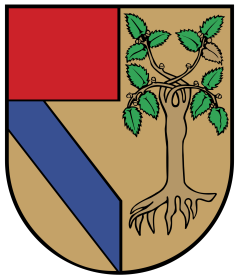
Proyecto: Intérprete Aritmético

Análisis y diseño de algoritmos

Septiembre 2023



UNIVERSIDAD
PANAMERICANA

Integrantes del equipo:

Natalia Malpica Blackaller
Leyre Anayantzin Ramírez Nieto
Alen Jerson Solis Ruíz

Profesor:

Carlos Prieto López

1 Índice

2 Introducción

Este documento pertenece al primer proyecto del semestre 1238.
Fue realizado en equipo, cuyos integrantes somos Natalia Malpica Blackeller, Leyre Anayantzin Ramírez Nieto y Alen Jerson Solís Ruíz; cada integrante del equipo aportó una tercera parte, es decir el 33.33%, del trabajo para lograr lo que sería nuestra entrega final.
Realizamos nuestro trabajo en conjunto durante horas libres y tiempo de clase que nos fue concedido por nuestro profesor, para la realización del mismo utilizamos lo aprendido en clases y las herramientas digitales que en las mismas se nos proporcionaron.
A continuación, presentaremos el desglose de nuestro trabajo.

3 Descripción y análisis del problema

Lo que se nos pide realizar es una especie de calculadora que solo recibe del usuario las operaciones de suma y de multiplicación, por lo que solo va a poder recibir los siguientes caracteres permitidos:

$$N\acute{u}meros: \ (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$$

$$Punto\ decimal\ (Ejemplo: 4.5, 2.0, 8.9, etc.)$$

$$Paréntisis\ ()$$

$$Signo\ de\ suma\ (+)$$

$$Signo\ de\ multiplicación\ (*)$$

(1)

En primera instancia tiene que determinar si la operación que le proporciona el usuario es válida.
Entonces tenemos dos casos:

1. La expresión es válida
2. La expresión no es válida porque generó un error

Caso 1:

La expresión proporcionada cumple con lo siguiente:

1. Es una cadena no vacía, significa que si contiene elementos con los que se pueda realizar alguna operación.
2. Usa sólo caracteres válidos, siendo los siguientes: números, punto (para agregar números con decimal), signo de suma (+), signo de multiplicación (*) y paréntisis.
3. En caso incluir paréntesis, tiene la cantidad de pares correctos. Es decir, no sobran ni faltan paréntisis.

En caso de cumplir con lo anterior, se va a poder evaluar la operación para obtener su resultado.

Caso 2:

La expresión proporcionada cumple con lo siguiente:

1. Es una cadena vacía, significa que no contiene elementos con los que se pueda realizar alguna operación.
2. Incluye caracteres inválidos, siendo los siguientes: letras, signos diferentes a los de suma (+) y/o multiplicación (*), paréntisis impares ó algún otro caracter que no sea de los permitidos. 1

3. En caso incluir paréntesis, no tiene la cantidad de pares correctos. Es decir, sobran o faltan paréntisis.

En caso de cumplir con lo anterior, no será posible evaluar la operación y se generará un error explicando lo ocurrido.

4 Solución propuesta

Nuestro intérprete aritmético está compuesto por 9 funciones que pueden dividirse en tres grupos. En primer lugar, está la función que permite la interacióncon el usuario, siendo la entrada de la expresión a evaluar. Después, la función principal, la cual verifica si una cadena contiene una expresión aritmética válida y de ser así, la interpreta. Finalmente, tenemos 8 funciones que auxilian a la función principal y le permiten validar distintos casos y lanzar errores de ser necesario. A continuación se detalla cada una de dichas funciones:

1. **Función marcaError():** Señala al usuario dónde se encuentra un error, en caso de que la cadena proporcionada no cumpla con las especificaciones.
2. **Función eliminarCaracteresIgnorados():** Esta función se deshace de los caracteres que no afectan la expresión, como espacios, tabuladores y saltos de línea.
3. **Función parentesisValidos():** Esta función se encarga de verificar si el uso de paréntesis en la cadena dada es adecuado. Esto implica que no se cierren paréntesis antes de abrirse y que al final de la expresión se hayan abierto exactamente la misma cantidad que cerrado.
4. **Función caracteresValidos():** Comprueba que una cadena contenga únicamente caracteres permitidos, sean dígitos, punto decimal, paréntesis, o signos de suma y multiplicación.
5. **Función numero():** Verifica que una cadena esté compuesta únicamente por dígitos y a lo más un punto decimal. Es decir, que represente a un número en su expresión decimal.
6. **Función subexpresionBase():** Verifica que las expresiones base contengan un número, el cual puede estar precedido por '(' o seguido de ')'.
7. **Función subexpresionesNoVacias():** Esta función va a ir partiendo la operación cada vez que haya un signo (+) o (*), además va a ir checando que si haya subexpresiones válidas.
8. **Función interpreteAritmetico():** Esta función va a ir llamando a las funciones anteriores para checar que cumpla con lo solicitado y así, poder analizarlos como expresiones matemáticas.
9. **Función entrada():** Guarda la expresión dada por usuario en una cadena que va ser analizadda por las demás funciones para verificar que sea una expresión matemática válida.

5 Algoritmo

En esta sección se presenta el desglose de lo que sería el algoritmo que sigue cada una de nuestras funciones.

5.1 Función marcaError()

Dada una cadena y la posición del error, concatena una nueva cadena donde se señala.

1. Inicio.
2. Recibe una cadena y la posición del caracter que contiene un error guardada en la variable caracterEnLinea.
3. Crea una cadena vacía llamada nuevaCadena.
4. Agrega todos la caracteres previos al error a la cadena nueva.
5. Concatena <, el caracter erróneo y > a la nueva cadena.
6. Concatena uno a uno a la nueva cadena todos los caracteres posteriores al erróneo en la cadena original.
7. Regresa la cadena nueva.
8. Termina la función.

