Presentacion Proyecto : HULK

Kevin Marquez Vega

October 12, 2023

Indice

Introduccion

Para el segundo proyecto de programacion, se nos encomendo implementar un interprete de un subconjunto del lenguaje de programacion *HULK*.

Dicho subconjunto se basa principalmente en expresiones en una unica linea entre las que encontramos: expresiones aritmeticas, booleanas, funciones, asignaciones de variables, etc.

Indice

Acerca del lenguaje

Operadores Aritmeticos

El programa soporta diversas expresiones aritmeticas como por ejemplo: $(3 + 5 \times 2)\%3$

Contiene los siguientes operadores :

- Suma (+)
- Resta (-)
- Division (/)
- Multiplicacion(*)
- Resto (%)
- Exponenciacion (^)

Acerca del lenguaje

Operadores Logicos

Contiene los siguientes operadores logicos :

- And (&)
- Doble And (&&)
- Or (|)
- Doble Or (||)
- Comparativos (<, >, <=, >=, ==, !=)

Acerca del lenguaje

Expresiones if-else

Tambien contiene expresiones if-else las cuales tienen la siguiente estructura:

if(condicion) expresion1 else expresion2

Las cuales en dependencia de si se cumple o no la condicion, ejecutan la expresion1 o la expresion2.

(Toda expresion if debe contar con un else)

Acerca del lenguaje

Strings

En HULK las cadenas de texto se representan mediante el uso de comillas, por ejemplo : "hola mundo".

Posee el operador de concatenacion(@) el cual convierte en string el valor resultante de las dos expresiones que relaciona y uniendolos uno a continuacion del otro.

// Input : $(4 + 3)$ @ " enanitos."
// Output : "7 enanitos."

Las declaraciones de variables se realizan utilizando la expresion let-in

$$\text{let x} = 5 \text{ , y} = 2 \text{ in expresion ;}$$

- Para declarar varias variables en el mismo let, se separaran por comas.
- Pueden anidarse varias expresiones let-in
- Si en expresiones let-in anidadas se le asignan varios valores a la misma variable se sobreescribira quedando el ultimo que se le asigno.

$$\text{let x} = 5 \text{ in let x} = 2 \text{ in x ;} \qquad \text{// Ouput : 2}$$

En *HULK* es posible declarar funciones utilizando la palabra clave *function*

$$\text{function Suc}(x) => \quad x + 1 \ ;$$

- Es posible declarar funciones recursivas.
- En esta version he implementado la sobrecarga de funciones(es posible declarar funciones con el mismo nombre si difieren en la cantidad de parametros)
- Declarar una funcion con el mismo nombre y cantidad de parametros que otra existente produce un error.

HULK cuenta con el siguiente grupo de funciones predefinidas

- print(x)
- sin(x)
- cos(x)
- tan(x)

