HULK: Havana University Language for Kompilers

Gabriel Herrera Carazana

2023

Abstract

El objetivo de este proyecto es implementar un intérprete del lenguaje de programación HULK en C#. La motivación de este proyecto es proveer una herramienta didáctica y práctica como un primer acercamiento a la teoría de compiladores.

HULK es un lenguaje de programación imperativo, funcional, estática y fuertemente tipado. Casi todas las instrucciones en HULK son expresiones. En particular, el subconjunto de HULK implementado se compone solamente de expresiones que pueden escribirse en una línea.

Índice

El código está distribuido en 3 archivos.cs Los archivos que conforman el intérprete de HULK son los siguientes:

Contents

1 Consola.cs

La clase Consola define un método Ejecutar que se encarga de leer las líneas de código que el usuario introduce por la consola y ejecutarlas usando el intérprete del lenguaje HULK. La clase Consola también tiene un campo estático entorno que representa el entorno de ejecución del intérprete, es decir, el conjunto de variables y valores que se definen y usan en el código HULK. El método Ejecutar tiene un bucle while que se repite hasta que el usuario introduce la palabra

“salir”. Dentro del bucle, se realizan los siguientes pasos: Se lee una línea de código de la consola y se almacena en la variable linea. Si la línea está vacía, se continúa con la siguiente iteración del bucle.

Se crea un objeto de la clase AnalizadorLéxico pasándole la línea como argumento. Esta clase se encarga de dividir la línea en tokens, que son las unidades mínimas de significado del lenguaje HULK, como palabras clave, identificadores, operadores, etc. Se llama al método ObtenerTokens del objeto AnalizadorLéxico para obtener una lista de tokens que representan la línea de código. Se crea un objeto de la clase AnalizadorSintáctico pasándole la lista de tokens como argumento. Esta clase se encarga de construir un árbol de sintaxis abstracta (AST) a partir de los tokens, siguiendo las reglas gramaticales del lenguaje HULK. El AST es una estructura de datos que representa la estructura lógica del código y sus relaciones.

Se llama al método Analizar del objeto AnalizadorSintáctico para obtener el AST que representa la línea de código. Se llama al método Evaluar del AST, pasándole el entorno como argumento. Este método se encarga de evaluar el AST y devolver el resultado de la ejecución del código HULK. El método Evaluar puede modificar el entorno si el código HULK define o asigna variables. Se imprime el resultado de la evaluación por la consola.Se captura cualquier excepción que pueda ocurrir durante el proceso de análisis o evaluación y se imprime el mensaje de error por la consola. La clase Program define el método Main que se ejecuta al iniciar el programa. Este método crea un objeto de la clase Consola y llama a su método Ejecutar.

2 Arbol.cs

El código define dos clases principales: Token y AST, y un enumerador: TipoToken. El enumerador TipoToken de tipo enum , contiene los diferentes tipos de tokens que se pueden encontrar en el lenguaje Hulk. Estos son:
PalabraReservada,Función,Identificador,Numero ,OperadorAritmético,OperadorLógico ,OperadorAsignación,DelimitadorAbierto, DelimitadorCerrado,Math,Comillas,PuntoComa, Coma,Desconocido,Flecha,Cadena y Concatenador.
La clase Token representa un token individual que se extrae del código fuente. Tiene dos propiedades: Tipo, que es de tipo TipoToken, y Valor, que es de tipo string. Ambas propiedades son de solo lectura, lo que significa que solo se pueden asignar en el constructor de la clase.
La clase AST es una clase abstracta que representa un árbol de sintaxis abstracta, que es una estructura de datos que almacena la información sintáctica y semántica del código. Tiene un método abstracto Evaluar, que toma un objeto Entorno como parámetro y devuelve el resultado de la evaluación del árbol.
Luego se definen varias clases que heredan de la clase abstracta AST. Cada clase tiene un método Evaluar que devuelve el resultado de la evaluación del nodo correspondiente. Las clases son las siguientes:
La clase ValorNumerico representa un valor numérico. Tiene una propiedad Valor de tipo double que almacena el valor numérico. El método Evaluar de-