5.2 Función eliminarCaracteresIgnorados()

En esta función eliminamos los caractéres que no afectan la validez de la cadena ni afectan el valor de la expresión.

1. Inicio.
2. Recibe una cadena.
3. Crea una lista de los caracteres ignorados: espacio (""), tabulador ("\t")$ysaltodelínea$("\n"), $Creaunanue$

4. Para cada caracter a en la cadena:
    (a) Crea un booleano noIgnorado = 1 (más adelante, 0 representará que va a ser ignorado y eliminado, mientras que 1 significa que permanecerá en la cadena).
    (b) Para todo caracter b en la lista de caracteres ignorados:
        i. Si a == b, entonces noIgnorados = 0 (el caracter a es un de los caracteres que vamos a eliminar).
    (c) Si noIgnorado == 1, entonces, concatena el caracter a al final de la nueva cadena (de lo contrario no se agrega a la cadena nueva).
5. Regresa a la nueva cadena (igual a la original sin los caracteres ignorados).
6. Fin.

5.3 Función parentesisValidos()

Esta función se encarga de verificar si el uso de paréntesis en la cadena dada es adecuado. Esto implica que no se cierren paréntesis antes de abrirse y que al final de la expresión se hayan abierto exactamente la misma cantidad que cerrado.

1. Inicio.
2. Recibe una cadena de texto.
3. Crea una variable c=0 que nos permitirá llevar la cuenta de los paréntesis abiertos y cerrados.
4. Crea una variable caracterEnLinea = 1, que guardará la posición actual en la cadena (con ello podremos indicar dónde hay errores de ser así).
5. Para cada caracter a en la cadena:
   (a) Si a == ‘(‘, entonces suma 1 a c.
   (b) Si a == ‘)‘, entonces resta 1 a c.
   (c) Si c < 0, esto significa que se han cerrado más paréntesis de los que se han abierto, así que devuelve un error y termina la función.
   (d) Suma 1 a caracterEnLinea.
6. Si c es distinto de cero después de haber recorrido toda la cadena, significa que no todos los paréntesis que se abrieron fueron cerrados, por lo tanto lanza un error y termina la función.
7. Termina la función.

5.4 Función caracteresValidos()

En esta función verificamos que la cadena no contenca ningún caracter que el intérprete aritmético no acepte.

1. Inicio.
2. Recibe una cadena.
3. Crea una lista con los caracteres válidos: ’0’, ’1’, ’2’, ’3’, ’4’, ’5’, ’6’, ’7’, ’8’, ’9’, ’.’, ’(’, ’)’, ’*’, ’+’
4. Crea una variable caracterEnLinea = 1, que guardará la posición actual en la cadena (con ello podremos indicar dónde hay errores de ser así).
5. Para cada caracter a en la cadena:
   (a) Para cada caracter b en la lista de caracteres válidos:
      i. Si a == b, pasa a la siguiente iteración de a.

(b) Si se recorrió toda la lista de caracteres válidos y a no coincidió con ninguno, entonces lanza un error y termina la función.

(c) Suma 1 a caracterEnLinea.

6. Termina la función.

5.5 Función numero()

Verifica si una cadena es únicamente un número. Para ello debe estar conformada por dígitos y a lo más un punto decimal.

1. Inicio.
2. Recibe:
   (a) cadena: una cadena de texto.
   (b) num: un fragmento de la cadena que queremos verificar que sea un número.
   (c) inicionumero: La posición del primer caracter de num con respecto a la cadena.
3. Crea puntos = 0, un contador de la cantidad de puntos en num (no puede tener más de un punto decimal).
4. Crea una variable tam que guarde el tamaño de num.
5. Crea una lista digitos que contiene los caracteres válidos para un número (dígitos y punto decimal): '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '.'
6. Sea i = 1 un iterador para recorrer los caracteres de num.
7. Mientras i <= tam:
   (a) Si num[i] == '.' :
      i. Suma 1 a puntos.
      ii. Si puntos > 1, lanza un error y termina la función (pues un número no puede contener más de un punto decimal).
   (b) De lo contrario:
      i. Sea j = 1 un iterador para recorrer dígitos.
      ii. Mientras j <= length(digitos):
         A. Si digitos[j] == num[i], entonces asigna j = length(digitos) + 1 (con ello j es mayor que length(digitos) y se detiene el ciclo).
         B. Si j == length(digitos) y num[i] != digitos[j], esto significa que se llegó al final del ciclo y num[i] no coincide con ninguno de los caracteres válidos. Por lo tanto, lanza un error y termina la función.

C. Suma 1 a j para pasar a la siguiente iteración.

iii. Suma 1 a i para pasar a la siguiente iteración.

(c) Si, tam == 1 y puntos == 1, esto significa que num == "." lo cual no es un número válido. Por lo tanto, lanza un error y termina la función.

8. Termina la función.

5.6 Función subexpresionBase()

Las expresiones base deben contener exactamente un número. Antes del número pueden haber paréntesis abiertos y después, paréntesis cerrados.

1. Inicio.
2. Recibe:
   (a) cadena: una cadena de texto.
   (b) subexpresion1: un fragmento de la cadena que queremos verificar que sea una subexpresión base.
   (c) inicioSubexpresión: La posición del primer caracter de subexpresion1 con respecto a la cadena.
3. Crea la variable inicioNumero = 0 (guardará la posición en que los caracteres de la subexpresión dejen de ser '(' y deba comenzar el número).
4. Si length(subexpresion1) == 0, esto significa que que está vacía (y por ello no es una subexpresión base). Por ello, lanzar un error y terminar la función.
5. Sea i = 1 un iterador para recorrer la subexpresión.
6. Mientras i <= length(subexpresion1):
   (a) Si subexpresion1[i] == '(':
      i. Si i == length(subexpresion1), esto significa que desde el inicio de la subexpresión, hasta el final todos los caracteres son '('. Por lo tanto, la subexpresión no contiene ningún número y no es una subexpresión base, así que lanza un error y termina la función.
      ii. Si subexpresion1[i] == ')', entonces se cierra un paréntesis antes del número, lo cual no está permitido, así que lanza un error y termina la función.
      iii. De lo contrario, si subexpresion1[i] no es un paréntesis, debe ser parte del número. Por lo tanto, inicioNumero = i y termina el ciclo.
   (b) Suma 1 a i para pasar a la siguiente iteración.

