Instituto Tecnológico de Costa Rica

Ingeniería en Computación

Compiladores e intérpretes

Tarea I: Análisis sintáctico

Profesor: Óscar Víquez

Estudiantes:

Kenneth Sancho, carnet 200941125

Luis Alonso Vega, carnet 201042592

Lunes, 30 de abril 2012

Análisis del lenguaje

El objetivo de la tarea es generar un analizador sintáctico para un subconjunto del lenguaje de programación Java. Sólo se evalúa una pequeña parte de las características, incluyendo algunas palabras reservadas básicas y agregando una función fija de imprimir (System.out.println). Los símbolos literales incluyen cadenas de caracteres, valores booleanos y números enteros no negativos.

El modelo de lenguaje analizado permite revisar un archivo con un código fuente formado por una clase principal la cual posee un comando de imprimir una expresión. Adicionalmente se pueden agregar definiciones de clases luego de la que contiene el método principal.

Dentro del lenguaje se pueden utilizar declaraciones, asignaciones, condicionales (if, switch), ciclos (while únicamente), envíos de mensajes a clases y comparaciones. Las declaraciones funcionan con tipos de arreglos, también, sin embargo no se puede procesar algunas operaciones (como asignaciones) en ellos.

Una de las características más fuertes del lenguaje es que pueden formarse expresiones a partir de muchas subcategorías, por ejemplo con operaciones artiméticas o booleanas, con mensajes, o incluso con operaciones.

Soluciones e implementación

JFLEX y Scanner

Para el desarrollo del Scanner es preciso utilizar la herramienta JFlex como procesador de lenguaje, donde a partir de un archivo *.flex* se genera el Scanner, en el flex se contiene las configuraciones para verificar si cada token existe, además de guardar información necesaria sobre cada token para luego devolverlo al parser.

El Scanner como antes se mencionó es el encargado de verificar si los tokens utilizados en el “código fuente” estan soportados en nuestro compilador en caso de que estos no estén soportados, el Scanner es el que debe reportar el error de sintaxis.

Para el presente proyecto se utilizó esta herramienta, agregamos los símbolos y reglas correspondientes a las definidas y utilizadas en nuestra declaración de la gramatica, en la etapa de diseño.

CUP y Parser

El archivo CUP es un procesador de lenguaje necesario para que nos genere el Parser y el Sym. Este último necesario para que el Scanner verifique los tokens que son palabras reservados. El archivo CUP además de configuraciones propias necesarias en el parser, contiene la gramática en BNF, que va a seguir nuestro compilador.

El Parser se encarga de descomponer el código en los diferentes tokens de acuerdo a la gramática BNF, luego se los envía uno a uno al Scanner para verificar que existan, si todo esta correctamente estructurado se compila y se construye el árbol en AST, en caso contrario se reporta a los usuarios los errores que se hayan encontrado.

AST

Se generó una serie de clases que conforman una jerarquía la cual representa la estructura de símbolos (expresiones, declaraciones, etc.) del lenguaje. Entre las clases que se encuentran en esta sección o paquete, están las clases abstractas que definen espacios de código que pueden aparecer en distintas formas, así como las concretas que definen una estructura específica para cada token del código fuente. También se encuentra en este paquete la especificación de la clase de visitor.

Editor

Finalmente se generó un editor simple para proporcionar al usuario la capacidad de escribir el código, guardar y abrir archivos, de compilarlo y generar el árbol de análisis sintáctico. Se conforma de una ventana que muestra una sección para insertar el código, un control de árbol que permitirá visualizar la estructura AST una vez que se ha generado, una zona para mostrar mensajes importantes sobre errores sintácticos y operaciones de archivos.
El menú principal de la ventana posee opciones de abrir y guardar archivos y de salir de la aplicación. También posee un menú para compilar el texto, el cual como resultado muestra la estructura de árbol y los mensajes importantes.

Nota: Debido a un problema particular con el JFLEX, tuvimos que sustituir el caracter de dos puntos, por el mismo pero encerrado entre comillas simples. Por tanto su uso queda limitado al siguiente formato:

```
case literal ’:’ ...
```

Resultados obtenidos

- Archivo .Cup (Generador de Parser y Sym) desarrollado con éxito.
- Archivo .Flex (Generador de Scanner) desarrollado con éxito.
- Creación editor, desarrollado con éxito.
- Creación de nuestra propia excepción (MyException), desarrollado con éxito.
- Creación del AST, desarrollado con éxito.
- El despliegue del árbol utilizando GUI, desarrollado con éxito.

Conclusiones

Al trabajar este proyecto nos enfrentamos a varios problemas y tuvimos que resolverlos de la manera adecuada, además de investigar un poco y repasar mucho las reglas y conceptos aprendidos en clase. Entre otros puntos podemos mencionar los siguientes:

- Para generar un analizador sintáctico es necesario conocer y diferenciar muy bien los conceptos de Scanner y Parser.
- Una vez entendidos, se necesita aprender a crear correctamente los archivos de CUP y JFLEX, utilizando bien las propiedades y métodos necesarios para mostrar errores, así como una estructura gramatical bien definida, reducida y simplificada.
- Las reglas gramaticales deben tener nombres con cierto estándar definido si se quiere evitar problemas para recordarlos luego.
- Los visitors deben crearse considerando siempre la forma en que se mostrará el resultado al final. En nuestro caso fue necesario hacer uso de una estructura de nodo de árbol para luego pasarla como modelo al control que lo mostraría.
- Se debe tener un buen control de la aparición de errores y excepciones, sin dejarlas escapar ni ocultarlas por completo del usuario, a quien le resultan útiles cuando se quieren entender los problemas de sintaxis. En nuestro caso se utilizó la zona de mensajes para mostrar cualquier problema de sintaxis.
- Cualquier manejo de archivos necesita un mínimo conocimiento de trato de I/O.

Bibliografía

- Georgia Institute of Technology. (Julio de 1999). Recuperado el 28 de Marzo de 2012, de CUP User's Manual: http://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html
- SCRIBD, (sf). Swing y JFC (Java Foundation Classes), Visitado el 20 de abril del 2012 de http://es.scribd.com/trukly/d/7545519-Manual-JAVA-Swing