TPieza_Simple).getOperador() = operB) ) then
    begin
      //Crea pieza de ejecución
      objPiezaEjecuta := TPieza_Ejecuta.Create(0,
                        (PiezaSimple[cont - 1] as TPieza_Simple).getTipo(),
                        (PiezaSimple[cont - 1] as TPieza_Simple).getNumero(), (PiezaSimple[cont - 1] as
TPieza_Simple).getVariable(), (PiezaSimple[cont - 1] as TPieza_Simple).getAcumula(),
                        (PiezaSimple[cont] as TPieza_Simple).getOperador(),
                        (PiezaSimple[cont + 1] as TPieza_Simple).getTipo(),
                        (PiezaSimple[cont + 1] as TPieza_Simple).getNumero(), (PiezaSimple[cont + 1] as
TPieza_Simple).getVariable(), (PiezaSimple[cont + 1] as TPieza_Simple).getAcumula());
      PiezaEjecuta.Add(objPiezaEjecuta);

      //Elimina la pieza del operador y la siguiente
      PiezaSimple.Delete(cont);
      PiezaSimple.Delete(cont);

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

      //Cambia la pieza anterior por pieza acumula
      (PiezaSimple[cont] as TPieza_Simple).setAcumula(Contador_Acumula);
      Inc(Contador_Acumula);
    end;
    Inc(cont);
  until (cont >= PiezaSimple.Count) or ((PiezaSimple[cont] as TPieza_Simple).getTipo() = ESPARCIERRA);
end;

//Calcula la expresión convertida en piezas de ejecución
function TEvaluar.Calcular(): double;
var
  valorA, valorB: double;
  totalPiezaEjecuta, cont: integer;
begin
  valorA := 0;
  valorB := 0;
  totalPiezaEjecuta := PiezaEjecuta.Count;

  try
    for cont := 0 to totalPiezaEjecuta-1 do
    begin
      case (PiezaEjecuta[cont] as TPieza_Ejecuta).getTipoOperA() of
        5: begin valorA := (PiezaEjecuta[cont] as TPieza_Ejecuta).getNumeroA(); end; //¿Es un número?
        6: begin valorA := VariableAlgebra[(PiezaEjecuta[cont] as TPieza_Ejecuta).getVariableA()]; end;  //¿Es una variable?
        7: begin valorA := (PiezaEjecuta[(PiezaEjecuta[cont] as TPieza_Ejecuta).getAcumulaA()] as
TPieza_Ejecuta).getValorPieza(); end; //¿Es una expresión anterior?
      end;

      case (PiezaEjecuta[cont] as TPieza_Ejecuta).getFuncion() of
        0:begin
            if (PiezaEjecuta[cont] as TPieza_Ejecuta).getTipoOperB() = 5 then valorB := (PiezaEjecuta[cont] as
TPieza_Ejecuta).getNumeroB() //¿Es un número?
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getTipoOperB() = 6 then valorB := VariableAlgebra[(PiezaEjecuta[cont]
as TPieza_Ejecuta).getVariableB()] //¿Es una variable?
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getTipoOperB() = 7 then valorB := (PiezaEjecuta[(PiezaEjecuta[cont]
as TPieza_Ejecuta).getAcumulaB()] as TPieza_Ejecuta).getValorPieza(); //¿Es una expresión anterior?

            if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '#' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(valorA * valorB)
```

```
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '+' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(valorA + valorB)
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '-' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(valorA - valorB)
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '*' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(valorA * valorB)
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '/' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(valorA / valorB)
            else if (PiezaEjecuta[cont] as TPieza_Ejecuta).getOperador() = '^' then (PiezaEjecuta[cont] as
TPieza_Ejecuta).setValorPieza(power(valorA, valorB));
          end;
        1, 2: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(sin(valorA)); end;
        3: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(cos(valorA)); end;
        4: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(tan(valorA)); end;
        5: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(abs(valorA)); end;
        6: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(arcsin(valorA)); end;
        7: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(arccos(valorA)); end;
        8: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(arctan(valorA)); end;
        9: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(ln(valorA)); end;
        10: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(ceil(valorA)); end;
        11: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(exp(valorA)); end;
        12: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(sqrt(valorA)); end;
        13: begin (PiezaEjecuta[cont] as TPieza_Ejecuta).setValorPieza(power(valorA, 0.333333333333)); end;
      end;
    end;
  except //Captura el error matemático
      on EMathError do
      begin
        Result := NaN;
        Exit;
      end;
  end;
  Calcular := (PiezaEjecuta[totalPiezaEjecuta - 1] as TPieza_Ejecuta).getValorPieza();
end;

// Da valor a las variables que tendrá la expresión algebraica
procedure TEvaluar.ValorVariable(variableAlgebra: char; valor: double);
begin
  self.VariableAlgebra[ord(variableAlgebra) - self.ASCIILETRA] := valor;
end;


end.
```

Anexo 6. Código completo en JavaScript

Evaluar.js

```
function Pieza_Simple()
{
    this.tipo; //Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
    this.funcion; //Que función es seno/coseno/tangente/sqrt
    this.operador; // +, -, *, /, ^
    this.numero; //Número real de la expresión
    this.variableAlgebra; //Variable de la expresión
    this.acumula;  //Indice de la microexpresión

    this.getTipo = function()
    {
      return this.tipo;
    }

    this.getFuncion = function()
    {
      return this.funcion;
    }

    this.getOperador = function()
    {
      return this.operador;
    }

    this.getNumero = function()
    {
       return this.numero;
    }

    this.getVariable = function()
    {
      return this.variableAlgebra;
    }

    this.getAcumula = function()
    {
      return this.acumula;
    }

    this.setAcumula = function(acumula)
    {
      this.tipo = 7;
      this.acumula = acumula;
    }

    this.ConstructorPieza_Simple = function(tipo, funcion, operador, numero, variable)
    {
      this.tipo = tipo;
      this.funcion = funcion;
      this.operador = operador;
      this.variableAlgebra = variable;
      this.acumula = 0;
      this.numero = numero;
    }
}

function Pieza_Ejecuta()
{
    this.valorPieza;

    this.funcion;

    this.tipo_operandoA;
    this.numeroA;
    this.variableA;
    this.acumulaA;

    this.operador;

    this.tipo_operandoB;
    this.numeroB;
    this.variableB;
    this.acumulaB;

    this.getValorPieza = function()
    {
      return this.valorPieza;
    }

    this.setValorPieza = function(valor)
    {
      this.valorPieza = valor;
    }

    this.getFuncion = function()
    {
```

```
      return this.funcion;
    }

    this.getTipoOperA = function()
    {
      return this.tipo_operandoA;
    }

    this.getNumeroA = function()
    {
      return this.numeroA;
    }

    this.getVariableA = function()
    {
      return this.variableA;
    }

    this.getAcumulaA = function()
    {
      return this.acumulaA;
    }

    this.getOperador = function()
    {
      return this.operador;
    }

    this.getTipoOperB = function()
    {
      return this.tipo_operandoB;
    }

    this.getNumeroB = function()
    {
      return this.numeroB;
    }

    this.getVariableB = function()
    {
      return this.variableB;
    }

    this.getAcumulaB = function()
    {
      return this.acumulaB;
    }

    this.ConstructorPieza_Ejecuta = function(funcion, tipo_operandoA, numeroA, variableA, acumulaA, operador, tipo_operandoB,
numeroB, variableB, acumulaB)
    {
      this.valorPieza = 0;

      this.funcion = funcion;

      this.tipo_operandoA = tipo_operandoA;
      this.numeroA = numeroA;
      this.variableA = variableA;
      this.acumulaA = acumulaA;

      this.operador = operador;

      this.tipo_operandoB = tipo_operandoB;
      this.numeroB = numeroB;
      this.variableB = variableB;
      this.acumulaB = acumulaB;
    }
}

function Evaluar()
{
    /* Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 */
    this.ASCIINUMERO = 48;

    /* Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 */
    this.ASCIILETRA = 97;

    /* Las funciones que soporta este evaluador */
    this.TAMANOFUNCION = 39;
    this.listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

    /* 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;

    //Listado de Piezas de análisis
    this.PiezaSimple = [];
```

```
    //Listado de Piezas de ejecución
    this.PiezaEjecuta = [];
    this.Contador_Acumula = 0;

    //Almacena los valores de las 26 diferentes variables que puede tener la expresión algebraica
    this.VariableAlgebra = [];