7. Crea una variable finNumero = inicioNumero (el numero debe contener al menos un dígito, por lo que finNumero es mayor o igual que inicioNumero).
8. Mientras finNumero < length(subexpresion1) el número puede contener otro caracter:
   (a) Si subexpresion1[finNumero + 1] != ')', entonces finNumero = finNumero + 1 (después del número no puede seguir nada distinto a paréntesis que cierran).
   (b) De lo contrario, termina el ciclo (si el siguiente caracter es ‘)’, entonces ya no pertenece al número).
9. Si finNumero < length(subexpresion1), entonces hay caracteres en la subexpresión posteriores al número y todos deben ser ‘)’ para que sea una expresión válida:
   (a) Sea i = finNumero + 1 un iterador para recorrer los caracteres restantes de la subexpresión.
   (b) Mientras i <= length(subexpresion1):
      i. Si if subexpresion1[i] != ')', lanza un error y termina la función, pues ‘)’ es el único caracter válido después de un número.
      ii. Suma 1 a i para pasar a la siguiente iteración.
10. Asigna candidatoNumero = subexpresion1[inicioNumero:finNumero].
11. Si numero(candidatoNumero, (inicioNumero + inicioSubexpresion - 1), cadena) no regresa un error, entonces candidatoNumero y subexpresion1 es una subexpresión base válida.
12. Termina la función.

5.7 Función subexpresionesNoVacias()

Esta función “fragmenta” la cadena con los caracteres ‘+’ y ‘*’. Además, verifica que que entre cada par de los caracteres anteriores, haya una subexpresión base válida (de lo contrario dicom que hay una vacía entre ellos).

1. Inicio.
2. Recibe una cadena.
3. Crea:
   (a) tam = length(cadena).
   (b) posicion = 1 (nos permitirá saber en qué caracter de la cadena nos encontramos.
   (c) subexpresion1 = “” (una cadena vacía).

(d) inicioNuevaSubexpresion = 1 (nos iniciará dónde comienza subexpresión que fragmentaremos de la cadena).

4. Mientras posicion <= tam:

   (a) Si cadena[posicion] == '+':

      i. Si posicion == 1 o posicion == tam, regresa un error y termina la función, pues una expresión válida no puede comenzar ni terminar con el caracter '+'.

      ii. De lo contrario subexpresion1 = cadena[inicioNuevaSubexpresion: posicion - 1] (del inicio de la nueva subexpresión que tenía guardado al caracter previo al '+' que "partió" la expresión".

      iii. Corre la función subexpresionBase(subexpresion1, inicioNuevaSubexpresion, cadena) para verificar que subexpresion1 sea una subexpresión base válida.

      iv. La siguiente subexpresion a la acabada de verificar comienza un caracter después del '+' que "fragmentó" la expresión, entonces inicioNuevaSubexpresion = posicion + 1.

   (b) Si cadena[posicion] == '*':

      i. Si posicion == 1 o posicion == tam, regresa un error y termina la función, pues una expresión válida no puede comenzar ni terminar con el caracter '*'.

      ii. De lo contrario subexpresion1 = cadena[inicioNuevaSubexpresion: posicion - 1] (del inicio de la nueva subexpresión que tenía guardado al caracter previo al '*' que "partió" la expresión.

      iii. Corre la función subexpresionBase(subexpresion1, inicioNuevaSubexpresion, cadena) para verificar que subexpresion1 sea una subexpresión base válida.

      iv. La siguiente subexpresion a la acabada de verificar comienza un caracter después del '*' que "fragmentó" la expresión, entonces inicioNuevaSubexpresion = posicion + 1.

   (c) Suma 1 a posicion para pasar a la siguiente iteración.

5. Termina la función.

5.8 Función interpreteAritmetico()

Interpreta a los valores de la cadena como expresiones matemáticas si cumplen los requisitos de la función.

1. Inicio.
2. Recibe una cadena.
3. Limpia la cadena con la función eliminarCaracteresIgnorados(cadena) y almacena el resultado en cadena.

4. Verifica que los paréntesis de la cadena con parentesisValidos(cadena).

5. caracteresValidos(cadena) verifica que todos que la cadena contenga únicamente los caracteres aceptados.

6. subexpresionesNoVacías(cadena) "parte" la cadena y verifica que cada fragmento contenga una subexpresión base válida.

7. Si ninguna de la funciones anteriores lanzó un error, entonces la cadena contiene una expresión matemática válida:

   (a) La transforma en un float y almacenarlo en resultado con las siguientes funciones: float(eval(Meta.parse(cadena)))

   (b) Regresa resultado.

8. Terminar la función.

5.9 Función entrada()

1. Inicio

2. Crea cadena = "" (una cadena vacía).

3. Lee una línea de texto dada por el usuario y la almacena en linea.

4. Mientras que length(linea) > 0 (mientras que el usuario no ingrese una línea vacía):

   (a) cadena = cadena * linea (concatena la nueva linea con la cadena)

   (b) vuelve a leer una línea del usuario y la guarda en cadena.

5. Regresa cadena.

6. Termina la función.

5.10 Archivo "interprete_aritmetico"

1. Inicio.

2. Indica al usuario las instrucciones y le piede una expresión válida.

3. Lee la expresión del usuario mediante entrada() y la almacena en cadena.

4. Verifica que sea una expresión válida a través de intérpreteAritmético(cadena):

   (a) Si la expresión es válida la almacena en resultado y la imprime.

