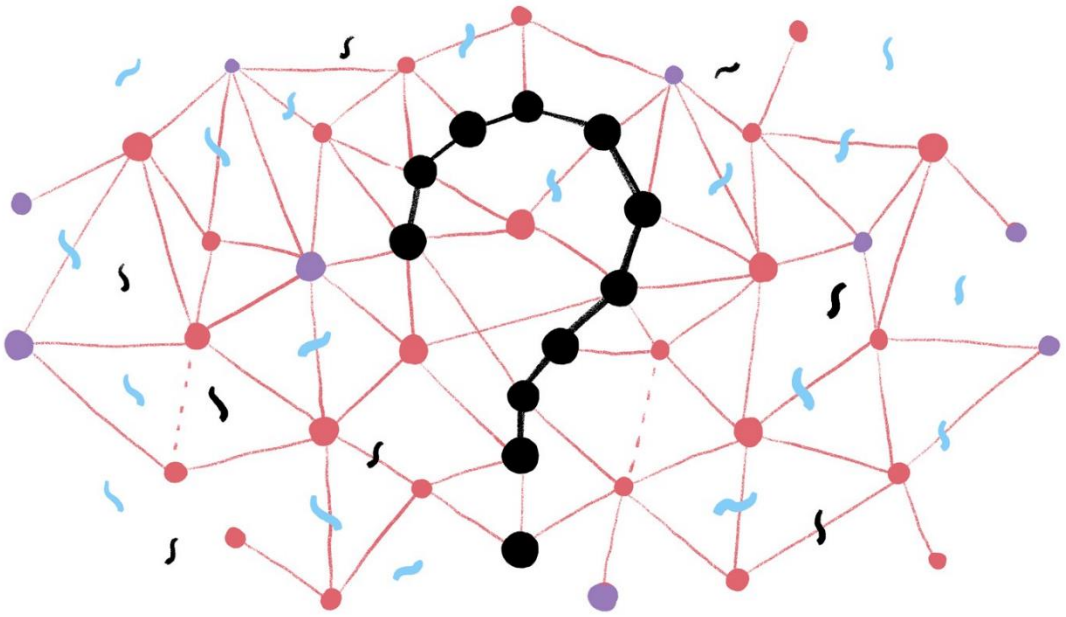
Tecnológico de Monterrey

Matemáticas Computacionales
MC. Luis Ricardo Peña Llamas

Proyecto de Segundo Parcial:

Analizador sintáctico de aritmética simple



17/05/2021

Héctor Rodolfo Álvarez Dávalos A01636166

Analizador sintáctico de aritmética simple

Con el objetivo de aplicar los conocimientos obtenidos en la clase de Matemáticas Computacionales, se nos asignaron dos proyectos, de los cuales yo escogí el analizador de gramáticas aritméticas. El proyecto consiste en un algoritmo que genera una GLC, que representa a los paréntesis bien balanceados y las cadenas aritméticamente coherentes, y determina que cadenas pertenecen a ella. Una vez se tiene la GLC, se recibe un string como entrada, que representa una cadena, se determina si pertenece a las posibles cadenas que se pueden generar en la gramática, si la cadena es aceptada, se muestra el árbol de derivación y el resultado de la operación que representa la cadena.

El algoritmo consiste en 3 clases distintas, una clase de estructura llamada Node, la clase encargada de la impresión del árbol de derivación, llamada BTreePrinter y la clase principal donde se define la GLC y se evalúa la entrada, llamada GrammarAnalyzer.

La clase nodo es muy simple, representa los nodos de un árbol binario y tiene 3 atributos, dos llamados left y right que se encargan de representar a los nodos vecinos y uno llamado data que representa el valor del nodo; la clase solo cuenta con métodos constructores y setters. La clase encargada de imprimir el árbol binario, se encarga de recorrer los nodos que lo conforman, asegurarse que la estructura este bien e imprimir adecuadamente por niveles. Finalmente, la clase principal, GrammarAnalyzer, se encarga de realizar las comparaciones, crear el árbol binario, manejar toda la información de entrada, correr el programa, crear la GLC y definir si se acepta la cadena.

