TÉCNICAS DE COMPILACIÓN

2023

Profesor : **Maximiliano A. Eschoyez**
Alumnos: **Calcara Juan Pablo - Millia Jorge**
Fecha: **08/02/23**

CONSIGNA

El objetivo de este Trabajo Final es mejorar el filtro generado en el Trabajo Práctico Nro. 2. Dado un archivo de entrada en C, se debe generar como salida el reporte de errores en caso de existir.

Para lograr esto se debe construir un parser que tenga como mínimo la implementación de los siguientes puntos:

- Reconocimiento de bloques de código delimitados por llaves y controlar balance de apertura y cierre.
- Verificación de la estructura de las operaciones aritmético/lógicas y las variables o números afectadas.
- Verificación de la correcta utilización del punto y coma para la terminación de instrucciones.
- Balance de llaves, corchetes y paréntesis.
- Tabla de símbolos.
- Llamado a funciones de usuario.

Si la fase de verificación gramatical no ha encontrado errores, se debe proceder a:

1. Detectar variables y funciones declaradas pero no utilizadas y viceversa,
2. Generar la versión en código intermedio utilizando código de tres direcciones, el cual fue abordado en clases y se encuentra explicado con mayor profundidad en la bibliografía de la materia.

En resumen, dado un código fuente de entrada el programa deberá generar un archivo de salida:

1. La versión en código de tres direcciones con las etiquetas de bloque.

Presentación del Trabajo Final

Código Fuente

El código fuente generado para este proyecto y la versión digital del informe en PDF deberán entregarse a través del enlace correspondiente en la plataforma MiUBP. En dicho enlace se deberá subir un único archivo en formato ZIP conteniendo todos los código fuente que se requieran para la realización del trabajo final y el informe.

Informe Escrito

Se entregará al profesor un informe escrito (en versión digital formato PDF) donde se debe describir la problemática abordada en el trabajo final, el desarrollo de la solución propuesta y una conclusión. El texto deberá ser conciso y con descripciones apropiadas. No se debe incluir el código fuente, sino los textos necesarios para realizar las explicaciones pertinentes.

DESARROLLO

Se desarrolló la solución, usando como lenguaje JAVA con la api de ANTLR4 (generador de analizadores para leer, procesar, ejecutar o traducir texto estructurado o archivos binarios).

Se generó un grammar file Reglas.g4 que contiene todas las reglas para poder identificar a cada uno de los tokens usando expresiones regulares.
Allí se definió la regla de entrada “programa” que es ejecutada por el parser desde el App.java donde se realiza la ejecución principal del programa.

Desde App.java se levanta un ejemplo.txt que contiene código en lenguaje C, se lee este código y se genera una secuencia de tokens con lo que luego se va generando un árbol de análisis, se le pasa el árbol al método visit si no se produce error se va generando el código de hasta 3 direcciones y se imprime por consola, a la vez que se genera desde la clase MiVisitor.java un archivo txt con el código intermedio con las etiquetas de bloque.

También se genera en la salida el reporte de errores en caso de producirse alguno marcando la línea donde ocurre y el tipo de error.

Reglas

La regla inicial ejecuta la regla instrucciones, que puede contener una instrucción seguida de más instrucciones y a su vez un declaración, bucles o un bloque de código.

La declaración de función, permite declarar una función de dos formas:
-Por declaración o implementación lo cual conlleva nombre de los parámetros y un bloque de código.
-Por prototipo de función, la cual tiene en cuenta el tipo, los parámetros y finaliza con el punto y coma.

Llamado de función verifica sintaxis, nombre y parámetros.

Bloque es un bloque de código con instrucciones alli se controla el balance de llaves.

Tabla de Símbolos

La tabla de símbolos registrará todos los identificadores que se utilizan a lo largo del proceso de compilación del programa, tanto de variables o de funciones esta tabla va tener el nombre de las variables, funciones y asociado el tipo de dato que corresponde en el caso de variables y el tipo de dato que devuelve o la lista de argumentos con sus tipos de datos que recibiría una función Para este caso se crea

una lista enlazada de HashMaps del tipo . Donde ID es una clase que representa a una variable o a una función. Cada vez que se abre una llave se crea un nuevo contexto, es decir, se agrega un nuevo HashMap a la lista enlazada. Para el caso de declarar una variable y/o función se usa un método de la Tabla de Símbolos implementada que permite ver si ésta ya ha sido declarada en el contexto actual o en alguno de los contextos padres del mismo para reportar el error.

Código Intermedio

Para este caso se utilizó MiVisitor que una vez que se genera el árbol, a medida que se va visitando se va generando el código de hasta 3 direcciones.
Para declaración/asignación de variable se toma el nombre de la misma y a partir de ese punto se trabaja con los factores a asignar que puede llevar operaciones.

Para definicion de funcion se toma el nombre de la función y las cláusulas BeginFunc y EndFunc, entre ellas se visitan todos los nodos hijos y se analiza

CONCLUSIÓN

El desarrollo de la materia y del trabajo nos permitió entender la estructura de un compilador, las etapas y cómo se realiza el análisis del código cada vez que compilamos un programa.