Universidad Nacional de Córdoba

Facultad de Ciencias Exactas, Física y Naturales

Práctica y Construcción de Compiladores — Trabajo Final

Docentes:

- Maximiliano Andres Eschoyez.

Alumno:

- Ciarrapico, Nicolás

Índice

Objetivos **3**

Desarrollo **3**

Conclusiones **4**

Objetivos

El objetivo de este Trabajo Final es completar las funcionalidades del compilador realizado durante el cursado. El programa a desarrollar tiene como objetivo tomar un archivo de código fuente en C, generar como salida una verificación léxica, gramatical y semántica reportando errores en caso de existir, y generar código intermedio.

Desarrollo

El código fuente C de entrada al proyecto deberá estar disponible en el directorio ./input para poder realizar su compilación.

El archivo compilador.g4 contiene todas las reglas gramaticales generadas para realizar el parseo del código fuente C de entrada.

La clase SymbolsTable representa la tabla de símbolos del sistema. En se generara una entrada por cada variable o función que se defina en el código fuente C a compilar, almacenando además los datos necesarios de dicha entrada para el posterior control semántico.

La clase App representa el punto de entrada al proyecto. Es la clase encargada de instanciar al resto de las clases y llamar las funciones necesarias para llevar adelante el proceso de compilación propuesto.

La clase MiListener es la implementación propia del proyecto de la interfaz ParseTreeListener de ANTLR. Esto se logra extendiendo la clase BaseListener que ANTLR genera.

MiListener es la clase encargada de realizar la verificación gramatical y semántica del código fuente de entrada. Esto se logra realizando un recorrido descendente durante la construcción del árbol que nos genera ANTRL al parsear el archivo de entrada.

Durante la verificación gramatical y semántica realizada por MiListener se analizan 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, controlando que las variables afectadas hayan sido previamente definidas en el código.
- Verificación de la correcta utilización del punto y coma para la terminación de instrucciones.
- Balance de llaves, corchetes y paréntesis.
- Llamado a funciones de usuario, controlando que:
    - La función a llamar haya sido previamente definida (exista la entrada correspondiente en la tabla de símbolos).
    - Que la cantidad de parámetros sea la correcta.
    - Que el tipo de dato pasado corresponda con el tipo esperado para el parámetro.

Si el análisis sintactico y semantico realizado por MiListener no presenta errores, App instanciara la clase MiVisitor para realizar la traduccion del código fuente de entrada a código de 3 direcciones.
En esta clase se realiza un nuevo recorrido del árbol ya construido por ANTRL, pero esta vez a demanda, lo que permite interpretar las instrucciones del código fuente de entrada, y y generando las variables y etiquetas necesarias para obtener una traducción de la misma a código de 3 direcciones.

Una vez traducidas todas las instrucciones del código fuente de entrada, el código intermedio obtenido se almacena en un nuevo archivo en el directorio ./output.

Por último, con el nuevo código intermedio generado es posible realizar las optimizaciones de código estudiadas durante el cursado de la materia.

Estas tareas de optimización son realizadas en la clase MiOptimizer y son las siguientes:

- Reemplazo de constantes: aquellas variables temporales que almacenan el valor de una constante (o el resultado de una operación entre dos constantes), serán reemplazadas dentro del código por el valor de dicha constante.
- Reducción de asignaciones: aquellas operaciones de asignación donde se utilice una variable temporal para almacenar el resultado de una operación y posteriormente copiar este valor a una variable del programa, serán reemplazadas por una sola operación de asignación donde se eliminará la variable temporal y se asignará directamente el valor de la operación a la variable del programa en cuestión.
- Reducción de operaciones: Si múltiples variables almacenan el resultado de una misma operación, se tomará valida solo la primera de ellas y se eliminaran las demás reemplazando su nombre por el de la primera en todas las operaciones donde se referencie su valor.

Finalmente el nuevo código intermedio optimizado resultante se encontrará disponible en un nuevo archivo en el directorio ./output.

Conclusiones

Con el presente trabajo fue posible poner en práctica los conocimientos adquiridos durante el cursado de la materia así como también profundizar en las bibliografías propuestas afianzando así el manejo de las herramientas.

Durante el desarrollo del proyecto se enfrentaron múltiples desafíos propios de la industria que se lograron sortear de manera satisfactoria gracias a la tutoría del docente a cargo.

Finalmente considero tener al día de hoy un mejor entendimiento de cómo funciona un compilador, las tareas que se realizan por detrás de la compilación de un código fuente, los problemas a los que se enfrenta un desarrollador de herramientas de compilación y el trabajo que requiere obtener un compilador que genere un código ejecutable de calidad.