Informe Trabajo Práctico

72.39 - Autómatas, Teoría de Lenguajes y Compiladores

2° Cuatrimestre 2018

Benenzon, Nicolás(57539)

Soracco, Tomás(56002)

Dallas, Tomás(56436)

Atar, Marcos Ariel(57352)

Índice

Objetivo

El objetivo de este trabajo fue implementar un compilador para un lenguaje de programación que inventamos nosotros, *natt* (Nicolás, Ariel, Tomás, Tomás), un lenguaje muy parecido a C pero con algunos cambios en la sintaxis de los ciclos y condicionales, además de que las variables dejan de ser tipadas y el agregado de otros features que detallaremos más adelante en el documento, y es traducido por nuestro compilador al lenguaje C. Para analizar *natt* construimos un árbol que lo analiza sintácticamente y que genera un código de salida en lenguaje C al standard output.

Desarrollo

A través de Lex definimos nuestra gramática a través de tokens que simbolizan nuestro lenguaje. Luego Yacc utiliza dichos tokens para definir las producciones que generan el árbol de parseo, encargado de generar el código C. De esta forma, nuestros programas quedan representados en forma de nodos raíces que se subdividen en nodos cada vez más simples (como por ejemplo instrucciones u operaciones) hasta llegar a los nodos hojas que representan los tokens.

Gramática

A continuación se detalla la gramática implementada.

● No se coloca el punto y coma al final de cada línea.

● Variables no tipadas (implementado para enteros y cadenas).

● Sintaxis del if:

```
if (expresión) {
//instrucciones
}
```

● Sintaxis del while:

```
while (expresión) {
//instrucciones
}
```

● Sintaxis del for (en este caso, var debe ser de tipo entero obligatoriamente, y debe estar declarada anteriormente):

```
for (var in 1 .. 5) {
//instrucciones
}
```

● Los return son siempre entre paréntesis. Por ejemplo, return(1).

● El printf es reemplazado por print, y se le pasa directamente la variable.Por ejemplo, print(var), donde var puede ser de tipo entero o cadena.

● No hay una función para indicar el comienzo de un programa, éstos comienzan siempre en la primera línea del archivo.

● El operador ternario (“?”) funciona igual que en C. Ejemplo: condición ? do_something : do_another_thing

● Permite las operaciones aritméticas: suma (“+”), resta (“-”),

multiplicación (“*”), división (“/”) y módulo (“%”).

● Permite las relaciones: menor (“<”), mayor (“>”), igual (“==”), distinto (“!=”), menor o igual (“<=”) y mayor o igual (“>=”).

● Permite asignaciones simples (“=”) o con operaciones (“+=”, “-=”, “*=”, “/=”).

● Permite operaciones lógicas de conjunción (“&&”), disyunción (“||”) y negación (“!”).

Decisiones

Para la definición de variables, se utilizó un arreglo de estructuras donde se almacenan todas las variables que fueron declaradas. La estructura contiene el nombre de la variable y un entero que indica si la variable fue definida previamente. Se decidió que se pueden declarar hasta 30 variables, y el nombre de las mismas es de 30 caracteres como máximo. De esta forma se tiene un registro de todas las variables que están en uso para poder usarlas en otras ocasiones. Para evitar conflictos con las palabras reservadas de C, se decidió concatenar un guión bajo a todas las variables definidas en nuestros programas.

Futuras Extensiones

Se realizaron implementaciones para estructuras (stack y queue en principio) las cuales no incluimos en el scope del trabajo. Esto se debe a que para poder demostrar su funcionamiento, se requiere realizar nuevas funciones en la generación de código, que agregaba mucha complejidad a la hora de acoplarlo a nuestros features existentes.

Conclusiones

El trabajo nos permitió aplicar los conceptos vistos en la teoría de la materia, lo cual nos pareció muy positivo. En algunas discusiones que tuvimos durante el desarrollo del trabajo concluimos en que nos hubiera gustado poder dedicarle más tiempo ya que teníamos buenas ideas de implementación, y nos resultaba mucho más entretenido que hacer ejercicios en papel.

A nivel académico, creemos que la realización del trabajo es una buena herramienta para entender de manera más profunda cómo funciona un compilador y creemos que la materia debería hacer más hincapié en estos temas, ya que agrega valor a nuestros conocimientos informáticos.

Referencias

1. https://www.epaperpress.com/lexandyacc/index.html: Lex & Yacc Tutorial
2. https://www.youtube.com/watch?v=_-wUHG2rfM&t=5s: Video con un ejemplo de Lex y Yacc
3. https://www.youtube.com/watch?v=BocfpYSGYNE&t=8s: Otro video de ejemplo de una calculadora con Lex y Yacc