# HULK

Raciel Alejandro Simón Domenech

## 1. Introducción

Estos dos proyectos, montados en el lenguaje C, usando .NET Core 7 ,forman un intérprete de un subconjunto lenguaje HULK. El primer proyecto consta de Una aplicación de consola donde se implementa la parte interactiva del intérprete y el segundo cuenta con una biblioteca de clases donde se implementa toda la lógica de parsing y evaluación del lenguaje HULK.

## 2. Interpretación

El proceso de interpretar el lenguaje HULK se ha dividido en tres partes fundamentales:
-Tokenización.
-Parsing.
-Evaluación.

### 2.1. Tokenización

Una vez recibido la entrada del usuario esta se introduce en el método:

```
public static List<Token> TokensInit(string input)
```

el cual inicializa un objeto de tipo Token. La clase Token se utiliza para representar los objetos que constituyen la entrada del usuario. Tiene propiedades y métodos para convertir los tokens, reemplazar identificadores por valores específicos y clonar los tokens. Tiene las siguientes propiedades:
Type: Representa el tipo de token.
Content: Representa el contenido del token.
bool_exp: Lista de tokens que contiene los valores de la condición del if en un token del tipo if-else.
exp: Contiene valores y se utiliza en tokens mixtos.
exp_2: Contiene valores y se utiliza en tokens mixtos que tienen varios cuerpos, como los tokens if-else y let-in.
Además, la clase Token tiene los siguientes métodos:
ToString: Retorna una representación en cadena del objeto Token.
Clone: Crea una copia del objeto Token.
Remplace : Reemplaza los identificadores por valores específicos en los tokens

mixtos, utilizando un diccionario que contiene el nombre de la variable y un token asociado.
TryConvertToBool: Intenta convertir el token actual de identificador a bool, si es posible.
TryConvertToConst: Intenta convertir el token actual de identificador a constante, si es posible.

La clase Token contiene los siguientes tipos:
func: Representa una función.
cpar: Representa una expresión de paréntesis.
if-else: Representa una expresión if-else.
let-in: Representa una expresión let-in.
number: Representa un número.
string: Representa un string.
bool: Representa un booleano.
iden: Representa un identificador.
Operator: Representa un operador.
Symbol: Representa la coma y el punto y coma.

## 2.2. Parsing

Una vez concluida la etapa de tokenizacion. Se construye un árbol de nodos utilizando el método:

```
public static Node Parse(List<Token> tokens, int startlevel)
```

el cual va a ir construyendo los nodos teniendo en cuenta el nivel que se le pase, desde el 1 hasta el 8. Los niveles están divididos:
Nivel 1: & —
Nivel 2:
Nivel 3: $\leq \geq \neq <>$
$Nivel4$ : @
Nivel 5: * / %
Nivel 6 ^
Nivel 7 No se implementó
Nivel 8 "let-in", "print", "number", çpar", "bool", ïf-else", "strings", "func"

La clase `Node` representa un nodo . Se utiliza de evaluar el valor del nodo actual y sus respectivos hijos, devolviendo un `Token` como resultado.

La función `GetValue` es la encargada de realizar esta evaluación.

La propiedad `IsTerminal` indica si el nodo es terminal o no. La propiedad `token` almacena el `Token` asociado al nodo. La propiedad `childs` es un array que contiene los posibles nodos hijos del nodo actual.

Caza nodo se evalua según su tipo, si es del tipo func, se llama a la clase Function y se evalua segun corresponda, sino se reduce a una serie operaciones aritméticas y lógicas que se resuelven en la clase Operations

# 3. Manejo de errores

El intérprete cuenta con un objeto de tipo Error donde y una lista de errores activos, donde se guardan los errores encontrados , cada uno contiene utilizando breve descripción al respecto. Entre cada paso antes mencionado se comprueba si ha habido algún error, de haber habido se detiene la ejecución y se informa al respecto.

# 4. Clases Adicionales

## 4.1. Clase Function

Clase que representa las funciones que se pueden declarar e interpretar. Consta de dos constructores, uno privado para definir las funciones predefinidas y otro público, el público es llamado por el método:

```
public static bool GetFunction(List<Token> tokens)
```

el cual es usado en la declaración de funciones. También cuenta con el método Call, retorna un valor según los valores que se le pasen como parámetros y se usa cuando se invoca una función. Por último cuenta con el método InitBasicFunctions, que inicializa las funciones predefinidas: seno, coseno,logaritmo, exponenciación, raíz cuadrada, print. Consta de una lista de funciones: ActiveFunctions , que contiene las funciones que ya se han declarado.

## 4.2. Clase Operations

Clase que contiene las definiciones de las operaciones matemáticas y lógicas básicas así como algunas expresiones. Presenta los métodos:
BinaryOperation: encargado de las operaciones de suma, resta , división con resto y sin resto, y exponenciación.
LogicOperation: encargado de las operaciones lógicas.
LetIn: evalúa las expresiones let-in.
IfElse: evalúa las expresiones if-else.
StringSum: Suma números o cadenas de texto para formar cadenas de texto.
UnaryNegation: operación lógica unaria de negación.

# 5. Consola

La consola representa la parte interactiva del intérprete. Se encarga de guardar la entrada del usuario e ir invocando los métodos de tokenizacion, parsing y evaluación. retornando el valor resultante o reportando el error correspondiente.