Diagrama de flujo:

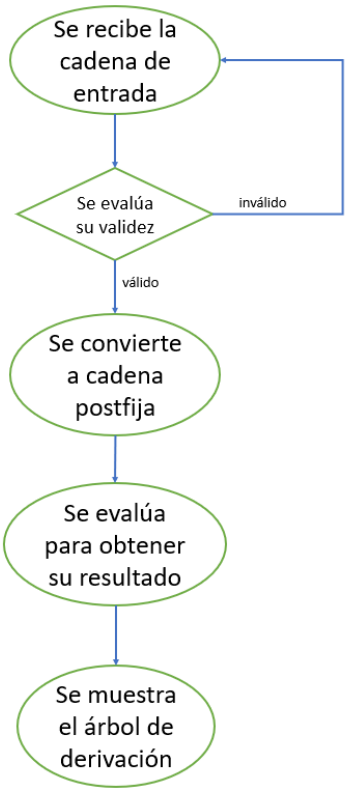
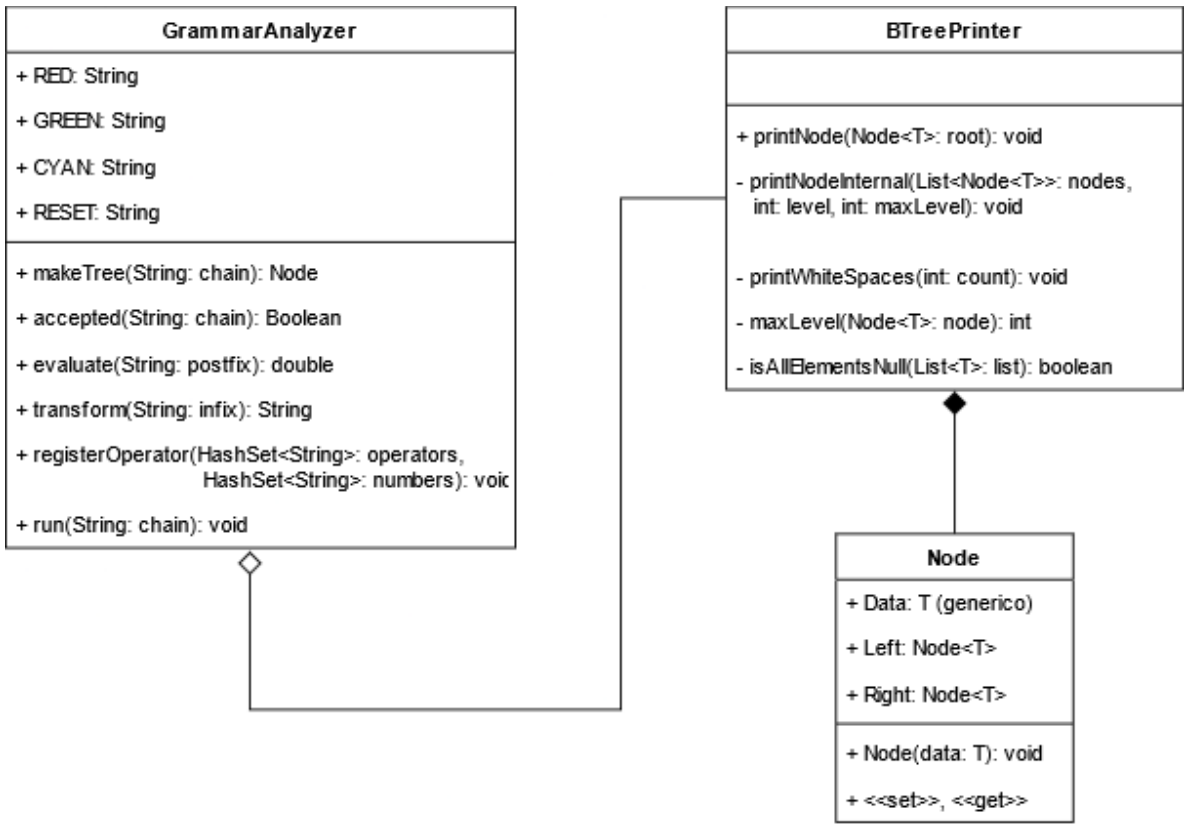


Diagrama de clases:



Métodos principales:

accepted(String chain): boolean

El método se encarga de manejar la entrada, comenzando por separarla en un arreglo de strings por carácter; después se recorre totalmente revisando cada elemento izquierdo y derecho de la cadena. Si el elemento izquierdo es un operador, se revisa si el elemento a la izquierda es un numero o un paréntesis de cierre, si es un número de o paréntesis de apertura, en caso de que no sea alguno de estos elementos, se declara como no válida y se retorna falso.