  //Valida la expresión algebraica
  this.EvaluaSintaxis = function(expresion)
  {
    //Hace 25 pruebas de sintaxis
    if (this.DobleTripleOperadorSeguido(expresion)) return 1;
    if (this.OperadorParentesisCierra(expresion)) return 2;
    if (this.ParentesisAbreOperador(expresion)) return 3;
    if (this.ParentesisDesbalanceados(expresion)) return 4;
    if (this.ParentesisVacio(expresion)) return 5;
    if (this.ParentesisBalanceIncorrecto(expresion)) return 6;
    if (this.ParentesisCierraNumero(expresion)) return 7;
    if (this.NumeroParentesisAbre(expresion)) return 8;
    if (this.DoblePuntoNumero(expresion)) return 9;
    if (this.ParentesisCierraVariable(expresion)) return 10;
    if (this.VariableluegoPunto(expresion)) return 11;
    if (this.PuntoluegoVariable(expresion)) return 12;
    if (this.NumeroAntesVariable(expresion)) return 13;
    if (this.VariableDespuesNumero(expresion)) return 14;
    if (this.Chequea4letras(expresion)) return 15;
    if (this.FuncionInvalida(expresion)) return 16;
    if (this.VariableInvalida(expresion)) return 17;
    if (this.VariableParentesisAbre(expresion)) return 18;
    if (this.ParCierraParAbre(expresion)) return 19;
    if (this.OperadorPunto(expresion)) return 20;
    if (this.ParAbrePunto(expresion)) return 21;
    if (this.PuntoParAbre(expresion)) return 22;
    if (this.ParCierraPunto(expresion)) return 23;
    if (this.PuntoOperador(expresion)) return 24;
    if (this.PuntoParCierra(expresion)) return 25;

    return 0; //No se detectó error de sintaxis
  }

  //Muestra mensaje de error sintáctico
  this.MensajeSintaxis = function(CodigoError)
  {
    switch(CodigoError)
    {
      case 0:  return "No se detectó error sintáctico en las 25 pruebas que se hicieron.";
      case 1:  return "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3";
      case 2:  return "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7";
      case 3:  return "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)";
      case 4:  return "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))";
      case 5:  return "5. Que haya paréntesis vacío. Ejemplo: 2-()*3";
      case 6:  return "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4";
      case 7: return "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)";
      case 8: return "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)";
      case 9: return "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2";
      case 10: return "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1";
      case 11: return "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3";
      case 12: return "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1";
      case 13: return "13. Un número antes de una variable. Ejemplo: 3x+1";
      case 14: return "14. Un número después de una variable. Ejemplo: x21+4";
      case 15: return "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9";
      case 16: return "16. Función inexistente. Ejemplo: 5*alo(78)";
      case 17: return "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5";
      case 18: return "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)";
      case 19: return "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)";
      case 20: return "20. Después de operador sigue un punto. Ejemplo: -.3+7";
      case 21: return "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)";
      case 22: return "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)";
      case 23: return "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2";
      case 24: return "24. Punto seguido de operador. Ejemplo: 5.*9+1";
      default: return "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5";
    }
  }

  //Retira caracteres inválidos. Pone la expresión entre paréntesis.
  this.TransformaExpresion = function(expr)
  {
    var validos = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
    var expr2 = expr.toLowerCase();
    var nuevaExpr = "(";
    for(var pos = 0; pos < expr2.length; pos++)
    {
      var letra = expr2.charAt(pos);
      for(var valida = 0; valida < validos.length; valida++)
        if (letra == validos.charAt(valida))
        {
          nuevaExpr += letra;
          break;
        }
    }
    nuevaExpr += ')';
    return nuevaExpr;
```

```
}

//1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
this.DobleTripleOperadorSeguido = function(expr)
{
  for(var pos = 0; pos < expr.length - 1; pos++)
  {
    var car1 = expr.charAt(pos);   //Extrae un carácter
    var car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

    //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
    if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
      if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^')
        return true;
  }

  for (var pos = 0; pos < expr.length - 2; pos++)
  {
    var car1 = expr.charAt(pos);   //Extrae un carácter
    var car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter
    var car3 = expr.charAt(pos + 2); //Extrae el siguiente carácter

    //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
    if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
      if (car2 == '+' || car2 == '-' || car2 == '*' || car2 == '/' || car2 == '^')
        if (car3 == '+' || car3 == '-' || car3 == '*' || car3 == '/' || car3 == '^')
          return true;
  }

  return false;  //No encontró doble/triple operador seguido
}

//2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
this.OperadorParentesisCierra = function(expr)
{
  for(var pos = 0; pos < expr.length - 1; pos++)
  {
    var car1 = expr.charAt(pos);   //Extrae un carácter

    //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
    if (car1 == '+' || car1 == '-' || car1 == '*' || car1 == '/' || car1 == '^')
      if (expr.charAt(pos + 1) == ')') return true;
  }
  return false; //No encontró operador seguido de un paréntesis que cierra
}

//3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
this.ParentesisAbreOperador = function(expr)
{
  for(var pos = 0; pos < expr.length - 1; pos++)
  {
    var car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

    //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
    if (expr.charAt(pos) == '(')
      if (car2 == '+' || car2 == '*' || car2 == '/' || car2 == '^') return true;
  }
  return false;  //No encontró paréntesis que abre seguido de un operador
}

//4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
this.ParentesisDesbalanceados = function(expr)
{
  var parabre = 0, parcierra = 0;
  for(var pos = 0; pos < expr.length; pos++)
  {
    var car1 = expr.charAt(pos);
    if (car1 == '(') parabre++;
    if (car1 == ')') parcierra++;
  }
  return parabre != parcierra;
}

//5. Que haya paréntesis vacío. Ejemplo: 2-()*3
this.ParentesisVacio = function(expr)
{
  //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
  for(var pos = 0; pos < expr.length - 1; pos++)
    if (expr.charAt(pos) == '(' && expr.charAt(pos + 1) == ')') return true;
  return false;
}

//6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
this.ParentesisBalanceIncorrecto = function(expr)
{
  var balance = 0;
  for(var pos = 0; pos < expr.length; pos++)
  {
    var car1 = expr.charAt(pos);   //Extrae un carácter
    if (car1 == '(') balance++;
    if (car1 == ')') balance--;
    if (balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
```

```
    }
    return false;
}

//7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
this.ParentesisCierraNumero = function(expr)
{
    for(var pos = 0; pos < expr.length - 1; pos++)
    {
        var car2 = expr.charAt(pos + 1); //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es número
        if (expr.charAt(pos) == ')')
            if (car2 >= '0' && car2 <= '9') return true;
    }
    return false;
}

//8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
this.NumeroParentesisAbre = function(expr)
{
    for(var pos = 0; pos < expr.length - 1; pos++)
    {
        var car1 = expr.charAt(pos);   //Extrae un carácter

        //Compara si el primer carácter es número y el siguiente es paréntesis que abre
        if (car1 >= '0' && car1 <= '9')
            if (expr.charAt(pos + 1) == '(') return true;
    }
    return false;
}

//9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
this.DoblePuntoNumero = function(expr)
{
    var totalpuntos = 0;
    for(var pos = 0; pos < expr.length; pos++)
    {
        var car1 = expr.charAt(pos);   //Extrae un carácter
        if ((car1 < '0' || car1 > '9') && car1 != '.') totalpuntos = 0;
        if (car1 == '.') totalpuntos++;
        if (totalpuntos > 1) return true;
    }
    return false;
}

//10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
this.ParentesisCierraVariable = function(expr)
{
    for(var pos = 0; pos < expr.length - 1; pos++)
        if (expr.charAt(pos) == ')') //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
                return true;
    return false;
}

//11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
this.VariableluegoPunto = function(expr)
{
    for(var pos = 0; pos < expr.length - 1; pos++)
        if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
            if (expr.charAt(pos + 1) == '.') return true;
    return false;
}

    //12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
    this.PuntoluegoVariable = function(expr)
    {
        for(var pos = 0; pos < expr.length - 1; pos++)
            if (expr.charAt(pos) == '.')
                if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
                    return true;
        return false;
    }

    //13. Un número antes de una variable. Ejemplo: 3x+1
    //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
    this.NumeroAntesVariable = function(expr)
    {
        for(var pos = 0; pos < expr.length - 1; pos++)
            if (expr.charAt(pos) >= '0' && expr.charAt(pos) <= '9')
                if (expr.charAt(pos + 1) >= 'a' && expr.charAt(pos + 1) <= 'z')
                    return true;
        return false;
    }

    //14. Un número después de una variable. Ejemplo: x21+4
    this.VariableDespuesNumero = function(expr)
    {
        for(var pos = 0; pos < expr.length - 1; pos++)
            if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
```

```
      if (expr.charAt(pos + 1) >= '0' && expr.charAt(pos + 1) <= '9')
        return true;
  return false;
}

