H.U.L.K. Interpreter

Luis Daniel Silva Martínez

MATCOM

School of Math and Computer Science
Havana University

Índice

1. Introducción

Este proyecto es un intérprete que evalúa instrucciones del lenguaje H.U.L.K. una por una, cada una terminada en punto y coma, las cuales pueden ser expresiones aritméticas, declaraciones de variables, condicionales y funciones definidas por el usuario.

1.1. H.U.L.K.

H.U.L.K. (Havana University Language of Kompilers) es un lenguaje de programación imperativo, funcional, estática y fuertemente tipado. Casi todas las instrucciones en H.U.L.K. son expresiones.

1.2. Intérprete

Este proyecto de programación es un interpréte del lenguaje H.U.L.K. en C# usando tecnología .NET Core 7.0. Tiene una solución que contiene dos proyectos:

- Una biblioteca de clases (HULK_Interpreter) donde se implementa toda la lógica de parsing y evaluación del lenguaje H.U.L.K. haciendo uso solamente la biblioteca estándar de .NET Core.
- Una aplicación de consola (HULK_ConsoleInterpreter) donde se implementa la parte interactiva del intérprete.

1.3. ¿Cómo usarlo?

El uso del intérprete es simple, solo escribir expresiones válidas del lenguaje H.U.L.K. y presionar `ENTER`, una única instrucción a la vez, y recibirá el resultado de evaluar la expresión. En caso de algún error en las instrucciones enviadas se le notificará sobre el mismo.

1.4. ¿Cómo ejecutarlo?

Ejecutar en la terminal desde la carpeta principal del proyecto el comando:
dotnet run –project HULK_ConsoleInterface
O abrir la carpeta del HULK_ConsoleInterface y ejecutar:
dotnet run

2. Implementación

Este es un ejemplo de una posible interacción:

```
>let x = 42 in print(x);
42
>function fib(n) => if (n > 1) fib(n-1) + fib(n-2) else 1;
Function 'fib' declared succesfully
>fib(5);
```

```
13
>let x = 3 in fib(x+1);
8
>print(fib(6));
21
```

Cada línea que comienza con `>` representa una entrada del usuario, e inmediatamente después se imprime el resultado de evaluar esa expresión.

2.1. Flujo

El proyecto sigue un camino marcado:

1. Instanciación de la clase Interpreter e inicialización de la memoria con las funciones predefinidas.
2. Recepción del código introducido por el usuario
3. Tokenización del código y validación léxica con la clase Lexer.
4. Parseo del código, construcción del árbol de sintaxis abstracta y validación de la sintaxis en la clase Parser.
5. Evaluación del árbol de sintaxis abstracta y comprobación semántica en la clase Evaluator.
6. Obtención e impresión del resultado.
7. Nuevo código.

2.2. Class Interpreter

La clase `Interpreter` es la clase principal que maneja todo el proceso del intérprete. Se encarga de coordinar el análisis léxico, el análisis sintáctico y la evaluación del código fuente. Actúa como una interfaz entre estas diferentes etapas y garantiza que el proceso de interpretación se realice de manera ordenada y correcta.

2.3. Class Memory

La clase `Memory` simula la memoria del intérprete. Guarda las funciones predefinidas del lenguaje y las definidas por el usuario durante la ejecución del programa.

2.4. Class Token

La clase `Token` representa un token individual en el análisis léxico, con su tipo de token definido en el Enum `TokenType`, su lexema y su valor en caso de tenerlo.

2.5. Class Lexer

La clase `Lexer` es responsable de realizar el análisis léxico. Escanea el código fuente y lo convierte en una secuencia de tokens significativos que posteriormente serán utilizados en el proceso de análisis sintáctico.

2.6. Class Parser

La clase `Parser` analiza la estructura sintáctica del código fuente, utilizando los tokens generados por el `Lexer`, y construye una representación estructurada del programa, un árbol de sintaxis abstracta (AST). Utiliza parsing recursivo descendente, concepto basado en la idea de dividir el problema en subproblemas más pequeños y resolverlos de manera recursiva.

2.7. Class Evaluator

La clase `Evaluator` es responsable de llevar a cabo la interpretación del código fuente y producir los resultados deseados según las reglas y semántica de nuestro lenguaje.

2.8. Class Expression

La clase `Expression` es la clase base de todas las expresiones de nuestro lenguaje H.U.L.K., de la cual heredan `BinaryExpression`, `UnaryExpression`, `LiteralExpression`, y el resto de expresiones, cada una con sus características propias.

2.9. Class Error

La clase `Error` hereda de Exception y permite representar y manejar errores léxicos, sintácticos y semánticos durante la ejecución del intérprete.