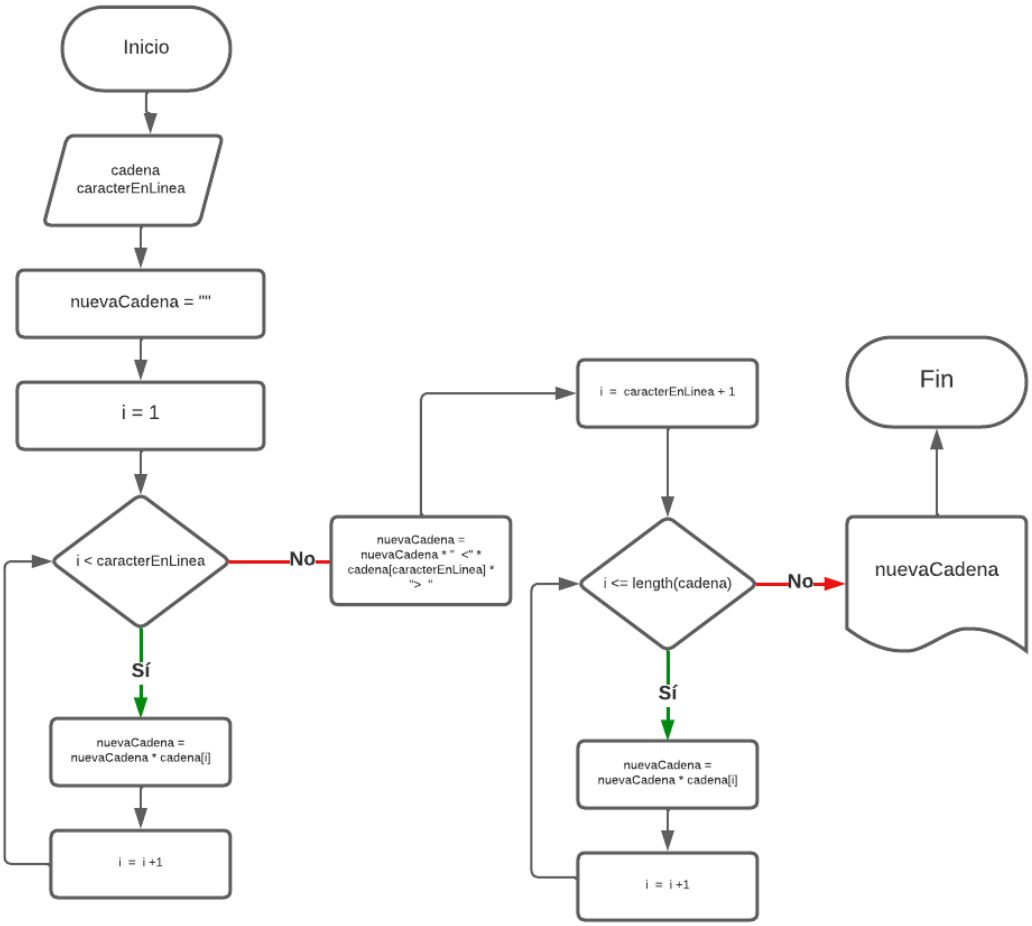
   (b) De lo contrario, regresa un error.

5. Fin.

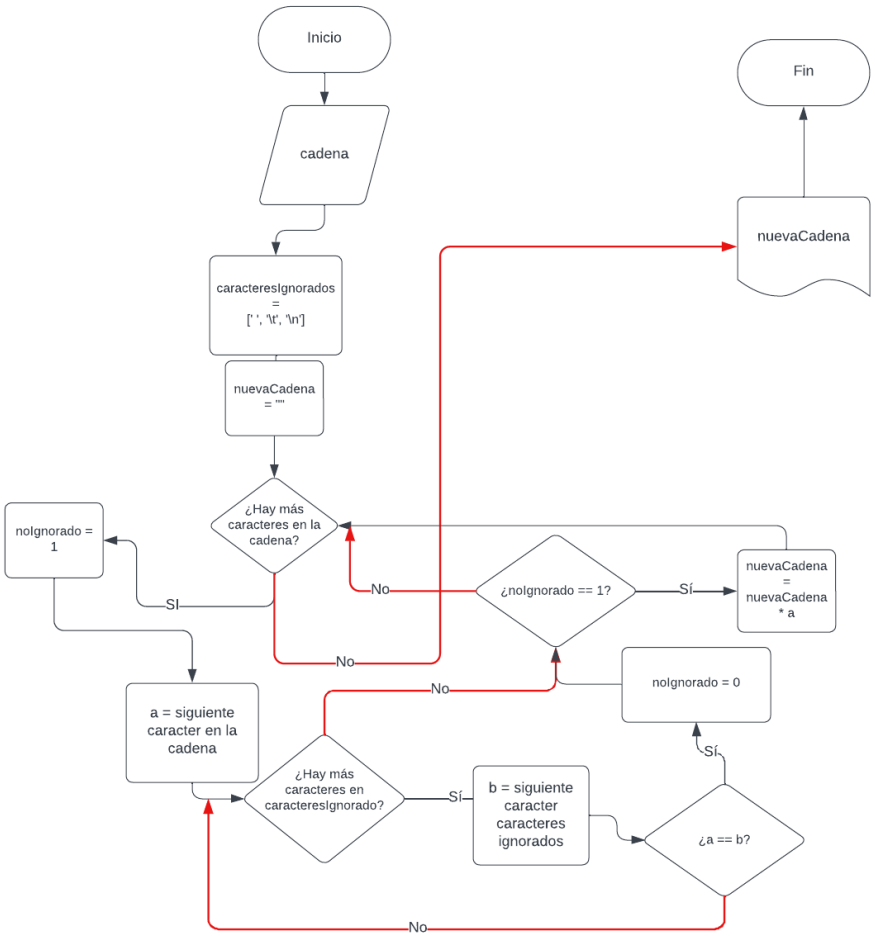
6 Diagramas de flujo

A continuación presentamos los respectivos diagramas de flujo de las funciones que realizamos. Para una mejor comprensión realizamos un diagrama por cada función y en el que corresponde a la *Función interpreteAritmetico()* se llama a cada una de las funciones. La *Función entrada()* es la que va a recibir la operación para poder ejecutar el intérprete aritmético.

