Compilador de C

Universidad de la Defensa Nacional

Centro Regional Universitario Córdoba IUA

Facultad de Ingeniería

Carrera: Ingeniería en Informática

Asignatura: Desarrollo de Herramientas de Software

Profesor: Ing. Maximiliano Eschoyez

Año: 2022

Equipo de Trabajo

Camargo Tenaglia, Gaston Matias

García Bidart, Aurelio

Índice

Consigna

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 afectados.
- 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 los siguientes archivos de salida:

1. La tabla de símbolos para todos los contextos,
2. La versión en código de tres direcciones del código fuente,

Marco teórico

ANTLR

ANTLR es una herramienta creada principalmente por Terence Parr, que opera sobre lenguajes, proporcionando un marco para construir reconocedores (parsers), intérpretes, compiladores y traductores de lenguajes a partir de las descripciones gramaticales de los mismos (conteniendo acciones semánticas a realizarse en varios lenguajes de programación).

Gramática

La gramática es el estudio de las reglas y principios que gobiernan el uso de las lenguas y la organización de las palabras dentro de unas oraciones y otro tipo de constituyentes sintácticos. También se denomina así al conjunto de reglas y principios que gobiernan el uso de una lengua concreta; así, cada lengua tiene su propia gramática.

Analizador léxico

Un analizador léxico o analizador lexicográfico es la primera fase de un compilador, consistente en un programa que recibe como entrada el código fuente de otro programa (secuencia de caracteres) y produce una salida compuesta de tokens (componentes léxicos) o símbolos. Estos tokens sirven para una posterior etapa del proceso de traducción, siendo la entrada para el analizador sintáctico (en inglés parser).
La especificación de un lenguaje de programación a menudo incluye un conjunto de reglas que definen el léxico. Estas reglas consisten comúnmente en expresiones regulares que indican el conjunto de posibles secuencias de caracteres que definen un token o lexema. En algunos lenguajes de programación es necesario establecer patrones para caracteres especiales (como el espacio en blanco) que la gramática pueda reconocer sin que constituya un token en sí.

Analizador sintáctico

El Analizador sintáctico analiza una cadena de símbolos según las reglas de una gramática formal. Convierte el texto de entrada en otras estructuras (comúnmente árboles), que son más útiles para el posterior análisis y capturan la jerarquía implícita de la entrada.

Analizador semántico

El analizador semántico utiliza el árbol sintáctico y la información en la tabla de símbolos para comprobar la consistencia semántica del programa fuente con la definición del lenguaje. También recopila información sobre el tipo y la guarda, ya sea en el árbol sintáctico o en la tabla de símbolos, para usarla más tarde durante la generación de código intermedio.

Código de tres direcciones

En ciencias de la computación, el código de tres direcciones es un lenguaje intermedio usado por compiladores optimizadores para ayudar en las transformaciones de mejora de código. Cada instrucción TAC tiene a lo sumo tres operandos y es típicamente una combinación de asignación y operador binario. Por ejemplo, $t1 := t2 + t3$. El nombre proviene del uso de tres operandos en estas declaraciones aunque instrucciones con menos operandos pueden existir.

Ya que el código de tres direcciones es usado como un lenguaje intermedio en los compiladores, los operandos normalmente no contendrán direcciones de memoria o registros concretos, sino que direcciones simbólicas que serán convertidas en direcciones reales durante la asignación de registros. Es también común que los nombres de los operadores sean numerados secuencialmente ya que el código de tres direcciones es típicamente generado por el compilador.

Procedimiento

Para poder desarrollar la consigna se declaran las reglas gramaticales, se implementa un analizador léxico, un analizador sintáctico y un analizador semántico.

Para las reglas gramaticales se decide utilizar etiquetas intuitivas para la fácil comprensión.

Para el analizador léxico se crea el "Listener" correspondiente, en este caso *MyListener*, encargado de cargar la *Tabla de Símbolos* a través de los eventos generados.

La tabla de símbolos es una clase que contiene una lista de diccionarios en la cual se cargan y eliminan los contextos a medida que se va leyendo el código C. En cada uno de estos contextos se almacenan las variables correspondientes, las cuales serán seguidas a lo largo de la lectura para verificar que hayan sido inicializadas y utilizadas. Finalmente, cuando el código C sea "compilado" completamente, se genera un archivo llamado "tablaSimbolos.txt" en la cual se encontrarán las variables inicializadas al momento de salir de cada contexto del programa. De esta forma se obtiene un reporte de las variables, el contexto en que fueron creadas y en el momento en que se destruyen debido a la destrucción del contexto en que se encontraban.

Para implementar el último ítem de la consigna, el generar el código intermedio a partir del programa en C dado, se utiliza el "Visitor", a través de la clase *MyVisitor.* De esta forma se puede recorrer el árbol generado por el archivo "Compiladores.g4".

El principal objetivo de esto es avanzar sentencia a sentencia, haciendo comprensión de lo codificado en lenguaje C, traducirlo a código intermedio y plasmar el resultado en un nuevo archivo, el cual tiene el nombre de "CodigoIntermedio.txt". En este archivo se encuentran plasmados los saltos producidos por las funciones de C, los algoritmos utilizados y la forma en la que el código debe avanzar según cada instrucción. También se verán plasmadas las operaciones aritmético-lógicas realizadas sobre las variables.

Tal como se vio en la teoría, este nuevo código de tres direcciones es un paso intermedio entre el código C y el assembler, siendo este último el lenguaje comprendido por el procesador para realizar operaciones.

Resultado

Al finalizar el desarrollo del proyecto, se pudieron observar que los objetivos planteados se cumplieron y también se logró hacer un mayor entendimiento de cómo es que trabajan los compiladores que utilizamos en el dia a dia. Cuando programamos hay muchas cosas que damos por hecho, pasamos por alto o simplemente las entendemos como "las cosas son así porque sí", pero en realidad detrás del código hay analizadores que realizan distintas tareas, de formas lógicas que uno al programar no se plantea. Pero luego de hacer este trabajo práctico, luego de incorporar el conocimiento sobre los compiladores podemos entender que hay reglas que se cumplen por simple metodología o costumbre, y otras que hasta se puede aprovechar su potencial para lograr hacer cosas más óptimas, simples y efectivas.

También, luego de hacer la transformación de código C a código de tres direcciones se comprendió que hay muchas operaciones que en un nivel relativamente alto parecen sencillas, sin demasiado costo computacional, mientras que cuando estas se representan en código de tres direcciones para luego ser pasadas a assembler su complejidad aumenta y esto lleva al procesador a realizar muchas más operaciones de las que uno en un principio podría pensar.

Repositorio

https://github.com/sharkymid/ANTLR4cCompiler