vuelve el valor numérico.
La clase OperacionAritmetica representa una operación aritmética entre dos operandos. Tiene tres propiedades: OperandoIzquierdo y OperandoDerecho, que son de tipo AST, y Operador, que es de tipo string. El método Evaluar convierte los operandos a double y realiza la operación indicada por el operador, que puede ser +, -, *, /,
La clase Negacion representa la negación de un valor numérico. Tiene una propiedad subexpresion de tipo AST que almacena el valor a negar. El método Evaluar obtiene el valor de la subexpresión y lo convierte a double. Si el valor es numérico, devuelve su negativo. Si no, se lanza una excepción.
La clase Cadena representa una cadena de texto. Tiene una propiedad Valor de tipo string que almacena el valor de la cadena. El método Evaluar devuelve el valor de la cadena.
La clase FuncionLog representa el logaritmo de un argumento en una base. Tiene dos propiedades: BaseLog y ArgumentoLog, que son de tipo AST. El método Evaluar convierte los valores de la base y el argumento a double y devuelve el resultado de aplicar la función Math.Log. La clase FuncionSin representa el seno de un argumento. Tiene una propiedad Argumento de tipo AST. El método Evaluar convierte el valor del argumento a double y devuelve el resultado de aplicar la función Math.Sin. La clase FuncionCos representa el coseno de un argumento. Tiene una propiedad Argumento de tipo AST. El método Evaluar convierte el valor del argumento a double y devuelve el resultado de aplicar la función Math.Cos.

La clase OperadorLogico representa una operación lógica entre dos operandos. Tiene tres propiedades: OperandoIzquierdo y OperandoDerecho, que son de tipo AST, y Operador, que es de tipo string. El método Evaluar obtiene los valores de los operandos y los convierte al tipo apropiado según el operador, que puede ser ==, !=, ¿, ¡, . Si el operador es desconocido, se lanza una excepción.

La clase Identificador representa el nombre de una variable. Tiene una propiedad Nombre de tipo string que almacena el nombre de la variable. El método Evaluar busca la variable en el entorno y devuelve su valor. Si la variable no está definida, se lanza una excepción. La clase DeclaracionVariable representa la declaración de una variable con un valor inicial. Tiene dos propiedades: Nombre de tipo string y ValorInicial de tipo AST. El método Evaluar obtiene el valor del ValorInicial y lo asigna a la variable en el entorno. El método devuelve null, ya que la declaración de una variable no produce ningún valor.
La clase Concatenacion representa la concatenación de dos valores con un espacio entre ellos. Tiene dos propiedades: Izquierdo y Derecho, que son de tipo AST. El método Evaluar obtiene los valores de las propiedades y los convierte a string. Luego, devuelve la concatenación de los valores con un espacio entre ellos.
La clase LetInExpression representa una expresión que define un nuevo entorno con algunas variables y luego evalúa un cuerpo en ese entorno. Tiene dos propiedades: Entorno de tipo Entorno y Cuerpo de tipo AST. El método Evaluar copia las variables del Entorno al entorno actual y luego devuelve el resultado de evaluar el Cuerpo en ese entorno.