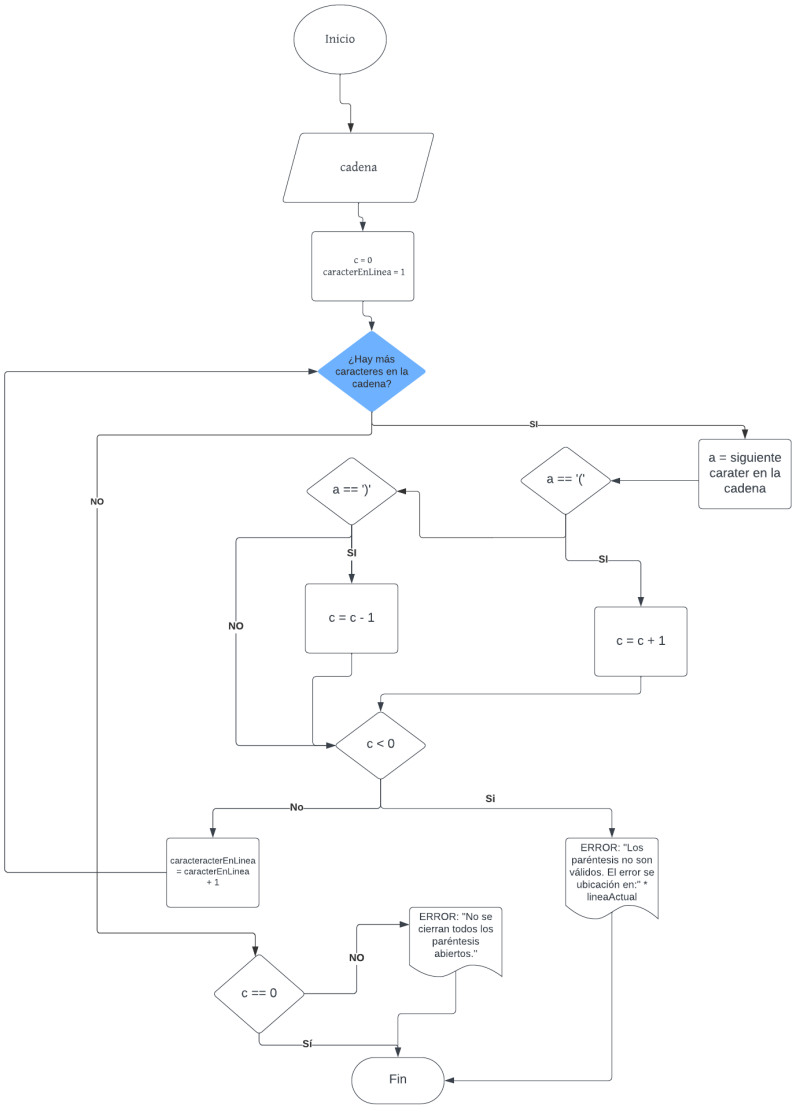
6.1 Función marcaError()



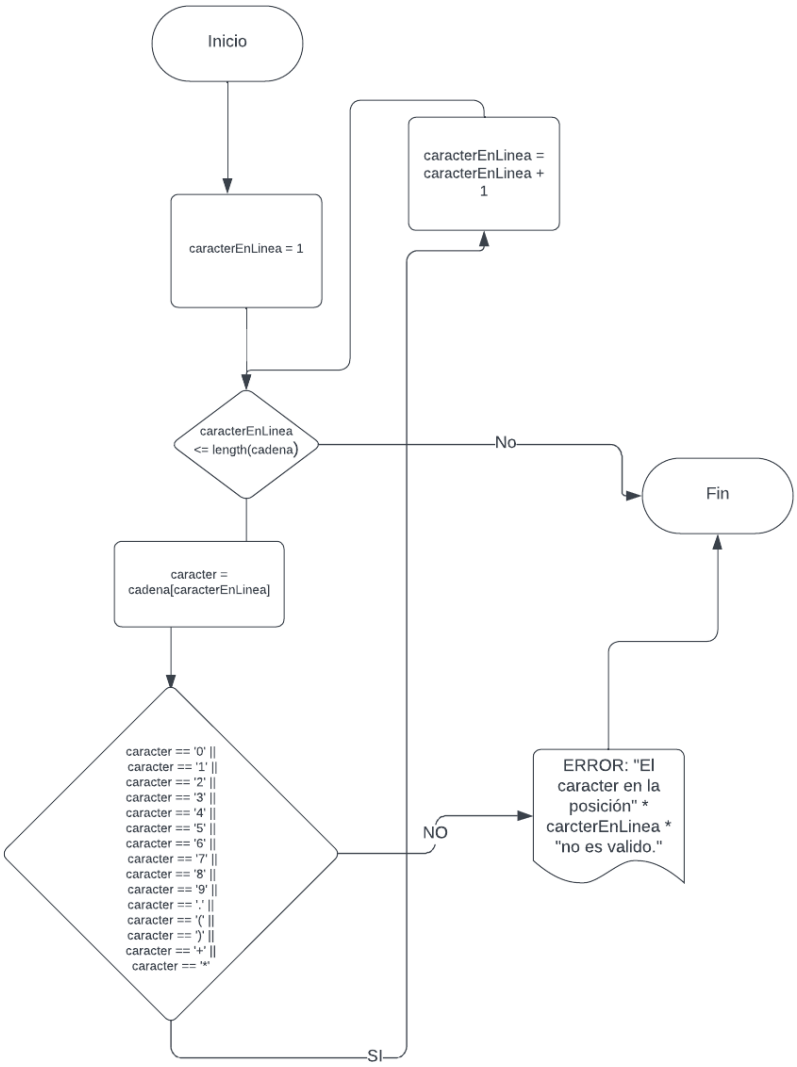
6.2 Función eliminarCaracteresIgnorados()



