Trabajo Final

Integrantes: Fabio Torrico FAI-1927
Joaquin Arias FAI-1629
Gonzalo Guerrero FAI-891
Correos:fabio.torrico@est.fi.uncoma.edu.ar
gonzalo.guerrero@est.fi.uncoma.edu.ar
joaquin.arias@est.fi.uncoma.edu.ar

Trabajo a realizar:

- Desarrollar un parser que tome la cadena de texto y convierta la cadena en alguna estructura que permita ser evaluada para un valor de x determinado
- Un módulo que permita evaluar f(x) para uno o más valores de x. El módulo debe ser capaz de:
  - Evaluar f(x) para un valor solo de X
  - Evaluar f(x) para un intervalo [a, b] con N puntos
- Un módulo visual que grafique f(x) para un intervalo [a, b] con N puntos.

Lenguaje utilizado: Smalltalk.

Materia: Programación Orientada a Objetos.

12 de Octubre 2020

Método utilizado:Shunting yard
Explicación:

El algoritmo shunting yard es un método para analizar (parsing) las ecuaciones matemáticas especificadas en la notación de infijo. Puede ser utilizado para producir la salida en la notación polaca inversa o como árbol de sintaxis abstracta.

El algoritmo en detalle:
Mientras haya tokens a ser leídos:
Lea un token.
Si el token es un número, entonces agregué a la cola de salida.
Si el token es un token de función, entonces póngalo (push) sobre la pila.
Si el token es un separador de un argumento de función (ej., una coma):
Hasta que el token en el tope de la pila sea un paréntesis abierto, retire (pop) de la pila a los operadores y póngalos en la cola de salida. Si no es encontrado ningún paréntesis abierto, el separador fue colocado mal o los paréntesis fueron mal emparejados.
Si el token es un operador, o1, entonces:
mientras que haya un operador, o2, en el tope de la pila (esto excluye el paréntesis abierto), y
o1 es asociativo izquierdo y su precedencia es menor que (una precedencia más baja) o igual a la de o2, ó
o1 es asociativo derecho y su precedencia es menor que (una precedencia más baja) que la de o2,
retire (pop) de la pila el o2, y póngalo en la cola de salida;
ponga (push) o1 en el tope de la pila.
Si el token es un paréntesis abierto, entonces póngalo en la pila.
Si el token es un paréntesis derecho:
Hasta que el token en el tope de la pila sea un paréntesis abierto, retire (pop) a los operadores de la pila y colóquelos en la cola de salida.
Retire (pop) el paréntesis abierto de la pila, pero no lo ponga en la cola de salida.
Si el token en el tope de la pila es un token de función, póngalo (pop) en la cola de salida.

Si la pila se termina sin encontrar un paréntesis abierto, entonces hay paréntesis sin pareja.
Cuando no hay más tokens para leer:
Mientras todavía haya tokens de operadores en la pila:
Si el token del operador en el tope de la pila es un paréntesis, entonces hay paréntesis sin la pareja correspondiente.
Retire (pop) al operador y póngalo en la cola de salida.
Fin.
Definición:https://es.wikipedia.org/wiki/Algoritmo_shunting_yard#:~:text=El%20algoritmo%20shunting%20yard%20es,de%20sintaxis%20abstracta%20(AST).

Breve explicación y aplicación en nuestro programa:

Clase Graficadora:

Variables de instancia: capa1 origen colValores resultadoX inicio final graficoDePuntos miGraficador buttonClose buttonSubmit puntoX buttonSubmit2 inputFormula inputInicio inputFinal X Y formula inputX.

Métodos:

dibujarFondo:
Método encargado para diseñar objetos de tipo morph. Diseñamos el fondo donde se encuentran los campos a rellenar para la ecuación, la variable X, los valores de los intervalos, el lienzo donde vamos a dibujar lo/los punto/s al reemplazar la variable dentro de la ecuación, botón de cerrado, ejes X Y, y los botones para graficar.

dibujarPunto:
Método encargado principalmente en crear un objeto tipo Morph darle la forma de un punto y colocarlo en el lugar exacto del lienzo donde la variable x fue reemplazada en la función.

dibujarPuntos:
Método encargado principalmente en crear objetos de tipo Morph darle la forma de un punto y colocarlo en los lugares exactos del lienzo donde la variable x fue reemplazada(varias veces con distinto x) en la función.

cerrar:
Cierra el programa Graficadora.

update:
Método encargado de actualizar el lienzo donde graficamos puntos o intervalos.

setParametroPunto:
Guarda las variables obtenidas del morph y las guarda en las variables de instancia de la graficadora luego envía las variables graficoModelo.
setParametros:
Guarda las variables obtenidas del morph y las guarda en las variables de instancia de la graficadora luego envía las variables graficoModelo.