La clase IfElseExpression representa una expresión condicional que ejecuta una de las dos expresiones dependiendo de una condición. Tiene tres propiedades: Condicion, ExpresionIf y ExpresionElse, que son de tipo AST. El método Evaluar convierte la Condicion a bool y, si es verdadera, devuelve el resultado de evaluar la ExpresionIf. Si es falsa, devuelve el resultado de evaluar la ExpresionElse.
La clase PrintExpression representa una expresión que imprime el valor de otra expresión. Tiene una propiedad Expresion de tipo AST. El método Evaluar obtiene el valor de la Expresion y lo convierte a string. Luego, devuelve el valor como una cadena. La clase FuncionInline representa una función definida en el código fuente. Tiene tres propiedades: Nombre de tipo string, Parametros de tipo List¡string¿ y Cuerpo de tipo AST. El método Evaluar define la función en el entorno y devuelve null, ya que la definición de una función no produce ningún valor.
La clase LlamadaFuncion representa una llamada a una función con una lista de argumentos. Tiene dos propiedades: Nombre de tipo string y Argumentos de tipo List¡AST¿. El método Evaluar busca la función en el entorno y crea un nuevo entorno para la ejecución de la función. Luego, evalúa los argumentos y los asigna a las variables correspondientes en el entorno de la función. Finalmente, devuelve el resultado de evaluar el cuerpo de la función en ese entorno.
La clase Entorno representa el entorno de ejecución del intérprete del lenguaje Hulk. Tiene dos campos: variables y funciones, que son de tipo Dictionary¡string, Variable¿ y Dictionary¡string, FuncionInline¿, respectivamente. Estos campos almacenan las variables y las funciones definidas en el código fuente o en el entorno interactivo. La clase tiene cuatro métodos:
El método DefinirVariable toma un objeto Variable como parámetro y lo añade al diccionario variables, usando el nombre de la variable como clave. Este método se usa para declarar o asignar una variable en el entorno.
El método BuscarVariable toma un string como parámetro y devuelve el objeto Variable asociado a ese nombre, si existe. Si no existe, devuelve null. Este método se usa para obtener el valor de una variable en el entorno.
El método DefinirFuncion toma un objeto FuncionInline como parámetro y lo añade al diccionario funciones, usando el nombre de la función como clave. Este método se usa para definir una función en el entorno.
El método BuscarFuncion toma un string como parámetro y devuelve el objeto FuncionInline asociado a ese nombre, si existe. Si no existe, devuelve null. Este método se usa para llamar a una función en el entorno.
La clase Variable representa una variable con un nombre y un valor. Tiene dos campos: name y value, que son de tipo string y object, respectivamente. Estos campos almacenan el nombre y el valor de la variable. La clase tiene dos propiedades: Name y Value, que permiten acceder y modificar los campos.

3 Lexico.cs

Este código contiene una clase llamada AnalizadorLéxico que tiene como objetivo analizar un código fuente y dividirlo en tokens, que son las unidades básicas de un lenguaje de programación. El código fuente se pasa como un argumento al constructor de la clase, y se almacena en una variable de tipo string llamada codigoFuente. La clase también tiene una variable de tipo int llamada indice que indica la posición actual en el código fuente. La clase tiene un método llamado ObtenerTokens que devuelve una lista de tokens generados a partir del código fuente. El método usa un bucle while que recorre el código fuente caracter por caracter, y usa instrucciones if para clasificar cada tipo de token según el caracter actual. Por ejemplo, si el caracter actual es un espacio, un salto de línea o un retorno de carro, el método lo ignora y pasa al siguiente caracter. Si el caracter actual es un punto y coma o una coma, el método crea un token de tipo PuntoComa o Coma, respectivamente, y lo añade a la lista de tokens. La clase también tiene dos diccionarios que almacenan las palabras reservadas y las funciones matemáticas del lenguaje, y los usa para crear tokens de tipo PalabraReservada o Math cuando el caracter actual coincide con alguna de estas palabras. Los diccionarios son de tipo Dictionary¡string, TipoToken¿, y guardan los pares de clave y valor que representan la relación de correspondencia entre las palabras y los tipos de token. El método continúa hasta que se llega al final del código fuente, y devuelve la lista de tokens.