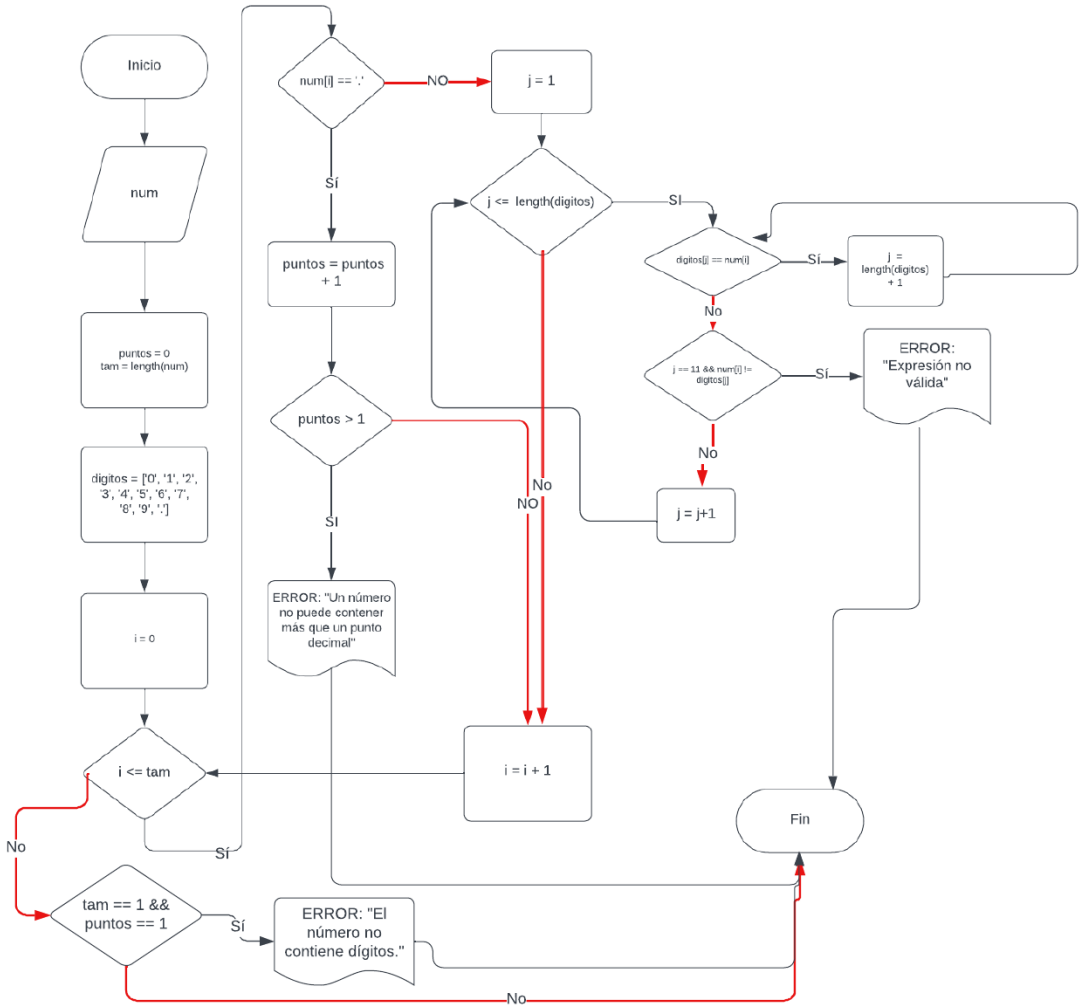
6.3 Función parentesisValidos()



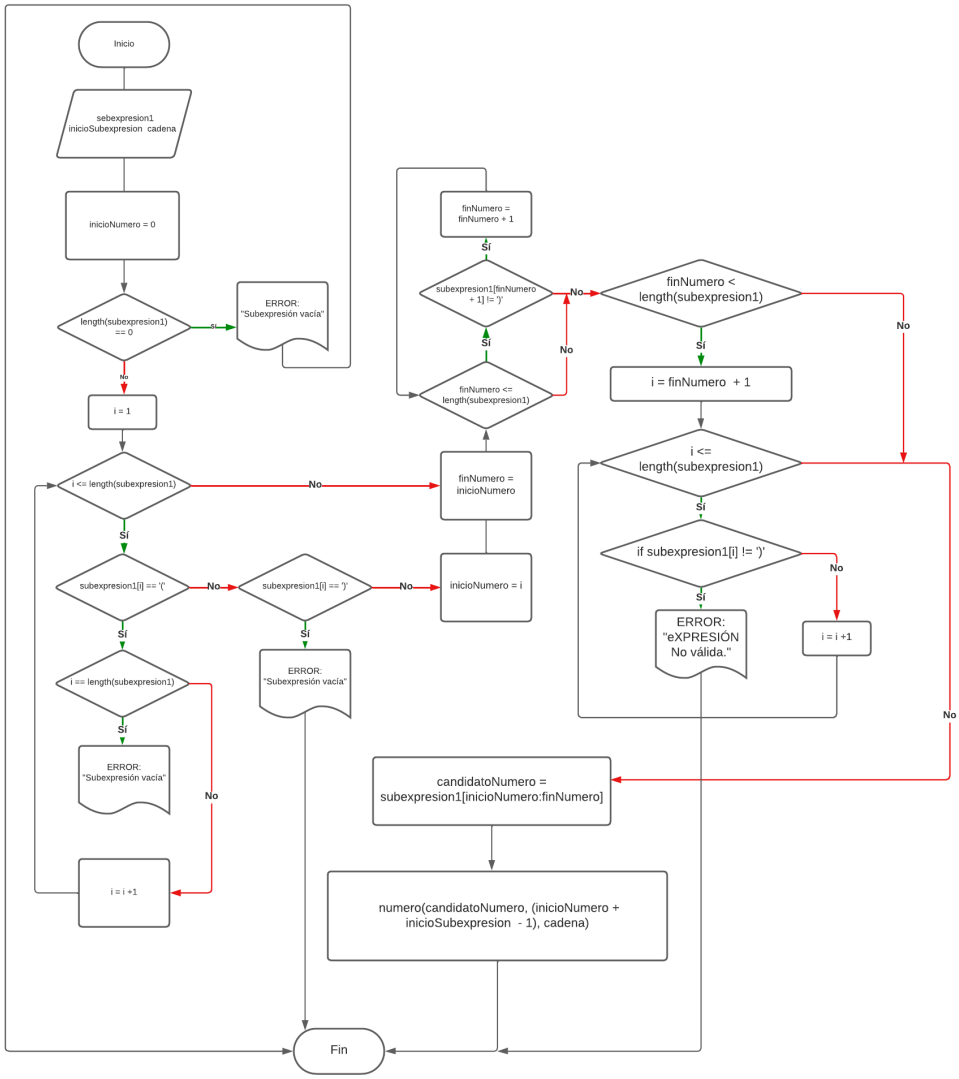
6.4 Función caracteresValidos()