//15. Chequea si hay 4 o más variables seguidas
this.Chequea4letras = function(expr)
{
  for(var pos = 0; pos < expr.length - 3; pos++)
  {
    var car1 = expr.charAt(pos);
    var car2 = expr.charAt(pos + 1);
    var car3 = expr.charAt(pos + 2);
    var car4 = expr.charAt(pos + 3);

    if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z' && car4 >= 'a' && car4 <= 'z')
      return true;
  }
  return false;
}

//16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
this.FuncionInvalida = function(expr)
{
  for(var pos = 0; pos < expr.length - 2; pos++)
  {
    var car1 = expr.charAt(pos);
    var car2 = expr.charAt(pos + 1);
    var car3 = expr.charAt(pos + 2);

    //Si encuentra tres letras seguidas
    if (car1 >= 'a' && car1 <= 'z' && car2 >= 'a' && car2 <= 'z' && car3 >= 'a' && car3 <= 'z')
    {
      if (pos >= expr.length - 4) return true; //Hay un error porque no sigue paréntesis
      if (expr.charAt(pos + 3) != '(') return true; //Hay un error porque no hay paréntesis
      if (this.EsFuncionInvalida(car1, car2, car3)) return true;
    }
  }
  return false;
}

//Chequea si las tres letras enviadas son una función
this.EsFuncionInvalida = function(car1, car2, car3)
{
  var listafunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";
  for(var pos = 0; pos <= listafunciones.length - 3; pos+=3)
  {
    var listfunc1 = listafunciones.charAt(pos);
    var listfunc2 = listafunciones.charAt(pos + 1);
    var listfunc3 = listafunciones.charAt(pos + 2);
    if (car1 == listfunc1 && car2 == listfunc2 && car3 == listfunc3) return false;
  }
  return true;
}

//17. Si detecta sólo dos letras seguidas es un error
this.VariableInvalida = function(expr)
{
  var cuentaletras = 0;
  for(var pos = 0; pos < expr.length; pos++)
  {
    if (expr.charAt(pos) >= 'a' && expr.charAt(pos) <= 'z')
      cuentaletras++;
    else
    {
      if (cuentaletras == 2) return true;
      cuentaletras = 0;
    }
  }
  return cuentaletras == 2;
}

//18. Antes de paréntesis que abre hay una letra
this.VariableParentesisAbre = function(expr)
{
  var cuentaletras = 0;
  for(var pos = 0; pos < expr.length; pos++)
  {
    var car1 = expr.charAt(pos);
    if (car1 >= 'a' && car1 <= 'z')
      cuentaletras++;
    else if (car1 == '(' && cuentaletras == 1)
      return true;
    else
      cuentaletras = 0;
  }
  return false;
}

//19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
this.ParCierraParAbre = function(expr)
{
```

```
      for (var pos = 0; pos < expr.length-1; pos++)
          if (expr.charAt(pos)==')' && expr.charAt(pos+1)=='(')
              return true;
      return false;
  }

  //20. Después de operador sigue un punto. Ejemplo: -.3+7
  this.OperadorPunto = function(expr)
  {
      for (var pos = 0; pos < expr.length-1; pos++)
          if (expr.charAt(pos)=='+' || expr.charAt(pos)=='-' || expr.charAt(pos)=='*' || expr.charAt(pos)=='/' ||
expr.charAt(pos)=='^')
                  if (expr.charAt(pos+1)=='.')
                    return true;
      return false;
  }

  //21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
  this.ParAbrePunto = function(expr)
  {
      for (var pos = 0; pos < expr.length-1; pos++)
          if (expr.charAt(pos)=='(' && expr.charAt(pos+1)=='.')
              return true;
      return false;
  }

  //22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
  this.PuntoParAbre = function(expr)
  {
      for (var pos = 0; pos < expr.length-1; pos++)
          if (expr.charAt(pos)=='.' && expr.charAt(pos+1)=='(')
              return true;
      return false;
  }

  //23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
  this.ParCierraPunto = function(expr)
  {
      for (var pos = 0; pos < expr.length-1; pos++)
          if (expr.charAt(pos)==')' && expr.charAt(pos+1)=='.')
              return true;
      return false;
  }

  //24. Punto seguido de operador. Ejemplo: 5.*9+1
  this.PuntoOperador = function(expr)
  {
      for (var pos = 0; pos < expr.length-1; pos++)
          if (expr.charAt(pos)=='.')
            if (expr.charAt(pos+1)=='+' || expr.charAt(pos+1)=='-' || expr.charAt(pos+1)=='*' || expr.charAt(pos+1)=='/' ||
expr.charAt(pos+1)=='^')
                    return true;
      return false;
  }

  //25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
  this.PuntoParCierra = function(expr)
  {
      for (var pos = 0; pos < expr.length-1; pos++)
          if (expr.charAt(pos)=='.' && expr.charAt(pos+1)==')')
              return true;
      return false;
  }

  /* Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones */
  this.ArreglaNegativos = function(expresion)
  {
    var NuevaExpresion = "";
    var NuevaExpresion2 = "";

    //Si detecta un operador y luego un menos, entonces reemplaza el menos con un "(0-1)#"
    for (var pos=0; pos<expresion.length; pos++)
    {
      var letra1 = expresion.charAt(pos);
      if (letra1=='+' || letra1=='-' || letra1=='*' || letra1=='/' || letra1=='^')
        if (expresion.charAt(pos+1)=='-')
        {
          NuevaExpresion += letra1 + "(0-1)#";
          pos++;
          continue;
        }
      NuevaExpresion += letra1;
    }

    //Si detecta un paréntesis que abre y luego un menos, entonces reemplaza el menos con un "(0-1)#"
    for (var pos=0; pos<NuevaExpresion.length; pos++)
    {
      var letra1 = NuevaExpresion.charAt(pos);
      if (letra1=='(')
        if (NuevaExpresion.charAt(pos+1)=='-')
        {
          NuevaExpresion2 += letra1 + "(0-1)#";
```

```
          pos++;
          continue;
        }
      NuevaExpresion2 += letra1;
    }

    return NuevaExpresion2;
  }


    //Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
    this.Analizar = function(expresion)
    {
      this.PiezaSimple.length = 0;
      this.PiezaEjecuta.length = 0;
      this.Generar_Piezas_Simples(expresion);
      this.Generar_Piezas_Ejecucion();
      /*var totalPiezaEjecuta = this.PiezaEjecuta.length;
      for (var cont = 0; cont < totalPiezaEjecuta; cont++)
        this.PiezaEjecuta[cont].Imprime(cont);*/
    }

    //Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
    this.Generar_Piezas_Simples = function(expresion)
    {
      var longExpresion = expresion.length;
      var NumeroPiezaSimple = 0;

      //Variables requeridas para armar un número
      var parteentera = 0;
      var partedecimal = 0;
      var divide = 1;
      var entero = true;
      var armanumero = false;

      for (var cont = 0; cont < longExpresion; cont++) //Va de letra en letra de la expresión
      {
        var letra = expresion.charAt(cont);
        if (letra == '.')  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
          entero = false;
        else if (letra >= '0' && letra <= '9')  //Si es un número, entonces lo va armando
        {
          armanumero = true;
          if (entero)
            parteentera = parteentera * 10 + parseFloat(letra); //La parte entera del número
          else
          {
            divide *= 10;
            partedecimal = partedecimal * 10 + parseFloat(letra); //La parte decimal del número
          }
        }
        else
        {
          if (armanumero) //Si tenía armado un número, entonces crea la pieza ESNUMERO
          {
            objeto = new Pieza_Simple();
            objeto.ConstructorPieza_Simple(this.ESNUMERO, 0, '0', parteentera+partedecimal/divide, 0, 0);
            this.PiezaSimple[NumeroPiezaSimple++] = objeto;
            parteentera = 0;
            partedecimal = 0;
            divide = 1;
            entero = true;
            armanumero = false;
          }

          if (letra == '+' || letra == '-' || letra == '*' || letra == '/' || letra == '^' || letra == '#') { objeto = new
Pieza_Simple(); objeto.ConstructorPieza_Simple(this.ESOPERADOR, 0, letra, 0, 0, 0); this.PiezaSimple[NumeroPiezaSimple++] =
objeto; }
          else if (letra == '(') { objeto = new Pieza_Simple(); objeto.ConstructorPieza_Simple(this.ESPARABRE, 0, '0', 0, 0, 0);
this.PiezaSimple[NumeroPiezaSimple++] = objeto; } //¿Es paréntesis que abre?
          else if (letra == ')') { objeto = new Pieza_Simple(); objeto.ConstructorPieza_Simple(this.ESPARCIERRA, 0, '0', 0, 0,
0); this.PiezaSimple[NumeroPiezaSimple++] = objeto; }//¿Es paréntesis que cierra?
          else if (letra >= 'a' && letra <= 'z') //¿Es variable o función?
          {
            /* Detecta si es una función porque tiene dos letras seguidas */
            if (cont < longExpresion - 1)
            {
              letra2 = expresion.charAt(cont + 1); /* Chequea si el siguiente carácter es una letra, dado el caso es una función
*/
              if (letra2 >= 'a' && letra2 <= 'z')
              {
                letra3 = expresion.charAt(cont + 2);
                funcionDetectada = 1;  /* Identifica la función */
                for (funcion = 0; funcion <= this.TAMANOFUNCION; funcion += 3)
                {
                  if (letra == this.listaFunciones.charAt(funcion)
                    && letra2 == this.listaFunciones.charAt(funcion + 1)
                    && letra3 == this.listaFunciones.charAt(funcion + 2))
                    break;
                  funcionDetectada++;
                }
                objeto = new Pieza_Simple();
```

