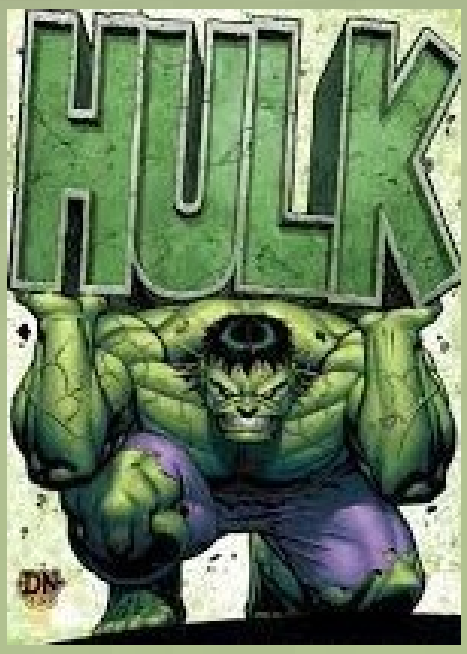
2do Proyecto de Programación
HULK: Havana University Language for Kompilers

Ronald Provance Valladares C113



Facultad de Matemática y Computación
Universidad de la Habana

Resumen

El lenguaje HULK (simplificado)

HULK es un lenguaje de programación imperativo, funcional, estático y fuertemente tipado. Casi todas las instrucciones en HULK son expresiones. En particular,hasta el momento, se compone solamente de expresiones que pueden escribirse en **una línea**. Y para ello se ha elaborado este proyecto,con el fin de compilar dicho lenguaje.el cual se divide en cuatro secciones:

1. Analizador Léxico
2. Analizador Semántico
3. Analizador Sintáctico
4. Evaluación de código

1. Introducción

Todo lo básico que debe conocer para poder usar **HULK**:

El intérprete de HULK será una aplicación de consola, donde el usuario puede introducir una expresión de HULK, presionar **ENTER**, e inmediatamente se verá el resultado de evaluar expresión (si lo hubiere) en la siguiente línea (en caso de ser una declaración de función, esta simplemente se guardará para futuros llamados). El proceso se continuará repitiendo, hasta que el usuario presione la tecla de *ESCAPE*, ocasionando el cierre de la aplicación.

1.1. ¿Qué es un Compilador?

Un **compilador** es un programa o software que se encarga de traducir el código fuente de un programa escrito en un lenguaje de programación de alto nivel a un lenguaje de programación de bajo nivel, generalmente lenguaje máquina o lenguaje ensamblador, que pueda ser entendido y ejecutado por la computadora.

El compilador realiza este proceso en varias etapas, que incluyen el ***análisis léxico, el análisis sintáctico, el análisis semántico, la generación de código***. entre otros.pero al ser un proyecto sencillo, a menor escala solo consta de estas etapas.

Los compiladores son esenciales para el desarrollo de software, ya que permiten a los programadores escribir código en lenguajes de alto nivel más legibles y comprensibles, en lugar de tener que escribir directamente en lenguaje máquina.

2. Estructura

2.1. Análisis Léxico(Tokenizer)

En este proceso la entrada dada por el usuario es analizada y la convierto en una secuencia de **Token** en el orden en que se va analizando la entrada guardándolos en una lista. Seguramente se preguntarán que es un token, no es mas que una secuencia de caracteres llamada cadena; estos token pueden ser de varios tipos como palabras claves,llamadas **keyword** como por ejemplo: int, if,function,entre otras; otro tipo son los **operadores** o conocidos normalmente como símbolos,los cuales son: +, —, ¿=, entre otros; también están los **identificadores** conocidos por los programadores como variables: x, z,entre otras; también existen declaraciones de **funciones** creadas por el usuario,como los son: el calculo del factorial de un número(Fact),los términos de la sucesión de Fibonacci(Fib) y por ultimo pero no menos importantes las expresiones **numéricas**,**cadenas** y **booleanos**. Donde cada tipo de token lo guardo en un diccionario

2.2. Análisis Sintáctico(Parser)

En este momento se tiene la lista de **Tokens** elaborada por el analizador léxico y analiza si dicha lista contiene un orden correcto. Un orden válido esta dado por el lenguaje a compilar, en este caso HULK. Para ello se emplea una gramática libre de contexto, la cual representa la sintaxis del lenguaje. Dicha gramática se muestra a continuación:

```
HULK Grammar
L=M;
M= Print(M)| If(M) | Function(M) | Let | A
A= B Z| !B Z
Z= and A | or A | e
B= W E
E= <B | >B | <=B | >=B | ==B | !=B | e
W= F X
X= +W | -W |e
F= T P
P= *F | /F |^F | % F | e
T= int | string | Function()| (M)
```

Figura 1: Gramática

Además en este momento se genera el **Árbol de Sintaxis Abstracta** (AST según sus siglas en Inglés). En este **AST**, los nodos son expresiones que se desglosan en: otras expresiones, operaciones o términos;como esta mostrado en la gramática antes dada. Este **AST** es el que conforma la guía de instrucciones que debe seguir rl proyecto utilizando **Recursividad,** y para obtenerlo se emplea la clase abstracta **Expression**. de la cual hablaremos posteriormente.

2.3. Análisis Semántico(CheckSemantic)

2.4. Evaluación de código(Evaluate)

En este momento del proyecto ya se llevó a cabo el análisis **léxico,sintáctico y semántico**, por lo tanto ya esta listo para realizar **la evaluación** de la entrada dada por el usuario y se **evalúa dicho código** y se devuelve el **resultado obtenido**.

3. Clase Expression

Es una clase abstracta, que contiene los métodos .ªvaluatez ÇheckSemantic"; por tanto todas las clases que hereden de ella tendrán estos métodos por lo que cada clase de subtipos de expresiones(clases que heredan de .ªxpression") tendrán su propio ÇheckSemantic"que se lleva a cabo antes de construir el AST y el .ªvaluate"después de confeccionado el AST.

4. Errores

En HULK hay 3 tipos de errores que se deben detectar. En caso de detectarse un error, el intérprete imprimirá una línea indicando el error siendo lo más informativa posible.

- ***Error léxico:***
  Errores que se producen durante el análisis léxico por la presencia de tokens inválidos
  Ejemplos: 3x, "palabra
- ***Error sintáctico:***
  Errores que se producen durante el análisis sintáctico por expresiones mal formadas como paréntesis no balanceados o expresiones incompletas.
  Ejemplos: (let x=5 y=2) , (if(3==4) print(false))
- ***Error semántico:***
  Errores que se producen durante el análisis semántico por el uso incorrecto de los tipos y argumentos.
  Ejemplos: 5*true , "palabra»=3

¡Muchas Gracias!