Gramática & Compilador

72.39 Autómatas, Teoría de Lenguajes y Compiladores
Trabajo Práctico Especial

Grupo YMCA

Ximena Zuberbuhler --------- 57287
Sebastián Favaron ---------- 57044
Rodrigo Navarro ---------- 57307
Joaquin Ormachea ---------- 57034

Introducción

Para dar un cierre práctico a la materia, se realizó un trabajo práctico especial intentando volcar los conocimientos, teóricos y prácticos, aprendidos a lo largo del curso en un desarrollo funcional de un compilador.

Para la implementación del mismo dividimos el trabajo en 2 partes. Primero la definición de una gramática que producirá el lenguaje deseado. Y segundo, un compilador que realizará la transformación del lenguaje producido y así validará que este esté bien formado para luego entregar el mismo en el lenguaje ya conocido, C.

Propuestas contempladas

Empezamos a realizar el trabajo enfocándonos en la idea de obtener un compilador creativo que permita hacer alguna tarea en particular como permitir escribir GRAFOS de operaciones y que a partir de esto se puedan desarrollar programas. Al ver que esta idea podría ser interesante pero poco útil decidimos optar por expandir la funcionalidad de algún lenguaje ya conocido como C.

Esto nos llevó a pensar que estaría bueno tener un tipo de variable matriz de integer que permita guardar información en forma de tabla de una manera sencilla por lo que decidimos agregar esta funcionalidad al lenguaje C.

Implementación

Como mencionamos antes, dividimos el trabajo en 2 secciones. Por lo que la implementación se realizó de la misma forma.

Gramática

Como pidió la cátedra la creación de la definición de la gramática que produce el lenguaje para nuestro compilador consiste en:

- Tipos de datos:
    - Normales:
        - Numérico: int
        - Cadenas: string
    - Compuestos:
        - Matrices genéricas: matrix
- Constantes
    - Define
- Delimitadores
- Operadores aritméticos:
    - +
    - -
    - *
    - /

    - %
- Operadores relacionales:
    - <
    - >
    - <=
    - >=
    - ==
    - !=
- Operadores lógicos:
    - &&
    - ||
- Operadores de asignación:
    - =
- Bloque condicional
    - If
    - Else
- Bloque repetición:
    - While
    - For

Como mencionamos en la introducción y se puede ver en esta definición, tenemos un nuevo tipo de dato “matrix” que consiste en un tipo de dato especial que nos va a permitir definir una matriz de ancho W y altura H y guardará de datos del tipo integer.

Descripción del compilador

En esta sección vamos a explicar cómo desarrollamos el compilador adecuado para aceptar nuestro lenguaje. Comenzamos tomando como referencia lo recomendado en el inciso 3.1 Yet Another Compiler Compiler por lo cual utilizamos Yacc como compilador para nuestro compilador.

Primero generamos el archivo YACC que describe cada producción para nuestros símbolos no terminales desarrollados en LEX.

Para evitar el manejo lógico dentro de las reducciones, decidimos modularizar su comportamiento en funciones asociadas a cada parte del árbol. Dentro de estas mismas funciones se implementó el manejo de errores para presentar al usuario un mensaje descriptivo del mismo.

El manejo de entrada de datos es realizado por stdin porque nos parecio comodo para el uso que le íbamos a dar al compilador. Mientras que el mecanismo de salida que elegimos fue un archivo cuyo nombre definimos como “compiled.c”.

Dificultades

En general no nos encontramos con muchas dificultades ya que seguimos los primeros capítulos del libro “Lex && Yacc” y nos parecio muy util. Podríamos decir que lo más difícil fue encontrar una idea que nos atraiga hacer.

Con respecto a la implementación elegida, nos surge un único conflicto con el manejo del árbol de parseo en el momento de declarar y asignar una variable en una única producción. Pero afortunadamente encontramos una forma de solucionarlo ya que esto imposibilitaba la declaración de una variable para usar dentro de un for y nos parecía importante.

Conclusión y Futuro

Como expansión del trabajo nos quedó pendiente hacer que las matrices puedan ser genéricas o al menos más versátiles. De esta forma el lenguaje tendria mas potencia e incluso pensar como utilizar operadores con distintos tipos de datos. Esto no lo logramos realizar por falta de tiempo ya que había que definir las operaciones para todo tipo de dato.

El trabajo nos pareció entretenido ya que es algo útil y muy versátil para adaptar a todos tipo de aplicaciones. Lo que no fue de nuestro agrado es que la materia ya tiene 3 parciales y los recuperatorios se terminan pisando con la entrega del trabajo y por motivos especiales de esta ocasión, con varios finales.

Estaria bueno que el último parcial se pueda combinar con el trabajo práctico especial y que este pueda tener más importancia.

Referencias

https://theswissbay.ch/pdf/Gentoomen%20Library/Misc/O%27Reilly%20Lex%20and%20Yacc.pdf
https://efxa.org/2014/05/25/how-to-create-an-abstract-syntax-tree-while-parsing-an-input-stream/