Universidad de Mariano Gálvez

Facultad de Ingeniería

Compiladores

Catedrático: Ing. Miguel Catalán

Proyecto Final

Manual Técnico

Nombre: Marco Vinicio Rac Cotzojay 7590-20-17448

Objetivos y alcances del sistema

Objetivos generales:

1. Entender el funcionamiento de un compilador en sus dos primeras fases.
2. Entender los conceptos de análisis.
3. Realizar el uso correcto de herramientas de análisis.
4. Comprender los tipos de análisis sintáctico.

Objetivos específicos:

1. Establecer y programar el proceso de reconocimiento de análisis léxico y sintáctico.
2. Reforzar el concepto y análisis del árbol sintáctico abstracto
   Realizar la construcción correcta de un árbol sintáctico.
3. Entender las diferencias sintácticas entre lenguajes.
4. Utilizar correctamente herramientas de análisis léxico y sintáctico JFex y Cup.

El programa tiene como objetivo el análisis de entrada de código por medio de una caja de texto o la carga de un archivo Java, este realizará su análisis de forma léxica y sintáctica, una vez finalizado el análisis y este funcione de manera correcta podremos realizar una traducción al lenguaje de programación Python.

Especificación Técnica

Requisitos de Hardware

- **Procesador:** Intel Core i3 (Intel Core i3 o equivalente)
- **RAM:** 2 GB, 4 GB (2 GB (32-bit), 4 GB (64-bit))
- **Disco duro:** 1.5 GB HDD
- **Tarjeta gráfica:** No necesario.
- **Resolución de la pantalla:** 1024 x 728

Requisitos de software

- **Sistemas Operativos:**
    - **Windows:** Windows 10, Windows 7, Windows XP, Windows Vista (Windows XP Professional SP3/Vista SP1/Windows 7 Professional).
    - **Mac:**
        - Mac con Intel que ejecuta Mac OS X 10.8.3+, 10.9+
        - Privilegios de administrador para la instalación
        - Explorador de 64 bits
    - **Linux:**
        - Oracle Linux 5.5+[1]
        - Oracle Linux 6.x (32 bits), 6.x (64 bits)[2]
        - Oracle Linux 7.x (64 bits)[2] (8u20 y superiores)
        - Red Hat Enterprise Linux 5.5+[1] 6.x (32 bits), 6.x (64 bits)[2]
        - Red Hat Enterprise Linux 7.x (64 bits)[2] (8u20 y superiores)
        - Suse Linux Enterprise Server 10 SP2+, 11.x
        - Suse Linux Enterprise Server 12.x (64 bits)[2] (8u31 y superiores)
        - Ubuntu Linux 12.04 LTS, 13.x
        - Ubuntu Linux 14.x (8u25 y superiores)
        - Ubuntu Linux 15.04 (8u45 y superiores)
        - Ubuntu Linux 15.10 (8u65 y superiores)
        - Exploradores: Firefox

- **Java 8:** Se adjunta un enlace de descarga: https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html
- **Maven 3.3.9:** Se adjunta un enlace de descarga: https://maven.apache.org/docs/3.3.9/release-notes.html
- **JFlex:** Se adjunta un enlace de descarga: https://www.jflex.de/
- **CUP**
- **Intellij o Netbeans IDE**

- **Visual estudio Code:** Para poder revisar correctamente el código es necesario el uso de un IDE de programación el cual se recomienda Visual studio code al ser elaborado en este, se adjunta un enlace de descarga seguro Download Visual Studio Code - Mac, Linux, Windows

Almacenamiento de Tokens

Lexer.flex: Este archivo es un analizador léxico generado con JFlex, una herramienta que genera analizadores léxicos en Java a partir de una especificación formal. El analizador léxico es la primera fase de un compilador, y su trabajo es leer el código fuente y convertirlo en una secuencia de tokens, que son las unidades básicas de significado del lenguaje.

Una descripción de cada una de las gramáticas ubicadas en el archivo .jlfex:

1. **import java_cup.runtime.*;**: Esta línea importa todas las clases del paquete **java_cup.runtime**, que proporciona las clases necesarias para interactuar con el analizador sintáctico generado por CUP.

2. **%%**: Este es un separador que divide el archivo en dos secciones. La primera sección contiene opciones y declaraciones de código de usuario, y la segunda sección contiene las reglas léxicas.

