Autómatas, Teoría de Lenguajes y Compiladores

Trabajo Práctico Especial

Primer Cuatrimestre 2016

JV

Autores:

Juan Pablo Orsay - 49373

Horacio Miguel Gomez - 50825

Daniel Alejandro Lobo - 51171

Juan Marcos Bellini - 52056

Resumen

A modern weapon for a more civilised era

3 de Julio de 2016

Índice

1. Introducción

El objetivo del proyecto es crear un lenguaje procedural simple, sintético, y que fuerce el buen estilo de programación, junto con un compilador para dicho lenguaje.

El lenguaje creado se llama JV. Procura, en todo momento, simplificar la experiencia del programador, para permitir que se concentre en las ideas abstractas. Por otro lado, para aquellas personas que se quieran inicial en el mundo de la programación, ofrece la posibilidad de aprender, con menor fricción, los principales conceptos de la programación, e implementar, con mayor sencillez, sus ideas.

El compilador de JV genera programas objeto que corren como una aplicación Java, dentro de una Java Virtual Machine. Estos programas pueden ser usados en conjunto con cualquier programa en dicho lenguaje.

2. Gramática del Lenguaje

2.1. Definición del Lenguaje

Para correr la version exacta de nuestro programa, es necesario ejecutar por consola los siguientes comandos, una vez ubicados en la raíz del proyecto:

2.2. Constantes y delimitadores

Se decidió utilizar como delimitadores las comillas simples para delimitar el inicio y el final de una cadena de caracteres o *string*.

Las constantes encontradas en este proyecto son números enteros, cadenas de caracteres y el valor análogo a *True* y *False* que serían *yes* y *no* respectivamente.

Todas estas características del lenguaje pueden ser encontradas en el archivo *Scanner.flex* de la carpeta *jflex*.

2.3. Operadores

2.3.1. Aritméticos

Para los operadores relacionales se tomó la decisión de implementar los siguientes: suma, resta, multiplicación y división Todos ellos son los equivalentes a los operadores built-in de Java. Se pueden encontrar listados en el archivo *Scanner.flex* de la carpeta *jflex*.

2.3.2. Relacionales

Para los operadores relacionales se tomó la decisión de implementar los siguientes: menor, mayor, menor igual, mayor igual, igual y diferente. Todos ellos son los equivalentes a los operadores de Java. Se pueden encontrar listados en el archivo *Scanner.flex* de la carpeta *jflex*.

2.3.3. Lógicos

Para las operadores lógicos se tomó la decisión de implementar los siguientes: not, and y or. Todos ellos son los equivales a los operadores de Java. Se pueden encontrar listados en el archivo *Scanner.flex* de la carpeta *jflex*.

2.3.4. De asignación

Par los operadores de asignación se tomó la decisión de implementar la asignación con el símbolo = para la asignación de valor a una variable y el símbolo *:* para la asignación de tipo a una variable. Se pueden encontrar listados en el archivo *Scanner.flex* de la carpeta *jflex*.

2.4. Bloque Condicional

Para la creación de bloques condicionales se tomó la decisión de implementar el *if* y el *else* de Java con los comandos *if* y *ls*. Buscando siempre ahorrarle tiempo y claridad al programador al momento de usar estas estructuras. Se pueden encontrar listados en el archivo *Scanner.flex* de la carpeta *jflex*.

2.5. Bloque Do-While

Para la creación de bloques do-while se tomó la decisión de implementar el *do* y el *while* de Java con los comandos *do* y *whl*. Se pueden encontrar listados en el archivo *Scanner.flex* de la carpeta *jflex*.

2.6. Un bloque principal de ejecución

Si bien inicialmente se había tomado la decisión de crear un bloque principal o *main* para la creación del programa, se decidió ahorrarnos este bloque. La razón detrás de esta decisión fue, nuevamente, más claridad para el programador y simplificar la cantidad de tabs en el código. Por lo tanto, no contamos con un bloque principal de ejecución.