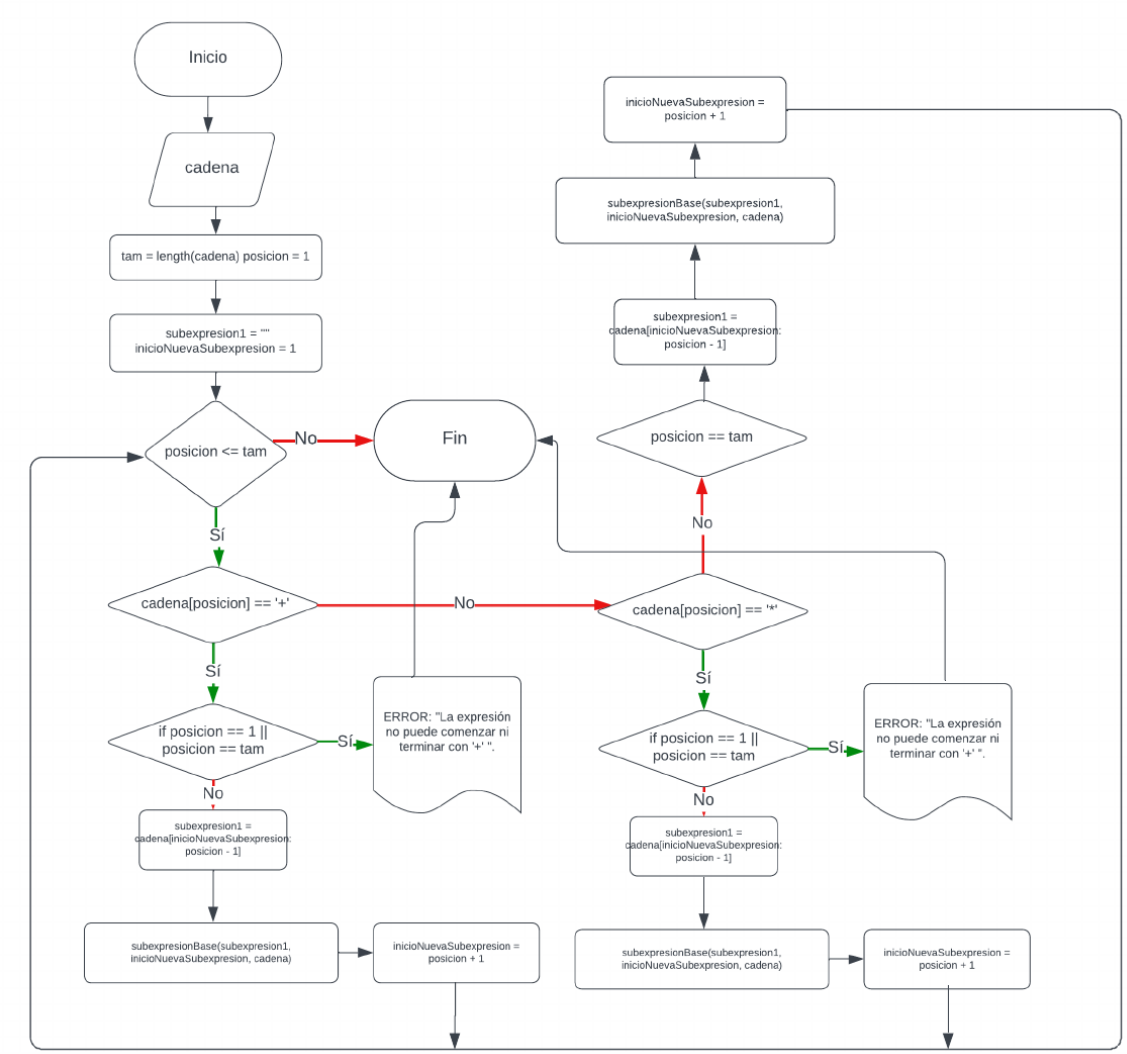
6.5 Función numero()