Clase GraficoModelo:

Variables de instancia: Función parser colRes cadena inicio final variableX resultadoX.

Métodos:

inicializarGraficoModelo:
Inicialización de las variables que utilizan los métodos.

obtenerPunto: con:
Recibe las variables de la clase graficadora y las almacena en las variables de instancia de gráficoModelo, este método es para cuando necesitamos calcular un solo punto.

obtenerResultados: desde: hasta:
Recibe las variables de la clase graficadora y las almacena en las variables de instancia de gráfico modelo"

ResetearX:
Sirve para resetear la variable x después de calcular un punto de esa forma si por error volviera a calcular no lo dejaría porque no es un número.

getColValores:
Retorna la colección de resultado de x reemplazada y calculada en la función a la clase Graficadora porque los necesita para poder realizar los gráficos.

getResultadoX:
Retorna el resultado de x reemplazada y calculada en la función a la clase Graficadora porque lo necesita para poder realizar el gráfico del punto.

resolver:
En este método se encarga de enviar una sola vez la cadena a el método armarArbol de la clase ParserTPFinal y retorna la cadena en notación infija luego crea una colección donde almacena todos los resultados calculados en el intervalo.

resolverPunto:
En este método se encarga de enviar una sola vez la cadena a el método armarArbol de la clase ParserTPFinal y retorna la cadena en notación infija luego calcula el resultado para un solo X.

Clase ParserTPFinal:

Variables de instancia:cadena arreglo.

Métodos:

agregarDigito: de: en:
Este método se encarga de recibir una cadena donde concatenamos números/coma decimal que se encuentran en forma de string y luego retorna esa cadena.

armarArbol:
Recibe la función del método parsear que se encuentra en la misma clase en forma de colección ordenada en forma infija, luego comienza a leer cada posición y dependiendo de lo que encuentra colocara valores dentro de una pila, luego el método se encarga de llamar a los métodos para crear nodo (armarTipoBinario y armarTipoUnario), estos se van agrupando (como si

fuera un árbol) hasta crear un nodo que contiene a la raíz de toda la ecuación y termina el método devolviendo ese nodo.

armarTipoUnario:conTermino1:
Este método recibe un operador y 1 término, dado el tipo de operador me crea un Nodo con dicha operación Unaria correspondiente (cos , sen , tan , sqrt , !) para luego retornar el nodo recientemente creado.

armarTipobinario:conTermino1:conTermino2:
Este método recibe un operador y 2 términos, dado el tipo de operador me crea un Nodo con dicha operación Binaria correspondiente (+ , - , *, / ,^) para luego retornar el nodo recientemente creado.

este:MayorQue:
Este método se encarga principalmente de chequear el orden de jerarquía dependiendo de la respuesta si es true o false le avisara al método llamador (parsear) quien tiene mayor peso si el valor de la pila que recibe por parámetro o si el elemento de la cadena.

inicializarParser:
En este método inicializamos el arreglo (variable de instancia) con los operadores cada operador esta en una posición del arreglo y esta representa el orden de la jerarquía.

parsear:
Este método se encarga de preparar la cola de elementos para luego ser usada por **armarArbol:**
El método comienza llamando al **separarPorSimbolos:** una vez ya obtenida la colección que nos devuelve el mensaje recientemente llamado.
Procedemos a recorrer la colección, que equivale directamente a recorrer la función ingresada. Mientras recorremos la función procedemos a apilar elementos o insertarlos en la cola dependiendo lo que el método **este:mayorQue:** (método encargado de revisar jerarquía de operadores) siguiendo el **Algoritmo de Shunting yard** el cual al final obtenemos una cola con los elementos de la función ingresada que será retornada**.**

separarPorSimbolos:

Este método se encarga de recibir una cadena que será la función ingresada por el usuario y el método separa por números, símbolos agregandolo a una colección de forma tal que quede igual pero separada en Celdas diferentes y retornando dicha colección.