Acerca del lenguaje

Funcionalidades Extra

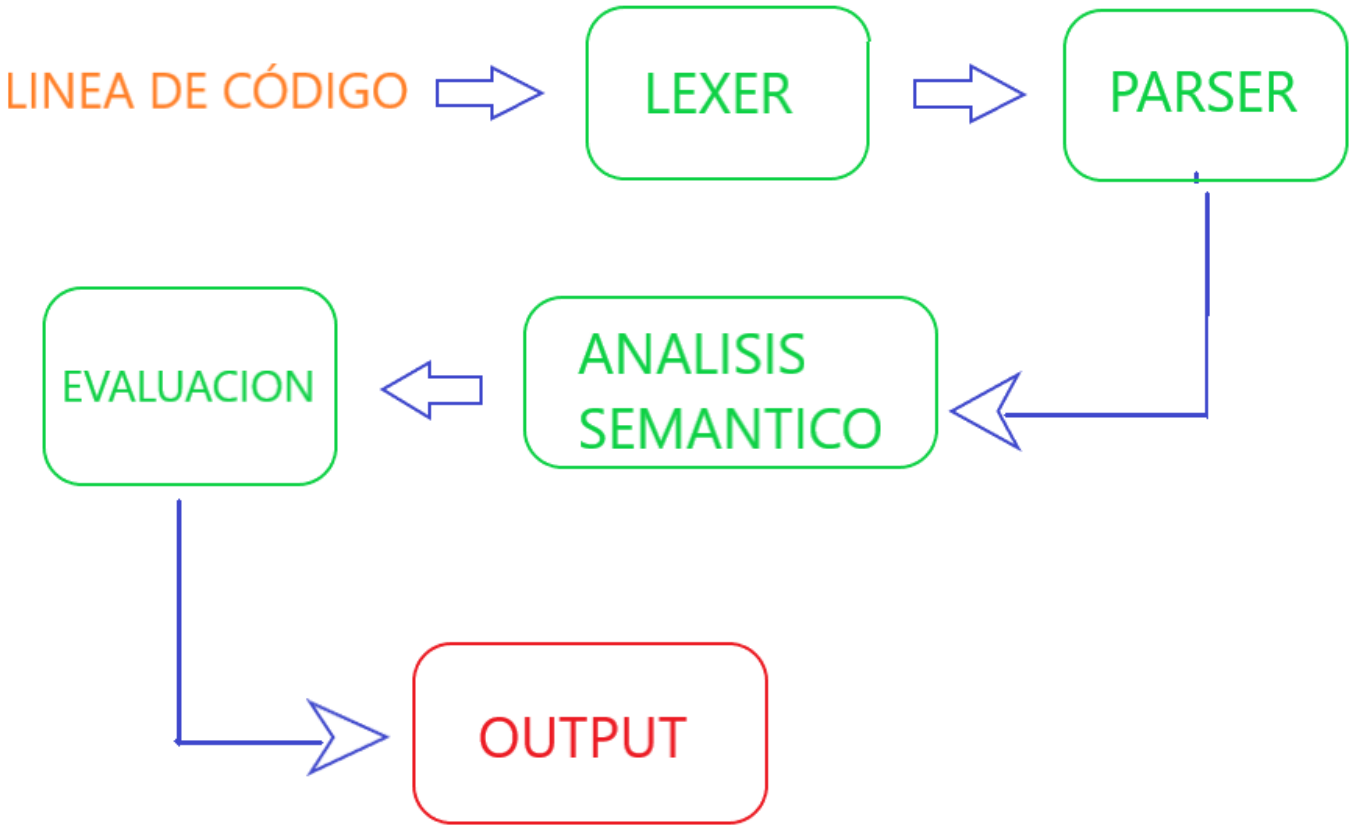
En esta version de HULK he incluido algunas funcionalidades extra

- Sobrecarga de funciones
- Comando #clear para limpiar la consola
- Comando #functions para mostrar la lista de funciones existentes.
- Mostrar el tiempo de ejecucion del programa
- Operadores logicos dobles
- Expresiones else-if

Indice

Como Funciona?

Flujo del Programa



Como Funciona?

CompilatorTools y FunctionPool

Dentro de la estructura del programa cuento con 2 clases auxiliares : CompilatorTools y FunctionPool.

CompilatorTools : es una clase abstracta que contiene metodos necesarios durante la ejecucion del programa, asi como una lista de strings llamada Bugs para almacenar los mensajes de error.

FunctionPool : Contiene una lista donde se guardan las funciones declaradas hasta el momento y metodos como chequear si una funcion existe.

Como Funciona?

Punto de Entrada

```
while(true){
    Console.Write(">") ;
    string line = Console.ReadLine() ;
    if(string.IsNullOrWhiteSpace)
    {
        return ;
    }
    var parser = new Parser(line) ;
}
```

El programa continuara pidiendo al usuario que introduzca una linea de codigo hasta que entre una linea vacia.
El constructor del *Parser* se encarga de llamar al *Lexer* para obtener la lista de tokens.

Como Funciona?

Lexer

Lexer : Recibe un string representando la linea de codigo y se encarga de reconocer y guardar los diferentes tokens.

Token : Secuencia de caracteres que tiene un significado para el compilador.

Como Funciona?

Lexer

La clase *Lexer* contiene las siguientes propiedades:

- _text : El string que representa la linea de codigo.
- _position : Permite ir iterando sobre los caracteres.
- Current : Representa el caracter actual

Y los siguientes metodos :

- Peek(n) : Permite mirar el caracter que se encuentra a cierta distancia del actual.
- GetToken() : Donde se realiza el proceso de reconocer y completar los tokens.