3. **%public**: Esta opción hace que la clase generada sea pública.

4. **%class Lexer**: Esta opción establece el nombre de la clase generada a **Lexer**.

5. **%char**, **%line**, **%column**: Estas opciones hacen que el analizador léxico cuente los caracteres, las líneas y las columnas del código fuente, respectivamente.

6. **%cup**: Esta opción hace que el analizador léxico genere código para interactuar con un analizador sintáctico CUP.

7. **%{ ... %}**: Este bloque contiene código de usuario que se copia directamente en la clase generada. En este caso, el código define un método **symbol** que crea y devuelve un objeto **Symbol**.

8. **palabra = [a-zA-Z]+**: Esta línea define un patrón llamado **palabra** que reconoce una o más letras.

9. **digito = -?[0-9]+**: Esta línea define un patrón llamado **digito** que reconoce un número entero, que puede ser negativo.

10. **digito_double = -?[0-9]+(\.[0-9]+)?**: Esta línea define un patrón llamado **digito_double** que reconoce un número de punto flotante, que puede ser negativo.

11. **digito_long = -?[0-9]+(\.[0-9]+)?**: Esta línea define un patrón llamado **digito_long** que reconoce un número largo, que puede ser negativo.

12. **identificador = [a-zA-Z_][a-zA-Z0-9_]***: Esta línea define un patrón llamado **identificador** que reconoce los identificadores en Java.

13. **espacios_blanco = [\r|\n|\r\n| |\t]**: Esta línea define un patrón llamado **espacios_blanco** que reconoce los espacios en blanco.

14. **"main"**: Reconoce la palabra clave "main" en el código fuente. Cuando se encuentra, imprime información sobre el lexema y devuelve un símbolo con el tipo **sym.MAIN**.

15. **"public"**, **"private"**, **"static"**, **"class"**, **"args"**, **"void"**, **"float"**, **"double"**, **"int"**, **"long"**, **"short"**, **"String"**, **"boolean"**, **"new"**, **"if"**, **"else"**, **"for"**, **"while"**, **"do"**, **"switch"**, **"case"**, **"break"**, **"default"**, **"Scanner"**, **"next"**, **"nextInt"**, **"nextFloat"**, **"nextLine"**, **"nextLong"**, **"nextShort"**, **"nextBoolean"**, **"nextDouble"**, **"out"**, **"print"**, **"println"**, **"return"**, **"System"**, **"in"**, **"true"**, **"false"**: Estas reglas reconocen varias palabras clave de Java. Cuando se encuentra una de estas palabras clave, imprime información sobre el lexema y devuelve un símbolo con un tipo correspondiente.

16. **"="**, **">"**, **"<"**, **"=="**, **"!="**, **"<="**, **">="**, **"+"**, **"-"**, **"*"**, **"^"**, **"/"**, **"&&"**, **"||"**, **"!"**, **"."**, **";"**, **":"**, **"_"**, **","**, **"\""**, **"'"**, **"("**, **")"**, **"{"**, **"}"**, **"["**, **"]"**: Estas reglas reconocen varios operadores y símbolos de puntuación en Java. Cuando se encuentra uno de estos símbolos, imprime información sobre el lexema y devuelve un símbolo con un tipo correspondiente.

17. **{identificador}**: Esta regla reconoce los identificadores en Java, que son nombres de variables, métodos, clases, etc. Cuando se encuentra un identificador, imprime información sobre el lexema y devuelve un símbolo con el tipo **sym.IDENTIFICADOR**.

18. **{digito}**, **{digito_double}**, **{digito_long}**: Estas reglas reconocen números en Java, incluyendo enteros, doubles y longs. Cuando se encuentra un número, imprime información sobre el lexema y devuelve un símbolo con un tipo correspondiente y el valor del número.

19. **{espacios_blanco}**: Esta regla reconoce espacios en blanco, que se ignoran en Java.

Gramáticas

Analizador Sintáctico: Dentro del packete cup podemos encontrar el analizador sintáctico, su función es recibir todos los tokens generados por el analizador léxico y organizarlos a modo de saber si estos están debidamente estructurados, la forma en la cual se creó este analizador fue siguiendo la misma lógica de un autómata teniendo estados que irán analizando los tokens y estos siguen el patrón correspondiente serán aceptados, caso contrario serán tomados como error.