```
                objeto.ConstructorPieza_Simple(this.ESFUNCION, funcionDetectada, '0', 0, 0, 0);  //Adiciona función a la lista
                this.PiezaSimple[NumeroPiezaSimple++] = objeto;
                cont += 3; /* Mueve tres caracteres  sin(  [s][i][n][(] */
              }
              else /* Es una variable, no una función */
              {
                objeto = new Pieza_Simple();
                objeto.ConstructorPieza_Simple(this.ESVARIABLE, 0, '0', 0, letra.charCodeAt(0) - this.ASCIILETRA, 0);
                this.PiezaSimple[NumeroPiezaSimple++] = objeto;
              }
            }
            else /* Es una variable, no una función */
            {
              objeto = new Pieza_Simple();
              objeto.ConstructorPieza_Simple(this.ESVARIABLE, 0, '0', 0, letra.charCodeAt(0) - this.ASCIILETRA, 0);
              this.PiezaSimple[NumeroPiezaSimple++] = objeto;
            }
          }
        }
      }
      if (armanumero) { objeto = new Pieza_Simple(); objeto.ConstructorPieza_Simple(this.ESNUMERO, 0, '0',
parteentera+partedecimal/divide, 0, 0);  this.PiezaSimple[NumeroPiezaSimple++] = objeto; }
    }

    //Toma las piezas simples y las convierte en piezas de ejecución de funciones
    //Acumula = función (operando(número/variable/acumula))
    this.Generar_Piezas_Ejecucion = function()
    {
      var cont = this.PiezaSimple.length - 1;
      this.Contador_Acumula = 0;
      do
      {
        if (this.PiezaSimple[cont].getTipo() == this.ESPARABRE || this.PiezaSimple[cont].getTipo() == this.ESFUNCION)
        {
          this.Generar_Piezas_Operador("#", "#", cont);  //Primero evalúa las potencias
          this.Generar_Piezas_Operador("^", "^", cont);  //Primero evalúa las potencias
          this.Generar_Piezas_Operador("*", "/", cont);  //Luego evalúa multiplicar y dividir
          this.Generar_Piezas_Operador("+", "-", cont);  //Finalmente evalúa sumar y restar

          //Crea pieza de ejecución
          objeto = new Pieza_Ejecuta();
          objeto.ConstructorPieza_Ejecuta(this.PiezaSimple[cont].getFuncion(),
              this.PiezaSimple[cont + 1].getTipo(), this.PiezaSimple[cont + 1].getNumero(), this.PiezaSimple[cont +
1].getVariable(), this.PiezaSimple[cont + 1].getAcumula(),
              '+', this.ESNUMERO, 0, 0, 0);
          this.PiezaEjecuta[this.Contador_Acumula] = objeto;

          //La pieza pasa a ser de tipo Acumulador
          this.PiezaSimple[cont + 1].setAcumula(this.Contador_Acumula++);

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

    //Toma las piezas simples y las convierte en piezas de ejecución
    //Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
    this.Generar_Piezas_Operador = function(operA, operB, inicio)
    {
      var cont = inicio + 1;
      do
      {
        if ((this.PiezaSimple[cont].getTipo() == this.ESOPERADOR) && (this.PiezaSimple[cont].getOperador() == operA ||
this.PiezaSimple[cont].getOperador() == operB))
        {
          //Crea pieza de ejecución
          objeto = new Pieza_Ejecuta();
          objeto.ConstructorPieza_Ejecuta(0,
              this.PiezaSimple[cont - 1].getTipo(),
              this.PiezaSimple[cont - 1].getNumero(), this.PiezaSimple[cont - 1].getVariable(), this.PiezaSimple[cont -
1].getAcumula(),
              this.PiezaSimple[cont].getOperador(),
              this.PiezaSimple[cont + 1].getTipo(),
              this.PiezaSimple[cont + 1].getNumero(), this.PiezaSimple[cont + 1].getVariable(), this.PiezaSimple[cont +
1].getAcumula());
          this.PiezaEjecuta[this.Contador_Acumula] = objeto;

          //Elimina la pieza del operador y la siguiente
          this.PiezaSimple.splice(cont, 1);
          this.PiezaSimple.splice(cont, 1);

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

          //Cambia la pieza anterior por pieza acumula
          this.PiezaSimple[cont].setAcumula(this.Contador_Acumula++);
        }
        cont++;
      } while (cont < this.PiezaSimple.length && this.PiezaSimple[cont].getTipo() != this.ESPARCIERRA);
```

```
    }

    //Calcula la expresión convertida en piezas de ejecución
    this.Calcular = function()
    {
      var valorA=0, valorB=0;
      var totalPiezaEjecuta = this.PiezaEjecuta.length;

      for (var cont = 0; cont < totalPiezaEjecuta; cont++)
      {
        switch (this.PiezaEjecuta[cont].getTipoOperA())
        {
          case 5: valorA = this.PiezaEjecuta[cont].getNumeroA(); break; //¿Es un número?
          case 6: valorA = this.VariableAlgebra[this.PiezaEjecuta[cont].getVariableA()]; break;  //¿Es una variable?
          case 7: valorA = this.PiezaEjecuta[this.PiezaEjecuta[cont].getAcumulaA()].getValorPieza(); break; //¿Es una expresión
anterior?
        }
        if (isNaN(valorA) || !isFinite(valorA)) return valorA;

        switch (this.PiezaEjecuta[cont].getFuncion())
        {
          case 0:
            switch (this.PiezaEjecuta[cont].getTipoOperB())
            {
              case 5: valorB = this.PiezaEjecuta[cont].getNumeroB(); break; //¿Es un número?
              case 6: valorB = this.VariableAlgebra[this.PiezaEjecuta[cont].getVariableB()]; break;  //¿Es una variable?
              case 7: valorB = this.PiezaEjecuta[this.PiezaEjecuta[cont].getAcumulaB()].getValorPieza(); break; //¿Es una
expresión anterior?
            }
            if (isNaN(valorB) || !isFinite(valorB)) return valorB;

            switch (this.PiezaEjecuta[cont].getOperador())
            {
              case '#': this.PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
              case '+': this.PiezaEjecuta[cont].setValorPieza(valorA + valorB); break;
              case '-': this.PiezaEjecuta[cont].setValorPieza(valorA - valorB); break;
              case '*': this.PiezaEjecuta[cont].setValorPieza(valorA * valorB); break;
              case '/': this.PiezaEjecuta[cont].setValorPieza(valorA / valorB); break;
              case '^': this.PiezaEjecuta[cont].setValorPieza(Math.pow(valorA, valorB)); break;
            }
            break;
          case 1:
          case 2: this.PiezaEjecuta[cont].setValorPieza(Math.sin(valorA)); break;
          case 3: this.PiezaEjecuta[cont].setValorPieza(Math.cos(valorA)); break;
          case 4: this.PiezaEjecuta[cont].setValorPieza(Math.tan(valorA)); break;
          case 5: this.PiezaEjecuta[cont].setValorPieza(Math.abs(valorA)); break;
          case 6: this.PiezaEjecuta[cont].setValorPieza(Math.asin(valorA)); break;
          case 7: this.PiezaEjecuta[cont].setValorPieza(Math.acos(valorA)); break;
          case 8: this.PiezaEjecuta[cont].setValorPieza(Math.atan(valorA)); break;
          case 9: this.PiezaEjecuta[cont].setValorPieza(Math.log(valorA)); break;
          case 10: this.PiezaEjecuta[cont].setValorPieza(Math.ceil(valorA)); break;
          case 11: this.PiezaEjecuta[cont].setValorPieza(Math.exp(valorA)); break;
          case 12: this.PiezaEjecuta[cont].setValorPieza(Math.sqrt(valorA)); break;
          case 13: this.PiezaEjecuta[cont].setValorPieza(Math.pow(valorA, 0.333333333333)); break;
        }
      }
      return this.PiezaEjecuta[totalPiezaEjecuta - 1].getValorPieza();
    }

