Informe Trabajo Práctico
Teoría de Lenguajes y Autómatas

Traducción de Lenguaje C reducido a Español (dino)

Grupo: Dinopianito

Integrantes:

- *Lucía Tay (Legajo: 56244)*
- *Catalina Varela Ballesteros (Legajo: 56433)*
- *Federico Mamone (Legajo: 56255)*
- *Nicolás Clozza (Legajo: 56268)*

Primer Cuatrimestre 2018

Índice

Objetivo

En este informe vamos a detallar el desarrollo de nuestro lenguaje de programación. La idea fue aplicar los conceptos aprendidos en la materia, comenzando con algunas funcionalidades básicas y dejando abierta la posibilidad a futuras extensiones.
Nuestro objetivo fue implementar una versión básica del lenguaje C pero en Español.

Consideraciones

Nuestro lenguaje, ***dino***, es una traducción del lenguaje C (reducido) al español.

Para analizar ***dino*** construimos un árbol que lo analiza sintácticamente y que genera un código de salida en lenguaje C al stdout. Una vez generado este código, se utiliza como entrada para generar .c.

Desarrollo

Mediante el uso de Lex definimos nuestra gramática a través de *tokens* que simbolizan nuestro lenguaje. Luego Yacc utiliza dichos símbolos para definir las producciones que generan el árbol de *parseo*. De esta manera, 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*. Este árbol es el encargado de generar el código C que se corresponde con nuestra sintaxis y gramática.

Gramática

En este apartado, detallaremos la gramática implementada.

- No se coloca el punto y coma al final de cada línea.
- No es necesario definir el tipo de las variables (pueden ser enteros o cadenas).
- El condicional “if”, es interpretado como “si” y va seguido de una expresión encerrada entre paréntesis. A diferencia de C, es necesario utilizar siempre llaves. En el caso del “else”, se traduce a “sino” donde las llaves deben estar en la misma línea (ver el ejemplo). Ejemplo:

```
si (expresión) {
     //instrucciones
} sino {
     //instrucciones
}
```

- El ciclo “while”, es interpretado como “mientras”. Utilizando la misma sintaxis que en C, pero siendo necesario el uso de llaves. Ejemplo:

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

- El “return” es interpretado como “retornar” donde el valor de retorno es pasado entre paréntesis. Ejemplo:

```
retornar (0)
```

- Para reemplazar el “printf”, se implementó “imprimir” donde se le pasa por parámetro una variable de tipo int o char. Ejemplo:

```
a = 1
b = “Hola mundo”
imprimir(a)
imprimir(b)
```

- 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 ? expresion_verdadera : expresion_falsa`
- 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 20 variables, y el nombre de las mismas es de 20 caracteres como máximo. Se implementó de esta manera para poder tener un registro de las variables que hayan sido declaradas y así poder utilizar las mismas en futuras ocasiones.

Para evitar conflictos con las palabras reservadas de C, se decidió concatenar un guión bajo a todas las variables definidas en nuestros programas.

En el caso de la función “imprimir”, el parámetro de la misma debe ser una variable. Esto se debe a como están definidas las mismas.

Al código parseado, se le agrega la función “main” para que pueda ser compilado en C.

Futuras Extensiones

Para futuras versiones sería bueno incluir más funcionalidades para intentar abarcar todo el abanico de funcionalidades que tiene el lenguaje C y ofrecer su traducción. Esto puede incluir las librerías de Math, ctype, errno, entre otros. Sin embargo, priorizamos en una primera instancia agregar las siguientes funcionalidades:

- Vectores
- Funciones
- Más tipos de datos (float, double, long, etc.)
- Bloque for
- Bloque switch
- Comentarios

Referencias

1. *https://www.epaperpress.com/lexandyacc/index.html* : Lex & Yacc Tutorial
2. *https://www.gnu.org/software/bison/manual/html_node/Shift_002fReduce.html* : Shift/Reduce Conflicts
3. *https://www.ibm.com/developerworks/library/l-lexyac/index.html* : Build code with Lex and Yacc