Estas estarán almacenadas dentro del archivo “Syntax.cup” en el cual podremos ver como se irán desarrollando los órdenes que debe seguir el proyecto y como este debe comportarse conforme al ingreso de los tokens generados por las expresiones regulares, esto ayuda mucho a poder identificar cada una de las diferentes funcionalidades.

Una descripción de cada una de las gramáticas ubicadas en el archivo .cup:

1. **package umg.compiladores;**: Este es el paquete en el que se encuentra la gramática.

2. **import java_cup.runtime.*;**: Esta línea importa todas las clases del paquete **java_cup.runtime**, que proporciona las clases necesarias para interactuar con el analizador sintáctico generado por CUP.

3. **import java.util.ArrayList;**: Esta línea importa la clase **ArrayList** de **java.util**, que se utiliza para almacenar listas de objetos.

4. **parser code {:**: Este bloque contiene código de usuario que se copia directamente en la clase generada. En este caso, el código define dos listas de objetos y cadenas que se utilizan para almacenar los resultados del análisis.

5. **terminal ...;**: Estas líneas definen los tokens terminales de la gramática, que son los elementos básicos del lenguaje.

6. **non terminal ...;**: Estas líneas definen los tokens no terminales de la gramática, que son las estructuras sintácticas del lenguaje.

7. **start with clase;**: Esta línea establece **clase** como el símbolo inicial de la gramática.

8. **... ::= ...;**: Estas líneas definen las reglas de producción de la gramática, que describen cómo se pueden combinar los tokens para formar estructuras sintácticas. Cada regla de producción tiene la forma **A ::= B C D ...;**, que significa que el token no terminal **A** puede ser reemplazado por la secuencia de tokens **B C D ...**.

9. **{: ... :}**: Estos bloques contienen código de acción que se ejecuta cuando se aplica la regla de producción correspondiente. En este caso, el código de acción generalmente agrega una cadena a la lista **resultados2**.

10. **/* ε */**: Este es un símbolo epsilon, que representa una cadena vacía en la gramática. **clase**: Esta regla define la estructura de una clase. Una clase puede ser pública y contener un método principal (**metodo_main**), o puede contener funcionalidades adicionales antes del método principal.

11. **metodo_main**: Esta regla define la estructura del método principal de una clase. El método principal puede contener funcionalidades adicionales o puede estar vacío.

12. **metodo**: Esta regla define la estructura de un método. Un método puede ser público o privado y tiene un tipo de método asociado (**tipo_metodo**).

13. **parametros**: Esta regla define la estructura de los parámetros de un método. Los parámetros son declaraciones de variables separadas por comas.

14. **tipo_metodo**: Esta regla define la estructura de un tipo de método. Un tipo de método puede ser vacío o puede tener un tipo de dato declarado, y puede contener funcionalidades y/o un método de retorno.

15. **declarar**: Esta regla define los tipos de datos que se pueden declarar en este lenguaje.

16. **funcionalidad**: Esta regla define las diferentes funcionalidades que se pueden utilizar en este lenguaje, como imprimir, escanear, declarar variables, asignar valores, utilizar estructuras de control y más.

17. **asignar**: Esta regla define cómo se asigna un valor a una variable.

18. **aritmetica**: Esta regla define las operaciones aritméticas que se pueden realizar en este lenguaje.

19. **imprimir**: Esta regla define cómo se imprime una salida en este lenguaje.

20. **tipo_imprimir**: Esta regla define los diferentes tipos de impresión que se pueden utilizar en este lenguaje.

21. **estado_bool**: Esta regla define los valores booleanos que se pueden utilizar en este lenguaje.

22. **operador**: Esta regla define los operadores que se pueden utilizar en este lenguaje.

23. **retorno_metodo**: Esta regla define cómo se retorna un valor de un método.

24. **concatenar**: Esta regla define cómo se concatenan las cadenas en este lenguaje.

25. **valor**: Esta regla define los diferentes tipos de valores que se pueden utilizar en este lenguaje.

26. **si**, **si_multiple**, **caso**, **condiciones**, **condicional_num**, **condicional_bool**, **condicional_string**, **repetir**, **mientras**: Estas reglas definen las diferentes estructuras de control que se pueden utilizar en este lenguaje, como if, switch, case, repeat y while.