    // Da valor a las variables que tendrá la expresión algebraica
    this.ValorVariable = function(varAlgebra, valor)
    {
      this.VariableAlgebra[varAlgebra.charCodeAt(0) - this.ASCIILETRA] = valor;
    }
}
```

Anexo 7. Código completo en PHP

Pieza_Simple.php

```php
<?php
class Pieza_Simple
{
	var $tipo; //Función, parentesis_abre, parentesis_cierra, operador, numero, variable, abreviación
	var $funcion; //Que función es seno/coseno/tangente/sqrt
	var $operador; // +, -, *, /, ^
	var $numero; //Número real de la expresión
	var $variableAlgebra; //Variable de la expresión
	var $acumula;  //Indice de la microexpresión

	public function getTipo() { return $this->tipo;}
	public function getFuncion() { return $this->funcion; }
	public function getOperador() {	return $this->operador; }
	public function getNumero() { return $this->numero;  }
	public function getVariable() {	return $this->variableAlgebra;	}
	public function getAcumula() { return $this->acumula; }
	public function setAcumula($acumula) { $this->tipo = 7; $this->acumula = $acumula; }

	public function ConstructorPiezaSimple($tipo, $funcion, $operador, $numero, $variable)
	{
		$this->tipo = $tipo;
		$this->funcion = $funcion;
		$this->operador = $operador;
		$this->variableAlgebra = $variable;
		$this->acumula = 0;
		$this->numero = $numero;
	}
}
?>
```

Pieza_Ejecuta.php

```php
<?php
class Pieza_Ejecuta
{
	var $valorPieza;

	var $funcion;

	var $tipo_operandoA;
	var $numeroA;
	var $variableA;
	var $acumulaA;

	var $operador;

	var $tipo_operandoB;
	var $numeroB;
	var $variableB;
	var $acumulaB;

	public function getValorPieza() { return $this->valorPieza;	}
	public function setValorPieza($valor) { $this->valorPieza = $valor;	}
	public function getFuncion() { return $this->funcion; }
	public function getTipoOperA() { return $this->tipo_operandoA; }
	public function getNumeroA() { return $this->numeroA; }
	public function getVariableA() { return $this->variableA; }
	public function getAcumulaA() {	return $this->acumulaA; }
	public function getOperador() {	return $this->operador; }
	public function getTipoOperB() { return $this->tipo_operandoB; }
	public function getNumeroB() { return $this->numeroB; }
	public function getVariableB() { return $this->variableB;}
	public function getAcumulaB() {	return $this->acumulaB; }

	public function ConstructorPiezaEjecuta($funcion, $tipo_operandoA, $numeroA, $variableA, $acumulaA, $operador,
$tipo_operandoB, $numeroB, $variableB, $acumulaB)
	{
		$this->valorPieza = 0;

		$this->funcion = $funcion;

		$this->tipo_operandoA = $tipo_operandoA;
		$this->numeroA = $numeroA;
		$this->variableA = $variableA;
		$this->acumulaA = $acumulaA;

		$this->operador = $operador;

		$this->tipo_operandoB = $tipo_operandoB;
		$this->numeroB = $numeroB;
		$this->variableB = $variableB;
		$this->acumulaB = $acumulaB;
	}
```

```
}
?>
```

Evaluar.php

```
<?php
include("Pieza_Simple.php");
include("Pieza_Ejecuta.php");

class Evaluar
{
   /* Esta constante sirve para que se reste al carácter y se obtenga el número.  Ejemplo:  '7' - ASCIINUMERO =  7 */
   var $ASCIINUMERO = 48;

   /* Esta constante sirve para que se reste al carácter y se obtenga el número de la letra. Ejemplo:  'b' - ASCIILETRA =  1 */
   var $ASCIILETRA = 97;

   /* Las funciones que soporta este evaluador */
   var $TAMANOFUNCION = 39;
   var $listaFunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";

   /* 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;

   //Listado de Piezas de análisis
   var $PiezaSimple = array();

   //Listado de Piezas de ejecución
   var $PiezaEjecuta = array();
   var $Contador_Acumula = 0;

   //Almacena los valores de las 26 diferentes variables que puede tener la expresión algebraica
   var $VariableAlgebra = array();

  //Valida la expresión algebraica
  public function EvaluaSintaxis($expresion)
  {
    //Hace 25 pruebas de sintaxis
    if ($this->DobleTripleOperadorSeguido($expresion)) return 1;
    if ($this->OperadorParentesisCierra($expresion)) return 2;
    if ($this->ParentesisAbreOperador($expresion)) return 3;
    if ($this->ParentesisDesbalanceados($expresion)) return 4;
    if ($this->ParentesisVacio($expresion)) return 5;
    if ($this->ParentesisBalanceIncorrecto($expresion)) return 6;
    if ($this->ParentesisCierraNumero($expresion)) return 7;
    if ($this->NumeroParentesisAbre($expresion)) return 8;
    if ($this->DoblePuntoNumero($expresion)) return 9;
    if ($this->ParentesisCierraVariable($expresion)) return 10;
    if ($this->VariableluegoPunto($expresion)) return 11;
    if ($this->PuntoluegoVariable($expresion)) return 12;
    if ($this->NumeroAntesVariable($expresion)) return 13;
    if ($this->VariableDespuesNumero($expresion)) return 14;
    if ($this->Chequea4letras($expresion)) return 15;
    if ($this->FuncionInvalida($expresion)) return 16;
    if ($this->VariableInvalida($expresion)) return 17;
    if ($this->VariableParentesisAbre($expresion)) return 18;
    if ($this->ParCierraParAbre($expresion)) return 19;
    if ($this->OperadorPunto($expresion)) return 20;
    if ($this->ParAbrePunto($expresion)) return 21;
    if ($this->PuntoParAbre($expresion)) return 22;
    if ($this->ParCierraPunto($expresion)) return 23;
    if ($this->PuntoOperador($expresion)) return 24;
    if ($this->PuntoParCierra($expresion)) return 25;

    return 0; //No se detectó error de sintaxis
  }

  //Muestra mensaje de error sintáctico
  public function MensajeSintaxis($CodigoError)
  {
    switch($CodigoError)
    {
      case 0:  return "No se detectó error sintáctico en las 25 pruebas que se hicieron.";
      case 1:  return "1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3";
      case 2:  return "2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7";
      case 3:  return "3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)";
      case 4:  return "4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))";
      case 5:  return "5. Que haya paréntesis vacío. Ejemplo: 2-()*3";
      case 6:  return "6. Paréntesis que abre no corresponde con el que cierra. Ejemplo: 2+3)-2*(4";
      case 7:  return "7. Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)";
      case 8:  return "8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)";
      case 9:  return "9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2";
      case 10: return "10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1";
      case 11: return "11. Una variable seguida de un punto. Ejemplo: 4-z.1+3";
```

```
      case 12: return "12. Un punto seguido de una variable. Ejemplo: 7-2.p+1";
      case 13: return "13. Un número antes de una variable. Ejemplo: 3x+1";
      case 14: return "14. Un número después de una variable. Ejemplo: x21+4";
      case 15: return "15. Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9";
      case 16: return "16. Función inexistente. Ejemplo: 5*alo(78)";
      case 17: return "17. Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5";
      case 18: return "18. Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)";
      case 19: return "19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)";
      case 20: return "20. Después de operador sigue un punto. Ejemplo: -.3+7";
      case 21: return "21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)";
      case 22: return "22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)";
      case 23: return "23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2";
      case 24: return "24. Punto seguido de operador. Ejemplo: 5.*9+1";
      default: return "25. Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5";
    }
  }

  //Retira caracteres inválidos. Pone la expresión entre paréntesis.
  public function TransformaExpresion($expr)
  {
    $validos = "abcdefghijklmnopqrstuvwxyz0123456789.+-*/^()";
    $expr2 = strtolower($expr);
    $nuevaExpr = "(";
    for($pos = 0; $pos < strlen($expr2); $pos++)
    {
      $letra = $expr2{$pos};
      for($valida = 0; $valida < strlen($validos); $valida++)
        if ($letra == $validos{$valida})
        {
          $nuevaExpr .= $letra;
          break;
        }
    }
    $nuevaExpr .= ')';
    return $nuevaExpr;
  }