2.7. Mecanismos de entrada y salida

Se cuenta con comandos para leer hasta el final de una línea y para escribir una línea incluyendo el *carriage return* al final de la misma. Adicionalmente, se cuenta con un comando para poder escribir sin incluir el *carriage return* luego de los caracteres ingresados. Se pueden encontrar listados en el archivo *Scanner.flex* de la carpeta *jflex*.

2.8. Tipos de datos

Se tomó la decisión de contar simplemente con tres tipos de datos que, a su vez, consideramos esenciales para un lenguaje de programación. Los tipos son los análogos a *Integer*, *Boolean* y *String* de Java. Se pueden encontrar listados en el archivo *Scanner.flex* de la carpeta *jflex*.

3. Compilador

3.1. Analizador Léxico

La primera herramienta utilizada es JFlex. JFlex es un generador de analizadores léxicos – o también conocido como generador de *scanners* – para Java. Cuando recibe una entrada, JFlex se encarga de transformar dicho *stream* de caracteres, en una cadena de *tokens* predefinidos. La ventaja de utilizar JFlex es su integración con CUP, un generador de *parsers* también para Java. Otra ventaja de JFlex es el hecho de poder extender su funcionalidad fácilmente, ya sea mediante código Java *inline*, o importando clases externas.

Una de las características más notorias de esta herramienta es la capacidad de definir estados de *lexing*. En este caso se definieron 4: YYINITIAL, NORMAL, STRING y FINAL.

Cada vez que comienza una línea, el *lexer* entra en el estado YYINITIAL. Dicho estado tiene la tarea de contar cuántas veces aparece el caracter ’ ’, y en base a eso, decidir si tiene que generar el *token* **<indent>** ó **<dedent>**.

En este estado, cuando aparece cualquier otro caracter – que no sea un final de línea – el *lexer* cambia su estado a modo NORMAL, en el cual convierte el texto en los correspondientes *tokens*. Al llegar al final de la línea, se vuelve al estado inicial.

Otro estado importante es STRING. Para ingresar en este estado, el *lexer* debe haber leído el caracter ’"’. Dentro del mismo, se consume el texto, hasta llegar nuevamente al caracter ’"’. A partir de eso, crea el *token* **<string>**, que tiene un atributo cuyo valor es el texto entre las comillas.

Finalmente, el estado FINAL es un estado que sirve para completar las *des-indentaciones* que pudieron haber quedado al final del código.

3.2. Analizador Sintáctico

Como analizador semántico, se decidió utilizar CUP (*Construction of Useful Parsers*), el cual es un generador de *parsers* para Java.

En CUP se define la gramática del lenguaje. Se declaran símbolos terminales y no terminales, y se crean las producciones de la misma. Con ella, crea el *parser* que va a aceptar cadenas que pertenezcan al lenguaje. Obviamente, los símbolos terminales son los mismos que generó previamente JFLEX.

Además, otra *feature* de CUP es la capacidad de definir reglas semánticas para el lenguaje. De esta manera, al *parsear* la cadena, se pueden tomar distintos tipos de decisiones, que ayudan a la hora de generar el programa objeto.

4. El futuro de JV

"Any sufficiently advanced technology is indistinguishable from magic."
Arthur C. Clarke

Creemos que con el apoyo de los primeros usuarios interesados en este lenguaje, se puede expander las capacidades del mismo.

Se pueden implementar más tipos de variables y aún más variaciones de diferentes controles de flujo.

Otra idea que se nos ocurre es permitir tener argumentos variables para una función.

5. Conclusiones

Consideramos que el trabajo, mayoritariamente, nos ayudo a reforzar nuestro manejo y seguimiento de llamadas recursivas. Junto a esto, creemos que fue muy valiosa la toma de decisiones continuas, no solo sobre la estrategia general para encarar estos problemas sino tambien para las decisiones puntuales en las distintas instancias de la creacion del algoritmo.