Si elemento es un paréntesis abierto, se revisa que el elemento izquierdo sea un paréntesis abierto, un operador o número; si no es uno de estos elementos se retorna falso. Si la pila se encuentra vacía o el elemento en la cima es un paréntesis de apertura, se añade a la pila el elemento; de lo contrario la cadena se declara no válida y se retorna falso.

Si el elemento es una cerradura, se evalúa que el elemento de la izquierda sea un paréntesis abierto, un número o un operador, además de que el elemento de la derecha sea un operador u otro paréntesis de cerradura; de lo contrario se retornará falso. Además, se saca el elemento de la cima de la pila en caso de que sea un paréntesis abierto, si la pila está vacía se retorna falso.

Si es un número, se revisa que el elemento a la izquierda sea un número, un paréntesis de apertura o un operador, también se revisa que el elemento derecho sea un número, una cerradura o un operador; de lo contrario, se regresa falso. Después de que se realicen todas las evaluaciones si la pila está vacía se retorna falso, de lo contrario se retorna verdadero.

transform(String infix): String

Este método se encarga de transformar la entrada, una cadena infija, a una cadena postfija. Para esto se obtiene cada carácter en la expresión, si se evalúa que no es un operador, se pasa directamente a postfija.

Si el símbolo es un operador y la pila está vacía, se agrega directamente a ella. Cuando la pila no está vacía se evalúa la prioridad de operación, en caso de tener

prioridad sobre el elemento sobre la pila, se mete sobre la pila se vuelve al inicio del método; de lo contrario se saca la cima de la pila y se pasa a postfija, repitiendo la evaluación de operandos.

Si se trata de un paréntesis, se saca el elemento superior de la pila y se pasa a la postfija, si la nueva cima es un paréntesis de apertura, se le hace pop, de lo contrario se vuelve a la evaluación de paréntesis. El entero se repite hasta haber leído todos los elementos, si al final quedan elementos en la pila se agregan a la postfija y se retorna.

evaluate(String postfix): int

Este método se encarga de evaluar la expresión postfija y todos sus elementos, se agregar los operandos a su pila y obtener el resultado de la expresión. Si el elemento evaluado se trata de un operando se mete a la pila, si se trata de un operador, se le designa al programa, después se sacan los dos elementos superiores de la pila asignándolos a variables, después se realiza la operación con el operador designado y las dos variables, finalmente se mete el resultado a la pila. El proceso se repite hasta acabar con los elementos, dejando el resultado total en la cima de la pila.

makeTree(String cadena): void

Este método se encarga de crear el árbol de derivación. Se evalua cada elemento de la cadena, si el elemento es un número, se crea un objeto tipo nodo con él y se agrega a una pila de nodos. Si el elemento es un operar, se crea un nuevo nodo con el elemento, se saca la cima de pila y se asigna como el nodo izquierdo de este nuevo nodo. Finalmente, se saca la cima de la pila y se le define como la raíz del árbol.

Ejecución:

Para correr el programa es necesario compilar y correr la clase principal, GrammarAnalyzer, cuando se encuentre en ejecución, la termina se verá de la siguiente manera:

```
PS C:\Users\renfe\Desktop\A01636166_HectorAlvarez> javac .\GrammarAnalyzer.java
Note: .\GrammarAnalyzer.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
PS C:\Users\renfe\Desktop\A01636166_HectorAlvarez> java GrammarAnalyzer
WELCOME!
Please, enter a gramatic expression:
```

Después, el programa estará listo para recibir una gramática, una vez se haya registrado se podrá dar enter.

```
PS C:\Users\renfe\Desktop\A01636166_HectorAlvarez> java GrammarAnalyzer
WELCOME!
Please, enter a gramatic expression:
(5+6+(8/2))
```

Si la gramática está bien escrita, el programa mostrará al usuario un mensaje informándole que la entrada es válida, el resultado de la operación representada por la expresión y el árbol de derivación resultante. A continuación, se presentan algunos ejemplos:

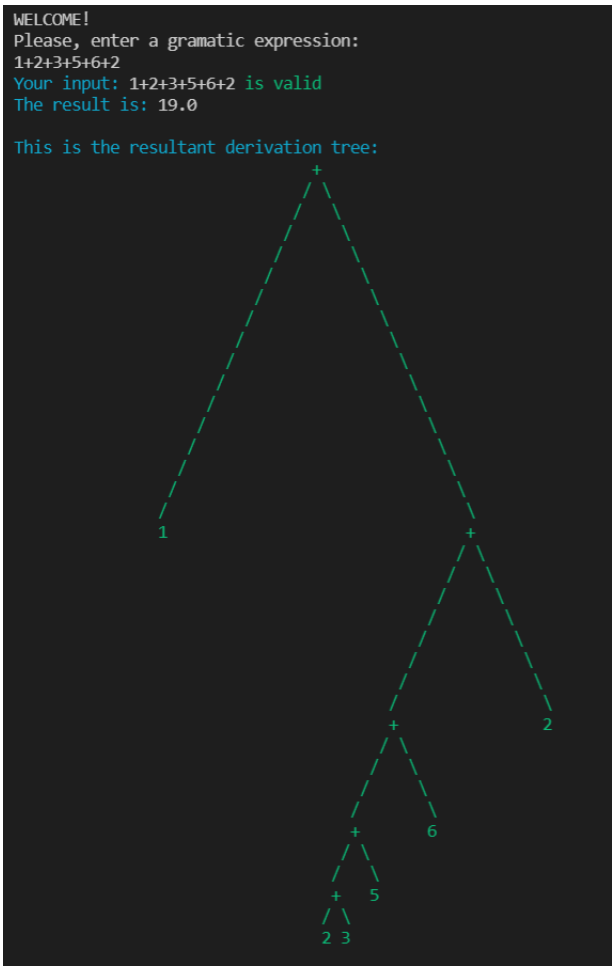
```
PS C:\Users\renfe\Desktop\A01636166_HectorAlvarez> java GrammarAnalyze
WELCOME!
Please, enter a gramatic expression:
(5+6+(8/2))
Your input: (5+6+(8/2)) is valid
The result is: 15.0

This is the resultant derivation tree:
      +
     / \
    /   \
   /     \
  /       \
 5         +
          / \
         /   \
        6     /
             / \
            8 2
```

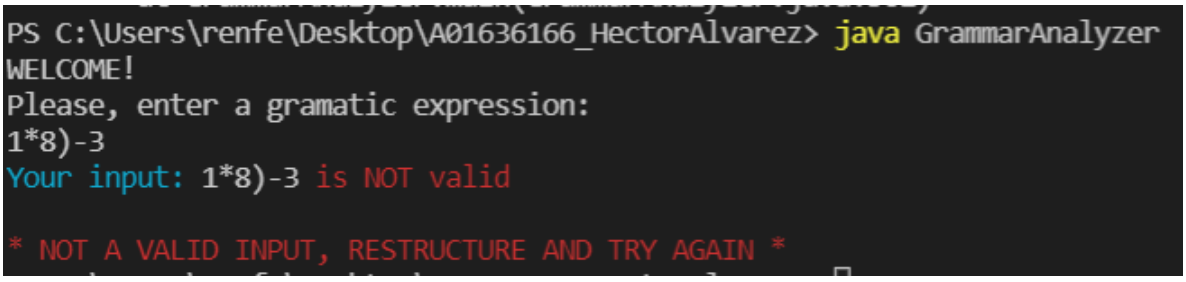
```
PS C:\Users\renfe\Desktop\A01636166_HectorAlvarez> java GrammarAnalyzer
WELCOME!
Please, enter a gramatic expression:
((1*8)-3)
Your input: ((1*8)-3) is valid
The result is: 5.0

This is the resultant derivation tree:
    -
   / \
  /   \
 1     *
      / \
     8 3


PS C:\Users\renfe\Desktop\A01636166_HectorAlvarez>
```



Finalmente, en caso de no ser una cadena aceptada, se informa al usuario, a continuación, algunos ejemplos:



```
PS C:\Users\renfe\Desktop\A01636166_HectorAlvarez> java GrammarAnalyzer
WELCOME!
Please, enter a gramatic expression:
1++8/2
Your input: 1++8/2 is NOT valid

* NOT A VALID INPUT, RESTRUCTURE AND TRY AGAIN *
```

```
PS C:\Users\renfe\Desktop\A01636166_HectorAlvarez> java GrammarAnalyzer
WELCOME!
Please, enter a gramatic expression:
1***2
Your input: 1***2 is NOT valid

* NOT A VALID INPUT, RESTRUCTURE AND TRY AGAIN *
```