  //1. Dos o más operadores estén seguidos. Ejemplo: 2++4, 5-*3
  function DobleTripleOperadorSeguido($expr)
  {
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
    {
      $car1 = $expr{$pos};   //Extrae un carácter
      $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

      //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
      if ($car1 == '+' || $car1 == '-' || $car1 == '*' || $car1 == '/' || $car1 == '^')
        if ($car2 == '+' || $car2 == '*' || $car2 == '/' || $car2 == '^')
          return true;
    }

    for ($pos = 0; $pos < strlen($expr) - 2; $pos++)
    {
      $car1 = $expr{$pos};   //Extrae un carácter
      $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter
      $car3 = $expr{$pos + 2}; //Extrae el siguiente carácter


      //Compara si el carácter y el siguiente son operadores, dado el caso retorna true
      if ($car1 == '+' || $car1 == '-' || $car1 == '*' || $car1 == '/' || $car1 == '^')
        if ($car2 == '+' || $car2 == '-' || $car2 == '*' || $car2 == '/' || $car2 == '^')
          if ($car3 == '+' || $car3 == '-' || $car3 == '*' || $car3 == '/' || $car3 == '^')
            return true;
    }

    return false;  //No encontró doble/triple operador seguido
  }

  //2. Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7
  function OperadorParentesisCierra($expr)
  {
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
    {
      $car1 = $expr{$pos};   //Extrae un carácter

      //Compara si el primer carácter es operador y el siguiente es paréntesis que cierra
      if ($car1 == '+' || $car1 == '-' || $car1 == '*' || $car1 == '/' || $car1 == '^')
        if ($expr{$pos + 1} == ')') return true;
    }
    return false; //No encontró operador seguido de un paréntesis que cierra
  }

  //3. Un paréntesis que abre seguido de un operador. Ejemplo: 2-(*3)
  function ParentesisAbreOperador($expr)
  {
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
    {
      $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

      //Compara si el primer carácter es paréntesis que abre y el siguiente es operador
      if ($expr{$pos} == '(')
        if ($car2 == '+' || $car2 == '*' || $car2 == '/' || $car2 == '^') return true;
```

```
    }
    return false;  //No encontró paréntesis que abre seguido de un operador
}

//4. Que los paréntesis estén desbalanceados. Ejemplo: 3-(2*4))
function ParentesisDesbalanceados($expr)
{
    $parabre = 0; $parcierra = 0;
    for($pos = 0; $pos < strlen($expr); $pos++)
    {
        $car1 = $expr{$pos};
        if ($car1 == '(') $parabre++;
        if ($car1 == ')') $parcierra++;
    }
    return $parabre != $parcierra;
}

//5. Que haya paréntesis vacío. Ejemplo: 2-()*3
function ParentesisVacio($expr)
{
    //Compara si el primer carácter es paréntesis que abre y el siguiente es paréntesis que cierra
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
        if ($expr{$pos} == '(' && $expr{$pos + 1} == ')') return true;
    return false;
}

//6. Así estén balanceados los paréntesis no corresponde el que abre con el que cierra. Ejemplo: 2+3)-2*(4
function ParentesisBalanceIncorrecto($expr)
{
    $balance = 0;
    for($pos = 0; $pos < strlen($expr); $pos++)
    {
        $car1 = $expr{$pos};    //Extrae un carácter
        if ($car1 == '(') $balance++;
        if ($car1 == ')') $balance--;
        if ($balance < 0) return true; //Si cae por debajo de cero es que el balance es erróneo
    }
    return false;
}

//7. Un paréntesis que cierra y sigue un número o paréntesis que abre. Ejemplo: (3-5)7-(1+2)(3/6)
function ParentesisCierraNumero($expr)
{
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
    {
        $car2 = $expr{$pos + 1}; //Extrae el siguiente carácter

        //Compara si el primer carácter es paréntesis que cierra y el siguiente es número
        if ($expr{$pos} == ')')
            if ($car2 >= '0' && $car2 <= '9') return true;
    }
    return false;
}

//8. Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)
function NumeroParentesisAbre($expr)
{
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
    {
        $car1 = $expr{$pos};    //Extrae un carácter

        //Compara si el primer carácter es número y el siguiente es paréntesis que abre
        if ($car1 >= '0' && $car1 <= '9')
            if ($expr{$pos + 1} == '(') return true;
    }
    return false;
}

//9. Doble punto en un número de tipo real. Ejemplo: 3-2..4+1  7-6.46.1+2
function DoblePuntoNumero($expr)
{
    $totalpuntos = 0;
    for($pos = 0; $pos < strlen($expr); $pos++)
    {
        $car1 = $expr{$pos};    //Extrae un carácter
        if (($car1 < '0' || $car1 > '9') && $car1 != '.') $totalpuntos = 0;
        if ($car1 == '.') $totalpuntos++;
        if ($totalpuntos > 1) return true;
    }
    return false;
}

//10. Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1
function ParentesisCierraVariable($expr)
{
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
        if ($expr{$pos} == ')') //Compara si el primer carácter es paréntesis que cierra y el siguiente es letra
            if ($expr{$pos + 1} >= 'a' && $expr{$pos + 1} <= 'z')
                return true;
    return false;
}
```

```
  //11. Una variable seguida de un punto. Ejemplo: 4-z.1+3
  function VariableluegoPunto($expr)
  {
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
      if ($expr{$pos} >= 'a' && $expr{$pos} <= 'z')
        if ($expr{$pos + 1} == '.') return true;
    return false;
  }

  //12. Un punto seguido de una variable. Ejemplo: 7-2.p+1
  function PuntoluegoVariable($expr)
  {
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
      if ($expr{$pos} == '.')
        if ($expr{$pos + 1} >= 'a' && $expr{$pos + 1} <= 'z')
          return true;
    return false;
  }

  //13. Un número antes de una variable. Ejemplo: 3x+1
  //Nota: Algebraicamente es aceptable 3x+1 pero entonces vuelve más complejo un evaluador porque debe saber que 3x+1 es en
realidad 3*x+1
  function NumeroAntesVariable($expr)
  {
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
      if ($expr{$pos} >= '0' && $expr{$pos} <= '9')
        if ($expr{$pos + 1} >= 'a' && $expr{$pos + 1} <= 'z')
          return true;
    return false;
  }

  //14. Un número después de una variable. Ejemplo: x21+4
  function VariableDespuesNumero($expr)
  {
    for($pos = 0; $pos < strlen($expr) - 1; $pos++)
      if ($expr{$pos} >= 'a' && $expr{$pos} <= 'z')
        if ($expr{$pos + 1} >= '0' && $expr{$pos + 1} <= '9')
          return true;
    return false;
  }

  //15. Chequea si hay 4 o más letras seguidas
  function Chequea4letras($expr)
  {
    for($pos = 0; $pos < strlen($expr) - 3; $pos++)
    {
      $car1 = $expr{$pos};
      $car2 = $expr{$pos + 1};
      $car3 = $expr{$pos + 2};
      $car4 = $expr{$pos + 3};

      if ($car1 >= 'a' && $car1 <= 'z' && $car2 >= 'a' && $car2 <= 'z' && $car3 >= 'a' && $car3 <= 'z' && $car4 >= 'a' && $car4
<= 'z')
        return true;
    }
    return false;
  }

  //16. Si detecta tres letras seguidas y luego un paréntesis que abre, entonces verifica si es función o no
  function FuncionInvalida($expr)
  {
    for($pos = 0; $pos < strlen($expr) - 2; $pos++)
    {
      $car1 = $expr{$pos};
      $car2 = $expr{$pos + 1};
      $car3 = $expr{$pos + 2};

      //Si encuentra tres letras seguidas
      if ($car1 >= 'a' && $car1 <= 'z' && $car2 >= 'a' && $car2 <= 'z' && $car3 >= 'a' && $car3 <= 'z')
      {
        if ($pos >= strlen($expr) - 4) return true; //Hay un error porque no sigue paréntesis
        if ($expr{$pos + 3} != '(') return true; //Hay un error porque no hay paréntesis
        if ($this->EsFuncionInvalida($car1, $car2, $car3)) return true;
      }
    }
    return false;
  }

  //Chequea si las tres letras enviadas son una función
  function EsFuncionInvalida($car1, $car2, $car3)
  {
    $listafunciones = "sinsencostanabsasnacsatnlogceiexpsqrrcb";
    for($pos = 0; $pos <= strlen($listafunciones) - 3; $pos+=3)
    {
      $listfunc1 = $listafunciones{$pos};
      $listfunc2 = $listafunciones{$pos + 1};
      $listfunc3 = $listafunciones{$pos + 2};
      if ($car1 == $listfunc1 && $car2 == $listfunc2 && $car3 == $listfunc3) return false;
    }
    return true;
  }
```