Como Funciona?

Lexer

El programa es capaz de reconocer y completar los siguientes tipos de tokens

- Literales(numericos, booleanos y string)
- Operadores
- Palabras clave
- Identificadores
- Espacios
- End-Of-Line(;)
- End-Of-File
- Bad Tokens(Tokens incorrectos)

Como Funciona?

Lexer

Al construir los tokens, se identifica su tipo de acuerdo al primer caracter.

Luego, para cada caso, se llama a una funcion encargada de completarlo(se mantiene leyendo caracteres hasta que encuentre alguno que no pueda formar parte del token)

Una vez completado un token, se devuelve y repite el proceso hasta llegar al ultimo caracter.

Como Funciona?

Parser

Parser : Recibe una grupo de tokens generado por el *Lexer* y crea expresiones a partir de ellos.

Contiene las siguientes propiedades :

- *_tokens* : array que representa el grupo de tokens.
- *_position* : para ir iterando sobre cada token.
- *Current* : se refiere al token actual.

Como Funciona?

Parser

Contiene los siguientes metodos :

- constructor : Ejecuta el Lexer obteniendo una lista de tokens.
- Peek(n) : Permite mirar el token que se encuentra a una distancia "n" del actual.
- NextToken() : Devuelve el token actual y cambia el puntero hacia el siguiente token.
- MatchKind(kind) : Chequea que el token actual coincida con el tipo especificado.
- ParseExpression() : Parsea las expresiones(Terminos ,Operadores Unarios, Operadores Binarios) segun la precedencia de los operadores.
- ParseTerm() : Parsea los terminos(literales, expresiones en parentesis, llamados de funcion , variables)
- ParseFunctionDeclaration() : Parsea la declaracion de funciones.

Como Funciona?

Parser

En la clase CompilatorTools implemente un metodo para obtener la precedencia de cada operador

- Operadores Unarios (-,+,!)
- Exponente (^)
- Division y Multiplicacion (/ , *)
- Suma, Resta , Resto y Concatenacion (+ , - , % , @)
- Comparativos(>= , == , != , etc)
- Operadores Logicos (& , && , |, ||)

Como Funciona?

Expresiones

Para representar las expresiones utilizo una clase padre llamada SyntaxExpression de la cual heredan las siguientes :

- LiteralExpression
- UnaryOperatorExpression
- BinaryOperatorExpression
- ParenthesiedExpression
- VariableExpression
- IfExpression
- LetInExpression
- DeclaredFunctionExpression
- PredefinedFunctionExpression
- FunctionCallExpression

Toda expresion tiene los metodos SemanticErrors() y Evaluate()

SemanticErrors() se encarga de chequear si la operacion que realiza la expresion es valida para el tipo de los datos ingresados.

Evaluate() se encarga de devolver el valor resultante de ejecutar la expresion

Ejemplo : Para evaluar BinaryOperatorExpression se obtienen los valores de la expresion a la izquierda y derecha del operador y se efectua la operacion correspondiente.

Como Funciona?

Manejo de Errores

En esta version de *HULK* existen 4 tipos de errores : Lexical Error, Syntactic Error , Semantic Error y RunTime Error

Lexical Error : Errores que se producen por la presencia de tokens invalidos.

Syntactic Error : Errores producidos por expresiones mal formadas, como parentesis mal balanceados.

Semantic Error : Errores producidos por el uso incorrecto de los tipos y argumentos.

RunTime Error : Errores producidos en tiempo de ejecucion, como StackOverflow.

Indice

Conclusiones

Trabajar en este segundo proyecto ha sido un gran desafio. Ha sido de gran utilidad para comprender una parte del complejo razonamiennto detras de los compiladores, asi como mejorar mis conocimientos de Programacion Orientada a Objetos, especificamente en el uso de herencia y encapsulacion de clases.

A pesar de las dificultades y que se podrian mejorar algunos aspectos(sobre todo en la organizacion del codigo) estoy contento con el resultado final.

Conclusiones