Instituto Universitario Aeronáutico

Facultad de Ingeniería

DESARROLLO DE HERRAMIENTAS DE SOFTWARE
EXAMEN FINAL

Alumnos:

Manfredi Angelo

Yanes Kevin

Profesor:

Ing. Maximiliano Eschoyez

2022

Índice

Consigna

Dado un archivo de entrada en lenguaje C, se debe generar como salida el Árbol Sintáctico (ANTLR) correcto. Para lograr esto se debe construir un parser que tenga
como mínimo la implementación de los siguientes puntos:

- Reconocimiento de bloques de código delimitado 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 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.

Conceptos teóricos

ANTLR

Es una herramienta de reconocimiento de lenguaje 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

Una gramática proporciona una estructura a un lenguaje de programación, siendo más fácil generar código y detectar errores. Es más fácil ampliar/modificar el lenguaje si está descrito con una gramática. La mayor parte de este tema está dedicada a los métodos de análisis sintáctico de uso típico en compiladores.

Analizador léxico

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*).

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 es el que emplea la información contenida en los atributos de los componentes léxicos. Recuerda que el sintáctico sólo ve las etiquetas de las categorías. En cuanto a error, suponemos que representa las acciones necesarias para tratar el error.

Código intermedio

El código intermedio es una representación intermedia del código fuente original de un programa en un lenguaje de alto nivel. Se genera durante la fase de análisis sintáctico de un compilador y su función es servir de base para la generación del código objeto final, que es el código ejecutable por la máquina. El código intermedio suele ser un lenguaje de bajo nivel, más cercano a la estructura interna de la máquina, y se utiliza como un formato común en el que se pueden aplicar diversas optimizaciones al código fuente original antes de generar el código objeto final. Para implementar el código intermedio se utilizó el código de tres direcciones.

Código de 3 direcciones

El código de tres direcciones es un tipo de código intermedio que utiliza tres operandos en cada instrucción. Cada operando puede ser una variable, una constante o una dirección de memoria, y cada instrucción representa una operación matemática o lógica que se aplica a los operandos. El código de tres direcciones se utiliza en muchos compiladores para facilitar el proceso de optimización del código y para permitir una mayor facilidad en la generación del código objeto final.

Procedimiento

Primero se declaró reglas gramaticales, se implementó un analizador léxico , un analizador sintáctico , un analizador semántico y la generación de código intermedio mediante el uso de código de tres direcciones.

En las reglas gramáticales se tuvo en cuenta que el nombramiento sea intuitivo y simple de entender.
Para ver dichas reglas definidas:
https://github.com/angelo59930/compiladores/blob/main/src/main/python/compiladores.g4

Para el analizador léxico creamos el listener correspondiente (en nuestro caso lo llamamos "MyListener") y lo que realiza es tratar los tokens que nos interesa para guardarlos en una tabla de símbolos.
La tabla de símbolos es una clase en la cual se encuentra un arreglo de diccionarios/mapas y vamos a guardar los distintos contextos que posee el archivo que estamos compilando y las variables que se encuentran declaradas en cada contexto. Para guardar estas variables vamos a crear más clases como la de Id, Variable y Función la cual nos van a ayudar a poder analizar correctamente el archivo. Una vez compilado el código fuente lo que ocurre es que el analizador léxico detecta variables que no existan y que estas hayan sido utilizadas o no, en caso detecte dichas situaciones, avisa su problema y muestra cual es la variable que tiene ese conflicto (para variables inexistentes es un "ERROR" y para las variables inutilizadas son un "WARNING").
También se realizó una implementación que logra escribir en un archivo todo el ciclo de vida de las variables en el momento que se sale de un determinado contexto, este archivo de salida se lo nombró como "TablaDeSimbolos.txt"

Para el desarrollo del código intermedio se utilizó la herramienta de código de tres direcciones , el programa fue creado mediante los "Visitors" correspondientes y para esto creamos una clase que nombramos "MyVisitor" y lo que recibe es el árbol definido en nuestra gramática.
La utilidad de este código de tres direcciones es tomar el código fuente y separar las operaciones complejas en 3 operandos y luego asignarse a un resultado (resultado = op1 operación op2) tal como se vió en el apartado de conceptos teóricos. Dicho código contempla saltos de línea o *jumps* (por ejemplo cuando realiza el salto de un prototipo de función hacia la implementación de esa función) y operaciones de una ALU (unidad aritmética lógica), así mismo se realizó una implementación que escribe en un archivo el código intermedio, dicho archivo fue nombrado como "CodigoIntermedio.txt".