4 Sintactico.cs

El analizador sintáctico es la parte de un intérprete que se encarga de verificar que la expresión que se quiere evaluar cumple con las reglas gramaticales del lenguaje. Para ello, el analizador sintáctico construye un árbol de sintaxis abstracta (AST), que es una representación jerárquica de la estructura de la expresión. El AST se compone de nodos que representan los elementos de la expresión, como literales, variables, operadores, funciones, etc. El analizador sintáctico usa los tokens que le proporciona el analizador léxico, que son las unidades mínimas de significado del lenguaje. La clase a clase AnalizadorSintáctico tiene los siguientes campos y métodos: Un campo privado List¡Token¿ tokens que almacena la lista de tokens que se van a analizar. Un campo privado int indice que lleva la cuenta del índice del token actual en la lista. Un campo privado Token tokenActual que almacena el token actual que se está analizando. Un campo privado Token ultimoToken que almacena el último token de la lista Un campo privado estático List¡string¿ Nombrefunciones que almacena una lista de nombres de funciones que se han definido en el código. Un constructor público AnalizadorSintáctico(List¡Token¿ tokens) que recibe una lista de tokens como parámetro e inicializa los campos de la clase con los valores correspondientes. Un método público y void siguienteToken() que avanza el índice al siguiente token de la lista y actualiza el valor de tokenActual. Metodo Analizar : El método Analizar() es el método principal que se encarga

de analizar el código fuente y devolver el árbol de sintaxis abstracta (AST) correspondiente. Para ello, utiliza dos variables: ultimoToken y tokenActual, que son de tipo Token. comprueba si el último token leído del código fuente es un punto y coma, que indica el final de una expresión. Si no lo es, lanza una excepción .

Si lo es, evalúa el tipo y el valor del token actual, que es el siguiente token a analizar, usando una estructura de control if. Según el caso, llama a diferentes funciones que analizan las distintas estructuras sintácticas del lenguaje y devuelven el AST correspondiente. Estas funciones son: AnalizarExpresion(): Este método analiza una expresión que puede contener operadores lógicos (¡,¿,!=, !) y aritméticos (+, -, *, /, AnalizarOperacionAritmetica(): Este método analiza una operación aritmética que puede contener los operadores + y - y devuelve un AST de tipo OperacionAritmetica. Para ello, llama al método AnalizarTermino() y luego verifica si hay más operadores + o -, en cuyo caso crea un nuevo AST con el resultado anterior y el siguiente operando. AnalizarTermino(): Este método analiza un término que puede contener los operadores * , / y AnalizarFactor(): Este método analiza un factor que puede contener el operador ŷ devuelve un AST de tipo OperacionAritmetica. Para ello, llama al método AnalizarExponente() y luego verifica si hay más operadores ; en cuyo caso crea un nuevo AST con el resultado anterior y el siguiente operando. AnalizarExponente(): Este método analiza un exponente que puede ser un número, un identificador, una llamada a una función, una expresión entre paréntesis o una negación y devuelve un AST de tipo ValorNumerico, Identificador, LlamadaFuncion, Expresion o Negacion, según el caso. Para ello, verifica el tipo y el valor del token actual y crea el AST correspondiente, llamando a otros métodos si es necesario. AnalizarIfElse(): Este método analiza una estructura condicional if-else y devuelve un AST de tipo IfElseExpression, que tiene tres atributos: Condicion, ExpresionIf y ExpresionElse, que son AST que representan la condición y las expresiones que se ejecutan si la condición es verdadera o falsa, respectivamente. Para ello, llama al método Analizar() para obtener cada uno de estos atributos y verifica que el token actual sea la palabra reservada else, en caso contrario lanza una excepción. FuncionDefinida(): Este método recibe un nombre de función como parámetro y devuelve un valor booleano que indica si la función está definida o no. Para ello, verifica si el nombre de la función está en la lista Nombrefunciones, que es una lista de cadenas que almacena los nombres de las funciones definidas. AnalizarFuncionInline(): Este método analiza una definición de función anónima con la palabra reservada function y devuelve un AST de tipo FuncionInline, que tiene tres atributos: NombreFuncion, Parametros y Cuerpo, que son una cadena, una lista de cadenas y un AST que representan el nombre de la función, la lista de parámetros y el bloque de código que se ejecuta al llamar a la función, respectivamente. Para ello, verifica que el token actual sea un identificador, un paréntesis abierto, una coma, un paréntesis cerrado o una flecha, según el caso, y llama al método Analizar() para obtener el cuerpo de la función. Además, agrega el nombre de la función a la lista Nombrefunciones. AnalizarLlamadaFuncion(): Este método analiza una llamada a una función y devuelve un AST de tipo LlamadaFuncion, que tiene