Clase: Termino(clase padre de Binario y Unario)

variable de instancia:termino1 expresión pila

Métodos:

iniciarTermino:
Inicializa la clase Termino guardando la cadena en una variable y crea una colección vacía.

Evaluar:
Método padre polimórfico de evaluar, mediante el subclassResponsability, entonces esto delegan a sus únicos dos hijos (Unario y Binario) que a su vez delegan la responsabilidad a sus respectivos hijos para poder aplicar los métodos evaluar concretos.

Aclaración:self subclassResponsibility . Esto se conoce como un “método marcador” e indica que las subclases tienen la responsabilidad de definir una versión concreta del método. Los métodos self subclassResponsibility siempre deben anularse y, por lo tanto, nunca deben ejecutarse. Si olvida anular uno y se ejecuta, se generará una excepción.

Clase: Binario(sub clase de Termino)

variable de instancia:termino2

Métodos:

evaluar:

Delega la responsabilidad a los evaluar de las clases hijo de Binario (División Multiplicación, suma, resta).

Clase:División (sub clase de Binario)

variable de instancia:
Métodos:
evaluar:
Calcula la división entre dos términos, llama a los métodos evaluar para cada término, y luego retorna el resultado.

Clase:Potencia (sub clase de Binario)

variable de instancia:
Métodos:
evaluar:
Calcula la división entre dos términos, llama a los métodos evaluar para cada término, y luego retorna el resultado.

Clase:Multiplicación (sub clase de Binario)

variable de instancia:
Métodos:
evaluar:
Calcula la multiplicación entre dos términos, llama a los métodos evaluar para cada término, y luego retorna el resultado.

Clase:Resta (sub clase de Binario)

variable de instancia:
Métodos:
evaluar:

Calcula la resta entre dos términos, llama a los métodos evaluar para cada término, y luego retorna el resultado.

Clase:Suma (sub clase de Binario)

variable de instancia:
Métodos:
evaluar:
Calcula la división entre dos términos, llama a los métodos evaluar para cada término, y luego retorna el resultado.

Clase: Unario(sub clase de Término)

variable de instancia:
Métodos:
evaluar:
Delega la responsabilidad a los evaluar de las clases hijo de Binario (Coseno, Seno, Tangente, RaizCuadrada, Numero, Factorial).

Clase:Coseno (sub clase de Unario)

variable de instancia:
Métodos:
evaluar:
Calcula el coseno de un término, llama al método evaluar el término1, y luego retorna el resultado.

Clase:Factorial (sub clase de Unario)

variable de instancia:
Métodos:
evaluar:
Calcula el factorial de un término, llama al método evaluar el término1, y luego retorna el resultado.

Clase:Numero (sub clase de Unario)

variable de instancia:
Métodos:
evaluar:
Verifica si lo que entra es un número y si este es un número lo retorna, en caso de ser ‘x’ retorna la variable ‘x’ enviada por parámetro.

Clase:RaizCuadrada (sub clase de Unario)

variable de instancia:
Métodos:
evaluar:
Calcula la raiz cuadrada de un término, llama al método evaluar el término1, y luego retorna el resultado.

Clase:Tangente (sub clase de Binario)

variable de instancia:
Métodos:
evaluar:
Calcula la tangente de un término, llama al método evaluar el término1, y luego retorna el resultado.

Clase:Seno (sub clase de Binario)

variable de instancia:
Métodos:
evaluar:

Calcula el seno de un término, llama al método evaluar el término1, y luego retorna el resultado.

Constructores:

crearGraficoModelo:
Crea el objeto graficoModelo, lo inicializa y luego lo retorna.

crearParser:
Crea el objeto ParserTPFinal, lo inicializa y luego lo retorna.

crearTermino:
Crea el objeto Termino, lo inicializa y luego lo retorna.

crearBinario: y:
Crea el objeto de tipo Binario, lo inicializa y luego lo retorna.

(Para división, resta, multiplicacion, suma y potencia el metodo clase crearBinario: y: , son polimorfico).

crearUnario:
Crea el objeto de tipo Unario, lo inicializa y luego lo retorna.

(Para Coseno, Factorial, Número, raizCuadrada, Seno y Tangente el metodo clase crearUnario: son polimorfico).