Descripción General de Proyecto

Evaluador de Expresiones

Clase: *Lenguajes de Programación*

Docente: *Ing. Jose Isaac Orellana Velasquez*

Semestre | Periodo | Año: *1 | 2 | 2019*

Parcial: *I*

ÍNDICE

1 Información del Proyecto

1.1 Datos

Clase	**Lenguajes de Programación**
Proyecto	Evaluador de Expresiones
Asignación:	Individual
Fecha de presentación	Jue, 20 Junio, 2019

2 Propósito y justificación del proyecto

Este proyecto tiene como objetivo principal poner en práctica los conocimientos adquiridos durante la clase y entender la lógica a la programación a un bajo nivel.

3 Descripción del proyecto y Entregables

Se requiere desarrollar en C++ un evaluador de expresiones, en el cual se pueda ingresar una expresión por el usuario, se puedan aplicar validaciones, realizar la evaluación y mostrar el resultado de la expresión.

En línea de comando se podrá ingresar una expresión y deberá poderse evaluar e imprimir el resultado.

La expresión será ingresada en notación infija, por ejemplo: 1+2*3 o (1+2)*3. El evaluador deberá convertir la expresión a notación postfija porque de esta forma se hace más sencilla su evaluación.

Se deben aplicar ciertas validaciones en el caso cuando la expresión no sean correctas, por ejemplo: 10+5+. Recordemos, que no siempre se debe considerar como correcto lo que ingresa el usuario.

Posibles errores para validar: Se termine con un operador la expresión infija, los paréntesis o corchetes no se cierren correctamente, existan caracteres diferentes a los permitidos.

El evaluador debe funcionar para números enteros y flotantes.

Operadores	Tipos
(), []	Paréntesis, Corchetes
^	Potencia
*, /, %	Multiplicación, División, Modulo
+, -	Suma, Resta

4 Expresiones Postfijas:

Las operaciones postfijas buscan resolver los mismos problemas de las expresiones infijas, pero atacan el problema de otra manera. En estas expresiones, no existen los paréntesis y los operados y operandos se representa de forma distinta, por ejemplo:

10 + 2 => 10, 2, +

10 / 6 => 10, 6, /

10 + 12 * 12 => 10, 12, 12, *, +

10 + (1 + 2) => 10, 1, 2, +, +

En las expresiones Postfijas, la ecuación de evalúa de forma lineal, pues los operadores y operandos están ordenados de tal forma que ya no es necesario tener en cuenta la precedencia de los operadores.

Como lo comenté anteriormente, los compiladores realizan la conversión de expresiones infijas a postfijas, para aumentar la efectividad en tiempo de ejecución.

Para realizar la conversión de infijas a postfijas es necesario utilizar una Pila y seguir las siguientes reglas:

- Operador = precedencia > se cambia
- Operador > precedencia > se agrega a la pila
- Operador < precedencia > sacar operador de la pila
- Paréntesis derecho > vaciar pila

No te preocupes si no quedan claras las reglas, las analizaremos con el ejemplo 10 + (1 + 2) * 2.

1. Lo primero será tomar el 10 de la expresión y pintarla en el resultado: Resultado => 10
2. Luego, tenemos el operados + el cual deberemos agregar a la pila: pila => **[+]**
3. Luego, tenemos un paréntesis izquierdo, lo agregamos a la pila: pila => **[** + , (**]**
4. Seguido, tenemos el número 1 y lo pasamos directo al resultado: Resultado => 10, 1
5. El siguiente operador es +, el cual es almacenado en la pila: pila => **[** + , (, +**]**
6. El siguiente número en aparecer es el 2, el cual pasa directo al resultado: resultado => 10, 1, 2
7. El siguiente valor es un paréntesis derecho, por lo que tendremos que vaciar la pila hasta el siguiente paréntesis izquierdo. Por lo que sacamos de la pila el operador +, dejando el resultado de la siguiente manera: Resultado => 10, 1, 2, + y la pila: **[** + **]**
8. El siguiente valor es el operador *, el cual al ser de mayor precedencia que +, entra en la pila: pila => **[** +, * **]**

9. Nuevamente aparece el operando 2, el cual pasa directo al resultado: resultado => 10, 1, 2, +, 2
10. En este punto hemos terminado de evaluar la expresión, por lo que solo resta vaciar toda la pila: resultado => 10, 1, 2, +, 2, *, + y la pila => **[]**

Resultado final: 10, 1, 2, +, 2, *, +

5 Evaluación de expresiones postfijas:

Una vez que tenemos la expresión postfija generada, será necesario evaluarla, por lo que el procedimiento será algo similar al de la generación, ya que nos apoyaremos de una pila para evaluar.

En este caso, cada número que encontremos en la expresión deberá ser introducido en la pila, y cuando se encuentre un operador, deberemos sacar los dos últimos valores de la pila y aplicarles el operador. El resultado deberá ser introducido nuevamente en la pila.

Evaluando la expresión: 10, 1, 2, +, 2, *, +

1. El valor 10 es encontrado y puesto en la pila: pila => **[**10**]**
2. El valor 1 es encontrado y puesto en la pila: pila => **[**10, 1**]**
3. El valor 2 es encontrado y puesto en la pila: pila => **[**10, 1, 2**]**
4. El operador + es encontrado, por lo que se suma los valores 1 + 2 = 3, y la pila queda de la siguiente manera: **[**10, 3**]**
5. El valor 2 es encontrado y puesto en la pila: **[**10, 3, 2**]**
6. El operador * es encontrado y se multiplica 3 * 2 = 6, dejando la pila: **[**10, 6**]**
7. El operador + es encontrado y se suma 10 + 6 = 16.

6 Rubrica de evaluación

Característica	Puntos Oro	Excelente (100%)	Notable (75%)	Suficiente (50%)	Insuficiente (25%)	No Desarrollado (0%)
Permite evaluar expresiones - **De números enteros** - **De números flotantes** - **Con operadores (),[],^,*,/,%,+,-** - **Con prioridades de los operadores** - **Evaluación postfija**	**11**	Permite evaluar expresiones - De números enteros - De números flotantes - Con operadores (),[],^,*,/,%,+,- - Con prioridades de los operadores - Evaluación postfija	Permite evaluar expresiones - De números enteros - Con operadores (),[],^,*,/,%,+,- - Con prioridades de los operadores - Evaluación postfija	Permite evaluar expresiones, pero no todos los operadores - De números enteros - Con prioridades de los operadores - Evaluación postfija	Permite evaluar expresiones, pero no todos los operadores, ni la prioridad de operadores - De números enteros - Evaluación postfija	No permite evaluar expresiones
Implementa el uso de TDD	**4**	Implemente TDD				No implementa TDD
Permite validar la expresión infija - **Error de paréntesis incompleto** - **Error de corchetes incompleto** - **Caracteres no permitidos** - **Expresión infija incorrecta**	**4**	Implementa 4 validaciones a la expresión infija	Implementa 3 validaciones a la expresión infija	Implementa 2 validaciones a la expresión infija	Implementa 1 validación a la expresión infija	No implementa validaciones
Uso de features V11 C++	**3**	Implementa 4 features	Implementa 3 features	Implementa 2 features	Implementa 1 feature	No implementa features
Uso de features V14 C++	**3**	Implementa 4 features	Implementa 3 features	Implementa 2 features	Implementa 1 feature	No implementa features
Uso de features V17 C++	**3**	Implementa 4 features	Implementa 3 features	Implementa 2 features	Implementa 1 feature	No implementa features
Total	**28**					