dos atributos: NombreFuncion y Argumentos, que son una cadena y una lista de AST que representan el nombre de la función y la lista de argumentos que se le pasan, respectivamente. Para ello, verifica que el token actual sea un paréntesis abierto, una coma o un paréntesis cerrado, según el caso, y llama al método Analizar() para obtener cada argumento. AnalizarIdentificador(): Este método analiza un identificador y devuelve un AST de tipo Identificador, que tiene un atributo: Valor, que es una cadena que representa el nombre del identificador. Para ello, crea el AST con el valor del token actual y luego verifica si hay un operador de concatenación, en cuyo caso llama al método Analizar() para obtener el otro operando y crea un nuevo AST de tipo Concatenacion, que tiene dos atributos: Izquierdo y Derecho, que son AST que representan los operandos de la concatenación. AnalizarLetIn(): Este método analiza una declaración de variables locales con la palabra reservada let y devuelve un AST de tipo LetInExpression, que tiene dos atributos: Entorno y Cuerpo, que son un objeto de tipo Entorno y un AST que representan el entorno donde se declaran las variables y el bloque de código que se ejecuta después de la declaración, respectivamente. Para ello, verifica que el token actual sea un identificador, un operador de asignación, una coma o la palabra reservada in, según el caso, y llama al método Analizar() para obtener el valor de cada variable. Además, agrega o actualiza las variables en el entorno usando la clase Variable, que tiene dos atributos: Nombre y Valor, que son una cadena y un objeto de tipo Object que representan el nombre y el valor de la variable, respectivamente. AnalizarPrint(): Este método analiza una instrucción de impresión por pantalla con la palabra reservada print y devuelve un AST de tipo PrintExpression, que tiene un atributo: Expresion, que es un AST que representa la expresión que se imprime por pantalla. Para ello, verifica que el token actual sea un paréntesis abierto, llama al método Analizar() para obtener la expresión y verifica que el token actual sea un paréntesis cerrado. AnalizarFuncionSin(): Este método analiza una llamada a la función matemática seno y devuelve un AST de tipo FuncionSin, que tiene un atributo: Expresion, que es un AST que representa el argumento de la función seno. Para ello, verifica que el token actual sea un paréntesis abierto, llama al método Analizar() para obtener la expresión y verifica que el token actual sea un paréntesis cerrado. AnalizarFuncionCos(): Este método analiza una llamada a la función matemática coseno y devuelve un AST de tipo FuncionCos, que tiene un atributo: Expresion, que es un AST que representa el argumento de la función coseno. Para ello, verifica que el token actual sea un paréntesis abierto, llama al método Analizar() para obtener la expresión y verifica que el token actual sea un paréntesis cerrado. AnalizarFuncionLog(): Este método analiza una llamada a la función matemática logaritmo y devuelve un AST de tipo FuncionLog, que tiene dos atributos: BaseLog y ArgumentoLog, que son AST que representan la base y el argumento del logaritmo, respectivamente. Para ello, verifica que el token actual sea un paréntesis abierto, llama al método Analizar() para obtener la base, verifica que el token actual sea una coma, llama al método Analizar() para obtener el argumento y verifica que el token actual sea un paréntesis cerrado. esOperadorLogico(): Este método recibe un valor de tipo cadena como parámetro y devuelve un valor booleano que indica si el

valor es un operador lógico o no. Para ello, compara el valor con los posibles operadores lógicos, como ==, !=, ¿, ¡. esOperadorAritmetico(): Este método recibe un valor de tipo cadena como parámetro y devuelve un valor booleano que indica si el valor es un operador aritmético o no. Para ello, compara el valor con los posibles operadores aritméticos, como +, -, *, y /