```php
//17. Si detecta sólo dos letras seguidas es un error
function VariableInvalida($expr)
{
  $cuentaletras = 0;
  for($pos = 0; $pos < strlen($expr); $pos++)
  {
    if ($expr{$pos} >= 'a' && $expr{$pos} <= 'z')
      $cuentaletras++;
    else
    {
      if ($cuentaletras == 2) return true;
      $cuentaletras = 0;
    }
  }
  return $cuentaletras == 2;
}

//18. Antes de paréntesis que abre hay una letra
function VariableParentesisAbre($expr)
{
  $cuentaletras = 0;
  for($pos = 0; $pos < strlen($expr); $pos++)
  {
    $car1 = $expr{$pos};
    if ($car1 >= 'a' && $car1 <= 'z')
      $cuentaletras++;
    else if ($car1 == '(' && $cuentaletras == 1)
      return true;
    else
      $cuentaletras = 0;
  }
  return false;
}

//19. Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)
function ParCierraParAbre($expr)
{
    for ($pos = 0; $pos < strlen($expr)-1; $pos++)
        if ($expr{$pos}==')' && $expr{$pos+1}=='(')
            return true;
    return false;
}

//20. Después de operador sigue un punto. Ejemplo: -.3+7
function OperadorPunto($expr)
{
    for ($pos = 0; $pos < strlen($expr)-1; $pos++)
        if ($expr{$pos}=='+' || $expr{$pos}=='-' || $expr{$pos}=='*' || $expr{$pos}=='/' || $expr{$pos}=='^')
                if ($expr{$pos+1}=='.')
                  return true;
    return false;
}

//21. Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)
function ParAbrePunto($expr)
{
    for ($pos = 0; $pos < strlen($expr)-1; $pos++)
        if ($expr{$pos}=='(' && $expr{$pos+1}=='.')
            return true;
    return false;
}

//22. Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)
function PuntoParAbre($expr)
{
    for ($pos = 0; $pos < strlen($expr)-1; $pos++)
        if ($expr{$pos}=='.' && $expr{$pos+1}=='(')
            return true;
    return false;
}

//23. Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2
function ParCierraPunto($expr)
{
    for ($pos = 0; $pos < strlen($expr)-1; $pos++)
        if ($expr{$pos}==')' && $expr{$pos+1}=='.')
            return true;
    return false;
}

//24. Punto seguido de operador. Ejemplo: 5.*9+1
function PuntoOperador($expr)
{
    for ($pos = 0; $pos < strlen($expr)-1; $pos++)
        if ($expr{$pos}=='.')
          if ($expr{$pos+1}=='+' || $expr{$pos+1}=='-' || $expr{$pos+1}=='*' || $expr{$pos+1}=='/' || $expr{$pos+1}=='^')
                return true;
    return false;
}

//25. Punto y sigue paréntesis que cierra. Ejemplo: (3+2.)*5
function PuntoParCierra($expr)
```

```
{
    for ($pos = 0; $pos < strlen($expr)-1; $pos++)
        if ($expr{$pos}=='.' && $expr{$pos+1}==')')
            return true;
    return false;
}

/* Convierte una expresión con el menos unario en una expresión valida para el evaluador de expresiones */
public function ArreglaNegativos($expresion)
{
  $NuevaExpresion = "";
  $NuevaExpresion2 = "";

  //Si detecta un operador y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  for ($pos=0; $pos<strlen($expresion); $pos++)
  {
    $letra1 = $expresion{$pos};
    if ($letra1=='+' || $letra1=='-' || $letra1=='*' || $letra1=='/' || $letra1=='^')
      if ($expresion{$pos+1}=='-')
      {
        $NuevaExpresion .= $letra1 . "(0-1)#";
        $pos++;
        continue;
      }
    $NuevaExpresion .= $letra1;
  }

  //Si detecta un paréntesis que abre y luego un menos, entonces reemplaza el menos con un "(0-1)#"
  for ($pos=0; $pos<strlen($NuevaExpresion); $pos++)
  {
    $letra1 = $NuevaExpresion{$pos};
    if ($letra1=='(')
      if ($NuevaExpresion{$pos+1}=='-')
      {
        $NuevaExpresion2 .= $letra1 . "(0-1)#";
        $pos++;
        continue;
      }
    $NuevaExpresion2 .= $letra1;
  }

  return $NuevaExpresion2;
}



//Inicializa las listas, convierte la expresión en piezas simples y luego en piezas de ejecución
function Analizar($expresion)
{
   unset($this->PiezaSimple);
   unset($this->PiezaEjecuta);
   $this->Generar_Piezas_Simples($expresion);
   /*for ($cont=0; $cont< sizeof($this->PiezaSimple); $cont++)
      echo $this->PiezaSimple[$cont]->Imprime();
      echo "<br>";*/
   $this->Generar_Piezas_Ejecucion();
   /*for ($cont=0; $cont< sizeof($this->PiezaEjecuta); $cont++)
      echo $this->PiezaEjecuta[$cont]->Imprime($cont);
      echo "<br>";*/
}

//Convierte la expresión en piezas simples: números # paréntesis # variables # operadores # funciones
function Generar_Piezas_Simples($expresion)
{
   $longExpresion = strlen($expresion);

   //Variables requeridas para armar un número
   $parteentera = 0;
   $partedecimal = 0;
   $divide = 1;
   $entero = true;
   $armanumero = false;

   for ($cont = 0; $cont < $longExpresion; $cont++) //Va de letra en letra de la expresión
   {
      $letra = $expresion{$cont};

      if ($letra == '.')  //Si letra es . entonces el resto de digitos leídos son la parte decimal del número
       $entero = false;
      else if ($letra >= '0' && $letra <= '9')  //Si es un número, entonces lo va armando
      {
       $armanumero = true;
       if ($entero)
          $parteentera = $parteentera * 10 + ord($letra) - $this->ASCIINUMERO; //La parte entera del número
       else
       {
          $divide *= 10;
          $partedecimal = $partedecimal * 10 + ord($letra) - $this->ASCIINUMERO; //La parte decimal del número
       }
      }
      else
      {
```

```
            if ($armanumero) //Si tenía armado un número, entonces crea la pieza ESNUMERO
            {
               $objeto = new Pieza_Simple();
               $objeto->ConstructorPiezaSimple($this->ESNUMERO, 0, '0', $parteentera+$partedecimal/$divide, 0);
               $this->PiezaSimple[] = $objeto;
               $parteentera = 0;
               $partedecimal = 0;
               $divide = 1;
               $entero = true;
               $armanumero = false;
            }

            if ($letra == '+' || $letra == '-' || $letra == '*' || $letra == '/' || $letra == '^' || $letra == '#')
            {
               $objeto = new Pieza_Simple();
               $objeto->ConstructorPiezaSimple($this->ESOPERADOR, 0, $letra, 0, 0);
               $this->PiezaSimple[] = $objeto;
            }
            else if ($letra == '(')
            {
               $objeto = new Pieza_Simple();
               $objeto->ConstructorPiezaSimple($this->ESPARABRE, 0, '0', 0, 0); //¿Es paréntesis que abre?
               $this->PiezaSimple[] = $objeto;
            }
            else if ($letra == ')')
            {
               $objeto = new Pieza_Simple();
               $objeto->ConstructorPiezaSimple($this->ESPARCIERRA, 0, '0', 0, 0);//¿Es paréntesis que cierra?
               $this->PiezaSimple[] = $objeto;
            }
            else if ($letra >= 'a' && $letra <= 'z') //¿Es variable o función?
            {
               /* Detecta si es una función porque tiene dos letras seguidas */
               if ($cont < $longExpresion - 1)
               {
                  $letra2 = $expresion{$cont + 1}; /* Chequea si el siguiente carácter es una letra, dado el caso es una función
*/
                  if ($letra2 >= 'a' && $letra2 <= 'z')
                  {
                     $letra3 = $expresion{$cont + 2};
                     $funcionDetectada = 1;  /* Identifica la función */
                     for ($funcion = 0; $funcion <= $this->TAMANOFUNCION; $funcion += 3)
                     {
                      if ($letra == $this->listaFunciones{$funcion}
                            && $letra2 == $this->listaFunciones{$funcion + 1}
                            && $letra3 == $this->listaFunciones{$funcion + 2})
                         break;
                      $funcionDetectada++;
                     }
                     $objeto = new Pieza_Simple();
                     $objeto->ConstructorPiezaSimple($this->ESFUNCION, $funcionDetectada, '0', 0, 0);  //Adiciona función a la
lista
                     $this->PiezaSimple[] = $objeto;
                     $cont += 3; /* Mueve tres caracteres  sin(  [s][i][n][(] */
                  }
                  else /* Es una variable, no una función */
                  {
                     $objeto = new Pieza_Simple();
                     $objeto->ConstructorPiezaSimple($this->ESVARIABLE, 0, '0', 0, ord($letra) - $this->ASCIILETRA);
                     $this->PiezaSimple[] = $objeto;
                  }
               }
               else /* Es una variable, no una función */
               {
                  $objeto = new Pieza_Simple();
                  $objeto->ConstructorPiezaSimple($this->ESVARIABLE, 0, '0', 0, ord($letra) - $this->ASCIILETRA);
                  $this->PiezaSimple[] = $objeto;
               }
             }
            }
         }
         if ($armanumero)
         {
            $objeto = new Pieza_Simple();
            $objeto->ConstructorPiezaSimple($this->ESNUMERO, 0, '0', $parteentera+$partedecimal/$divide, 0);
            $this->PiezaSimple[] = $objeto;
         }
      }

