# 

**Autómatas, Teoría de Lenguajes y Compiladores**

***TPE - Frod-Getheode***

Primer Cuatrimestre de 2019

Integrantes

* Karpovich, Lucia 58131
* Martin, Fernando 57025
* Reyes, Santiago 58148
* Tarradellas, Manuel 58091

**Índice**

Introducción al Lenguaje 2

Desarrollo 2

Descripción de la gramática 2

Dificultades encontradas 3

Ejemplos 3

Compilación 3

Futuras extensiones 4

Diccionario 4

Referencias 6

**Introducción al Lenguaje**

El lenguaje Frod-Getheode surge ante la necesidad de simplificar la vida de gente mejor versada en el inglés antiguo que en el inglés actual (como viajeros en el tiempo o gente muy excéntrica).

Este lenguaje consiste en una traducción y simplificación del lenguaje C al inglés antiguo (para más información sobre las palabras usadas ver sección Diccionario).

Se eligió hacer la traducción a C dado que es un lenguaje muy trabajado por los miembros del grupo.

**Desarrollo**

Las herramientas utilizadas fueron Yacc y Lex acorde a lo visto en clase. La primera parte consistió en la elección de las palabras para el lenguaje y a partir de esto la creación del archivo Lex (awendan.l).

Una vez determinados todos los símbolos a utilizar, se creó la estructura que implementa el árbol de sintaxis. Para esto se obtuvo una implementación de Árbol N-ario online (ver sección Referencias) y se modificó para satisfacer las necesidades del proyecto. De esto surgen los archivos node.

Por último, se desarrolló la gramática en el archivo Yacc (forscieppan.y) la cual hace uso de la implementación de árbol mencionada anteriormente para crear el árbol de sintaxis.

**Descripción de la gramática**

Como describimos anteriormente, la gramática aplicada acepta un lenguaje similar a C que incluye los siguientes componentes:

* Tipos de datos: int, string, boolean
* Delimitadores: paréntesis, corchetes y punto y coma.
* Operadores aritméticos: + , - , \* , / , %
* Operadores relacionales: < , > , <= , >= , == , !=
* Operadores lógicos: &&, ||
* Operador de asignación (=)
* Bloque condicional If
* Bloque repetición While
* Comentarios (de una sola línea)

La gramática acepta varias funciones pero se requiere que sean escritas antes que la función main. Para más información sobre la gramática se puede consultar el archivo Yacc (forscieppan.y).

**Dificultades encontradas**

La primera y más importante dificultad encontrada fue al comienzo, dado que el grupo no tenía experiencia con Lex y Yacc. El mayor desafío del trabajo fue descifrar cómo comenzar y hacia donde proyectarlo. La parte de investigación fue lo que más tiempo requirió de todo el trabajo.

Hubo un conflicto a la hora de crear tipos string, se debatió si hacerlos estáticos o dinámicos. El problema de hacerlos estáticos era a la hora de la declaración, ya que se dificultaba realizar el chequeo del tamaño cuando se hiciera la asignación eventualmente por la forma en que está construido el árbol de sintaxis. Se decidió que las declaraciones serian del tipo char \*, es decir punteros, pero que no se permitirían mallocs sino asignaciones directas a strings estáticos.

De manera similar surgió un problema a la hora de implementar algún tipo de lectura de entrada, ya que las funciones de lectura requerían un buffer dinámico para devolver lo leído pero el usuario no maneja su liberación por cómo está formado el lenguaje. Se consideró requerir al mismo usuario liberar el espacio cada vez que realiza un scan pero finalmente se optó por invocar una función de exit específica obligatoriamente para terminar la función main. Esta función libera todos los strings alocados por la función scan y luego retorna.

**Ejemplos**

* **diegol.fg:** Ejemplo simple de uso de comentarios. Se sugiere ver como queda el archivo c compilado para ver que el compilador ignora efectivamente los comentarios.
* **geeftgadrian.fg:** Juego de simon dice donde se muestra el funcionamiento del read y write.
* **gefunden.fg:** Demostración del uso del tipo boolean y comparaciones lógicas.
* **gerimian.fg:** Función de factorial que muestra el uso de variables, asignación, ciclo while y operadores aritméticos.
* **treowwyrhta.fg:** Programa de prueba de varios aspectos. Lee e imprime de entrada estándar, luego imprime 10 numeros en un ciclo while y muestra una multiplicación simple con un llamado a función.