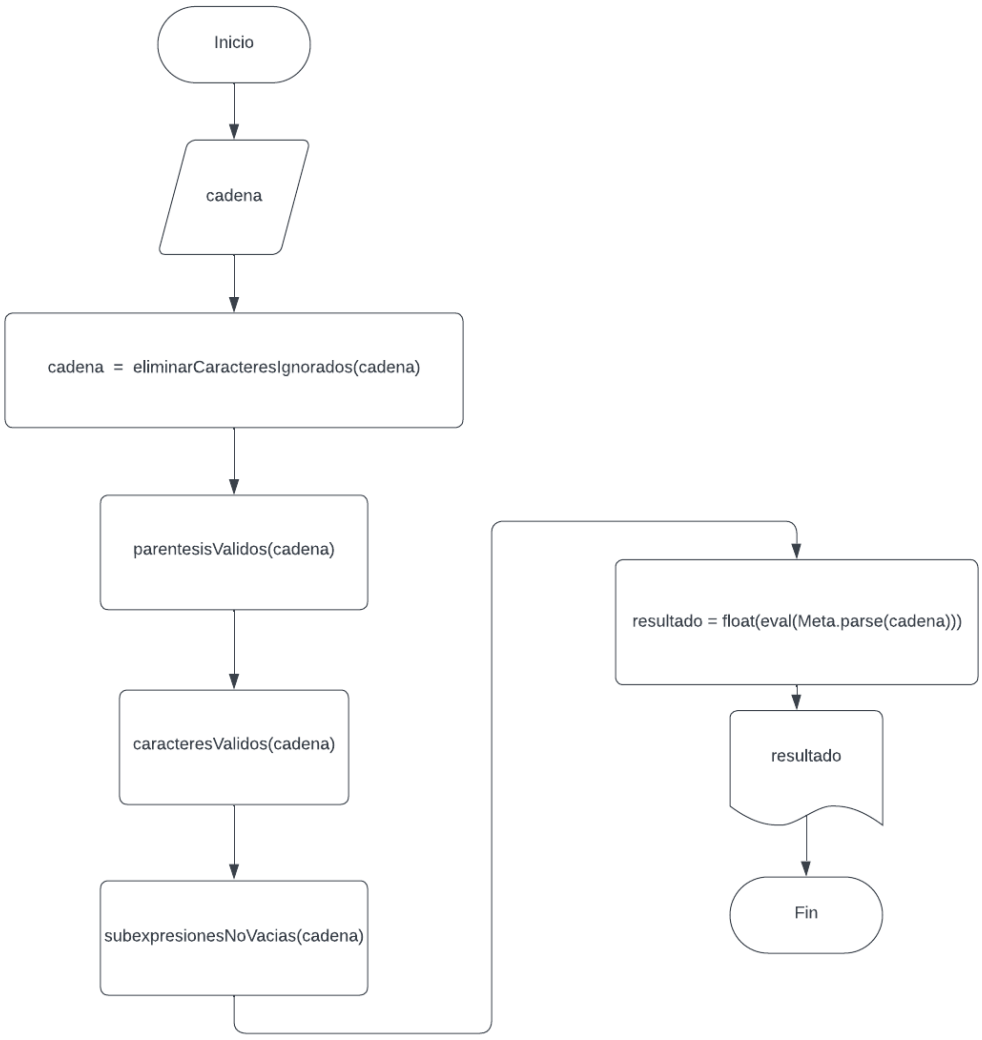
6.6 Función subexpresionBase()



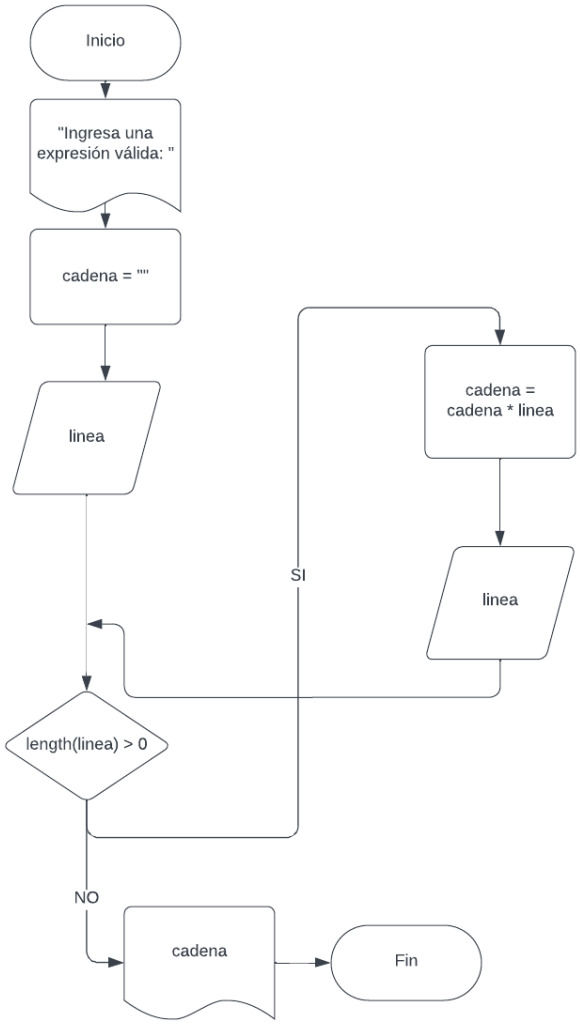
6.7 Función subexpresionesNoVacias()



6.8 Función interpreteAritmético()



6.9 Función entrada()



7 Instrucciones para utilizar el programa

Para la utilización de nuestro programa se tienen que realizar los siguientes pasos:

1. Descargar desde GitHub los archivos en la carpeta de nuestro equipo, dichos archivos van a llevar por nombre:
   - interprete_aritmetico-funciones.jl
   - interprete_aritmetico.jl
2. Una vez descargados, se va a ingresar a julia en alguna aplicación de programación o bien en la máquina virtual.
3. Tenemos que incluir el archivo interprete_aritmetico.jl a julia.
4. Una vez que ya esta incluido automáticamente se va a iniciar el programa, donde se te mencionará qué es y los requisitos que tiene que cumplir la expresión que quieras analizar.
5. Finalmente, eres libre de usar nuestro intérprete aritmético para sacar el valor de alguna operación con suma o multiplicación .

CONSIDERACIONES IMPORTANTES:

- Se debe llamar la función en cada ocasión que se quiera usar.
- Para mandar a analizar la operación deseada se debe dar doble click, de lo contrario se considera que el usuario quiere ingresar otra línea.
- En caso de haber error se va a marcar entre "<" y ">".
- Si el resultado es entero, lo devuelve con terminación .0.