      //Toma las piezas simples y las convierte en piezas de ejecución de funciones
      //Acumula = función (operando(número/variable/acumula))
      function Generar_Piezas_Ejecucion()
      {
         $cont = sizeof($this->PiezaSimple)-1;
         $this->Contador_Acumula = 0;
         do
         {
            if ($this->PiezaSimple[$cont]->getTipo() == $this->ESPARABRE || $this->PiezaSimple[$cont]->getTipo() == $this-
>ESFUNCION)
            {
             $this->Generar_Piezas_Operador('#', '#', $cont);  //Primero evalúa los menos unarios
             $this->Generar_Piezas_Operador('^', '^', $cont);  //Luego evalúa las potencias
```

```
            $this->Generar_Piezas_Operador('*', '/', $cont);  //Luego evalúa multiplicar y dividir
            $this->Generar_Piezas_Operador('+', '-', $cont);  //Finalmente evalúa sumar y restar

            //Crea pieza de ejecución
            $objeto = new Pieza_Ejecuta();
            $objeto->ConstructorPiezaEjecuta($this->PiezaSimple[$cont]->getFuncion(),
                  $this->PiezaSimple[$cont + 1]->getTipo(), $this->PiezaSimple[$cont + 1]->getNumero(), $this->PiezaSimple[$cont +
1]->getVariable(), $this->PiezaSimple[$cont + 1]->getAcumula(),
                  '+', $this->ESNUMERO, 0, 0, 0);
            $this->PiezaEjecuta[] = $objeto;

            //La pieza pasa a ser de tipo Acumulador
            $this->PiezaSimple[$cont + 1]->setAcumula($this->Contador_Acumula++);

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

     //Toma las piezas simples y las convierte en piezas de ejecución
     //Acumula = operando(número/variable/acumula)  operador(+, -, *, /, ^)   operando(número/variable/acumula)
     function Generar_Piezas_Operador($operA, $operB, $inicio)
     {
        $cont = $inicio + 1;
        do
        {
           if ($this->PiezaSimple[$cont]->getTipo() == $this->ESOPERADOR && ($this->PiezaSimple[$cont]->getOperador() == $operA ||
$this->PiezaSimple[$cont]->getOperador() == $operB))
           {
            //Crea pieza de ejecución
            $objeto = new Pieza_Ejecuta();
            $objeto->ConstructorPiezaEjecuta(0,
                  $this->PiezaSimple[$cont - 1]->getTipo(),
                  $this->PiezaSimple[$cont - 1]->getNumero(), $this->PiezaSimple[$cont - 1]->getVariable(), $this-
>PiezaSimple[$cont - 1]->getAcumula(),
                  $this->PiezaSimple[$cont]->getOperador(),
                  $this->PiezaSimple[$cont + 1]->getTipo(),
                  $this->PiezaSimple[$cont + 1]->getNumero(), $this->PiezaSimple[$cont + 1]->getVariable(), $this-
>PiezaSimple[$cont + 1]->getAcumula());
            $this->PiezaEjecuta[] = $objeto;

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

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

            //Cambia la pieza anterior por pieza acumula
            $this->PiezaSimple[$cont]->setAcumula($this->Contador_Acumula++);
           }
           $cont++;
        } while ($cont < sizeof($this->PiezaSimple) && $this->PiezaSimple[$cont]->getTipo() != $this->ESPARCIERRA);
     }

     //Calcula la expresión convertida en piezas de ejecución
     function Calcular()
     {
        $valorA=0;
        $valorB=0;
        $totalPiezaEjecuta = sizeof($this->PiezaEjecuta);

        for ($cont = 0; $cont < $totalPiezaEjecuta; $cont++)
        {
           switch ($this->PiezaEjecuta[$cont]->getTipoOperA())
           {
            case 5: $valorA = $this->PiezaEjecuta[$cont]->getNumeroA(); break; //¿Es un número?
            case 6: $valorA = $this->VariableAlgebra[$this->PiezaEjecuta[$cont]->getVariableA()]; break;  //¿Es una variable?
            case 7: $valorA = $this->PiezaEjecuta[$this->PiezaEjecuta[$cont]->getAcumulaA()]->getValorPieza(); break; //¿Es una
expresión anterior?
           }
           if (is_nan($valorA) || is_infinite($valorA)) return $valorA;

           switch ($this->PiezaEjecuta[$cont]->getFuncion())
           {
            case 0:
               switch ($this->PiezaEjecuta[$cont]->getTipoOperB())
               {
                  case 5: $valorB = $this->PiezaEjecuta[$cont]->getNumeroB(); break; //¿Es un número?
                  case 6: $valorB = $this->VariableAlgebra[$this->PiezaEjecuta[$cont]->getVariableB()]; break;  //¿Es una
variable?
                  case 7: $valorB = $this->PiezaEjecuta[$this->PiezaEjecuta[$cont]->getAcumulaB()]->getValorPieza(); break; //¿Es
una expresión anterior?
               }
               if (is_nan($valorB) || is_infinite($valorB)) return $valorB;

               switch ($this->PiezaEjecuta[$cont]->getOperador())
               {
                  case '#': $this->PiezaEjecuta[$cont]->setValorPieza($valorA * $valorB); break;
```

```
                case '+': $this->PiezaEjecuta[$cont]->setValorPieza($valorA + $valorB); break;
                case '-': $this->PiezaEjecuta[$cont]->setValorPieza($valorA - $valorB); break;
                case '*': $this->PiezaEjecuta[$cont]->setValorPieza($valorA * $valorB); break;
                case '/': if ($valorB == 0) return NAN; else $this->PiezaEjecuta[$cont]->setValorPieza($valorA / $valorB);
break;
                case '^': $this->PiezaEjecuta[$cont]->setValorPieza(pow($valorA, $valorB)); break;
             }
            break;
          case 1:
          case 2: $this->PiezaEjecuta[$cont]->setValorPieza(sin($valorA)); break;
          case 3: $this->PiezaEjecuta[$cont]->setValorPieza(cos($valorA)); break;
          case 4: $this->PiezaEjecuta[$cont]->setValorPieza(tan($valorA)); break;
          case 5: $this->PiezaEjecuta[$cont]->setValorPieza(abs($valorA)); break;
          case 6: $this->PiezaEjecuta[$cont]->setValorPieza(asin($valorA)); break;
          case 7: $this->PiezaEjecuta[$cont]->setValorPieza(acos($valorA)); break;
          case 8: $this->PiezaEjecuta[$cont]->setValorPieza(atan($valorA)); break;
          case 9: $this->PiezaEjecuta[$cont]->setValorPieza(log($valorA)); break;
          case 10: $this->PiezaEjecuta[$cont]->setValorPieza(ceil($valorA)); break;
          case 11: $this->PiezaEjecuta[$cont]->setValorPieza(exp($valorA)); break;
          case 12: $this->PiezaEjecuta[$cont]->setValorPieza(sqrt($valorA)); break;
          case 13: $this->PiezaEjecuta[$cont]->setValorPieza(pow($valorA, 0.333333333333)); break;
        }
      }
      return $this->PiezaEjecuta[$totalPiezaEjecuta - 1]->getValorPieza();
   }

   // Da valor a las variables que tendrá la expresión algebraica
   function ValorVariable($variableAlg, $valor)
   {
      $this->VariableAlgebra[ord($variableAlg) - $this->ASCIILETRA] = $valor;
   }
}

?>
```