Conclusión

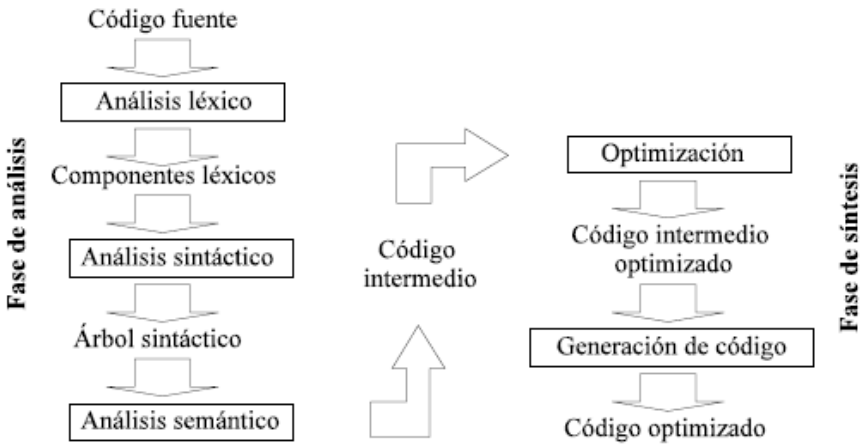
Se logró construir un compilador que cumple con las consignas propuestas y tener una noción de que es lo que ocurre detrás del proceso de compilación de un código fuente y como este pasa por distintos procesos hasta llegar al código objeto. También se incorporó el conocimiento de un nuevo lenguaje de programación que suma como experiencia a nuestro crecimiento profesional y poder sacar la conclusión de que no importa cual sea el lenguaje propuesto, con las bases sólidas se puede dominar cualquier lenguaje independientemente de su sintaxis.

Apreciación Personal

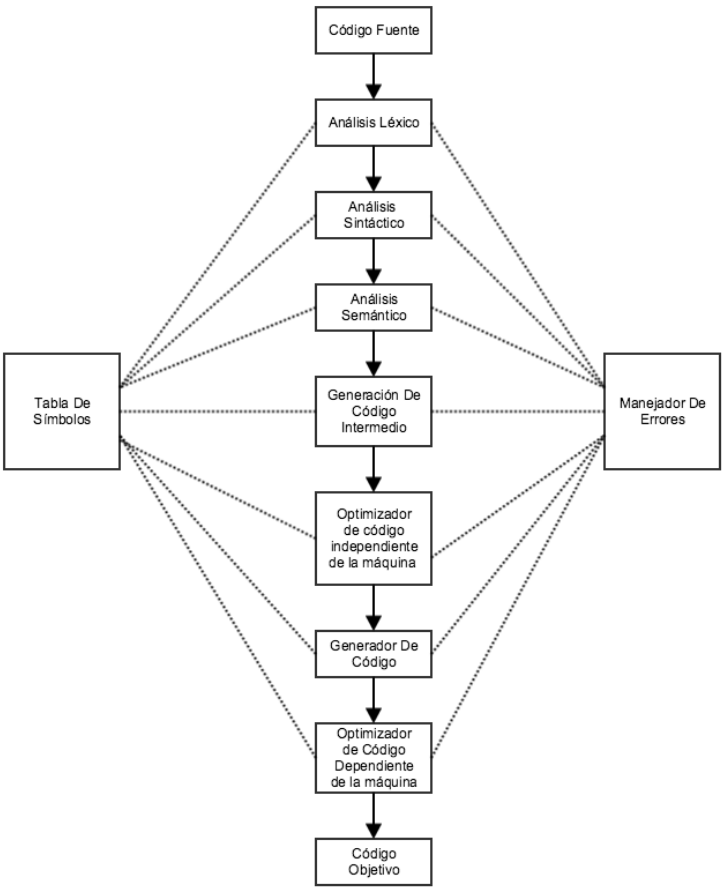
Al principio de la materia suponíamos cómo un compilador toma el código que uno escribe y nos devuelve los errores,avisos,etc. Más no resultó ser de esa manera y tampoco funcionar como "por arte de magia" ya que posterior a la materia pudimos tener un conocimiento real de lo que ocurre dentro de un proceso de compilación y así mismo llevarnos la idea lógica de mejoría acerca cómo solucionar distintos tipos de problemas desde otro punto de vista a fines de darle menos trabajo al compilador, y sumar mejores prácticas de código limpio.

Anexo

Se tomó como referencia las siguientes imágenes durante el proceso de compilación:



Etapas de traducción de un compilador: análisis y síntesis



Repositorio

- GitHub: https://github.com/angelo59930/compiladores