**Compilación**

El usuario debe estar posicionado dentro de la carpeta src, dentro del proyecto. Es necesario tener instalado Bison, Yacc y Gcc. A continuación se muestra cómo compilar uno de los archivos de prueba:

**$> ./iluvatar.sh examples/gefunden.fg**

Donde iluvatar.sh es el archivo bash que compila y gefunden.fg es el archivo de prueba.

Esto genera dos archivos: parsedCode (el ejecutable) y parsedCode.c (el archivo en lenguaje c).

**Futuras extensiones**

La extensión más clara que requiere el lenguaje es manejo de strings dinámicos. Sería útil para simplificar cómo se maneja en la actualidad la función que lee de entrada (ver sección de problemas encontrados). Esta implementación no es complicada si se utilizan exclusivamente strings dinámicos como tipo de dato ya que no requeriría chequear que las asignaciones son correctas.

Sería conveniente, además, hacer un mejor manejo de errores en un futuro para hacer que el debugging sea más sencillo para el usuario. Actualmente se esta aprovechando el uso de gcc para encontrar pequeños errores que se pueden haber pasado por alto durante el análisis de la gramática del archivo Yacc.

Otra extensión que enriquecería el lenguaje sería el agregado de más tipos de datos y declaraciones de constantes. Tampoco presenta una gran complejidad. Esta y todas las extensiones que se pensaron no se implementaron porque significaba hacer grandes cambios a la gramática con poco tiempo faltante para la entrega.

**Diccionario**

**Nombres (de archivos y utilidades)**

Frod old

Getheode language

awendan transform

forscieppan transform

gescieppan transform

Iluvatar dios supremo de todos los universos

dugan to be of use

diegol Hidden from sight, secret

gerimian To calculate, compute

#### gefunden Ascertained by calculation

geeftgadrian To repeat

treowwyrhta Carpenter, lumberjack

**Comandos**

"beginnan".\*"ablinnan" /\* comment \*/;

"aernan" main

"ut" return

"gif" if

"hwil" while

"andac" and (&&)

"othe" or (||)

"writan" printf

"araedan" read

"soth" true

"nasoth" false

"rim" int

"irenbend" string

"cunnan" boolean

"beon" assignment (=)

"ilca" equal (==)

"nailca" different (!=)

"na" not (!)

"laes" less (<)

"laesilca" less or equal (<=)

"mara" greater (>)

"marailca" greater or equal (>=)

"aegther" plus (+)

"butan" minus (-)

"eacen" multiplication (\*)

"dael" division (/)

"laf" modulus (%)

Es importante aclarar que no todas las palabras son traducciones directas sino interpretaciones, dado que no todas las palabras que el proyecto requería existían en la época de manera exacta. Un ejemplo de esto puede ser “print”, que en el proyecto reemplaza a “writan” que se traduce a “to write” en inglés actual.

Para obtener las traducciones de las palabras o curiosidad sobre nuevas palabras, los invitamos a usar los traductores y herramientas que se encuentran en la sección Referencias bajo la parte de lenguaje.

**Referencias**

**Lex - Yacc**

<https://www.youtube.com/watch?v=__-wUHG2rfM>

<https://github.com/jengelsma/yacc-tutorial>

<https://www.youtube.com/watch?v=gpmBEx_Cg8k>

<https://github.com/ajhalthor/interpreter>

**Lenguaje**

<https://oldenglishthesaurus.arts.gla.ac.uk/>

<https://www.st-andrews.ac.uk/~cr30/vocabulary/>

<http://www.majstro.com/Web/Majstro/bdict.php?gebrTaal=eng&bronTaal=eng&doelTaal=ang&teVertalen=read>

<https://www.oldenglishtranslator.co.uk/>

**Implementación de árbol**

<https://github.com/ChuOkupai/n-ary-tree>