Autómatas, Teoría de Lenguajes y Compiladores 72.39
Trabajo Práctico Especial
2C 2021

Justina Vacas Castro (60567)
Ian Mejalelaty (60584)
Josefina Assaff (57735)
Ariadna Fernandez Truglia (60199)

Fecha de entrega: 1/12/2021

Índice

1. Idea subyacente y objetivo del lenguaje

En este trabajo práctico se propuso construir un lenguaje de programación propio. Después de proponer distintas ideas nos quedamos con la más original. Esta consiste en crear un lenguaje que permite realizar traducciones de texto a braille y poder verificar si lo escrito en braille está redactado de forma correcta.

Este lenguaje va a poder ayudar a quienes estén aprendiendo a escribir en braille y quieran corroborar si lo están haciendo correctamente. El nombre que decidimos ponerle fue Brai

2. Consideraciones realizadas

Como primera instancia decidimos enfocarnos en lo que sería las operaciones aritméticas comunes (+ , - , * , /) y la traducción de texto a braille por entrada estándar y más adelante agregamos la inversa, dado una palabra u oración en braille traducirla a texto.

3. Descripción del desarrollo del TP

Para el desarrollo del trabajo utilizamos las herramientas Lex, para definir la sintaxis del lenguaje mediante tokens y YACC para definir la gramática utilizando los símbolos definidos en Lex. En esta gramática se definen las producciones que luego generan el código en C.

Por otra parte, en la gramática se agregaron restricciones al momento de crear el código, tales como: no se puede definir la misma una variable dos veces, la variable solo puede ser de alguno de los tipos definidos y el nombre de la variable debe cumplir con criterios determinados.

4. Descripción de la gramática

Para iniciar y finalizar el programa se debe escribir “inicio” y “fin” respectivamente. Dentro de estos delimitadores se pueden declarar, asignar e imprimir variables y relaciones operaciones aritméticas, de la siguiente manera:

- `num aux = 1,`
- `aux = aux + 1,`
- `imprimir aux,`
- `texto txt = “hola como estas”,`
- `imprimir txt,`

La gramática también cuenta con ciclos if y while que se inician de la siguiente manera:

- `si ( condición ) entonces [instrucción], terminado,`
- `hacer:`
  `// instrucciones`
  `mientras(condición),`

También dentro de un ciclo se puede realizar traducciones de texto a braille, ingresando por entrada estándar la oración o palabra que se quiera traducir. Como por ejemplo:

- `imprimir braille “hola”,`
- `leer y traducir, (por entrada estándar el texto)`

Además se puede verificar si lo escrito en braille es correcto o no, ingresando por ejemplo: “125.153.123.1” el cuál verifica que la palabra es “hola”. De ser incorrecto se le indica al usuario en qué letra se equivocó.

- `braille_a_texto , (por entrada estándar el texto en braille)`

Se añadió un tipo de variable braille con la cual se pueden realizar dos funcionalidades: traducir y concatenar. La traducción se puede realizar de braille a texto y viceversa. La concatenación se puede realizar entre dos tipos de variables diferentes y el resultado de la concatenación será del mismo tipo de la primera variable.

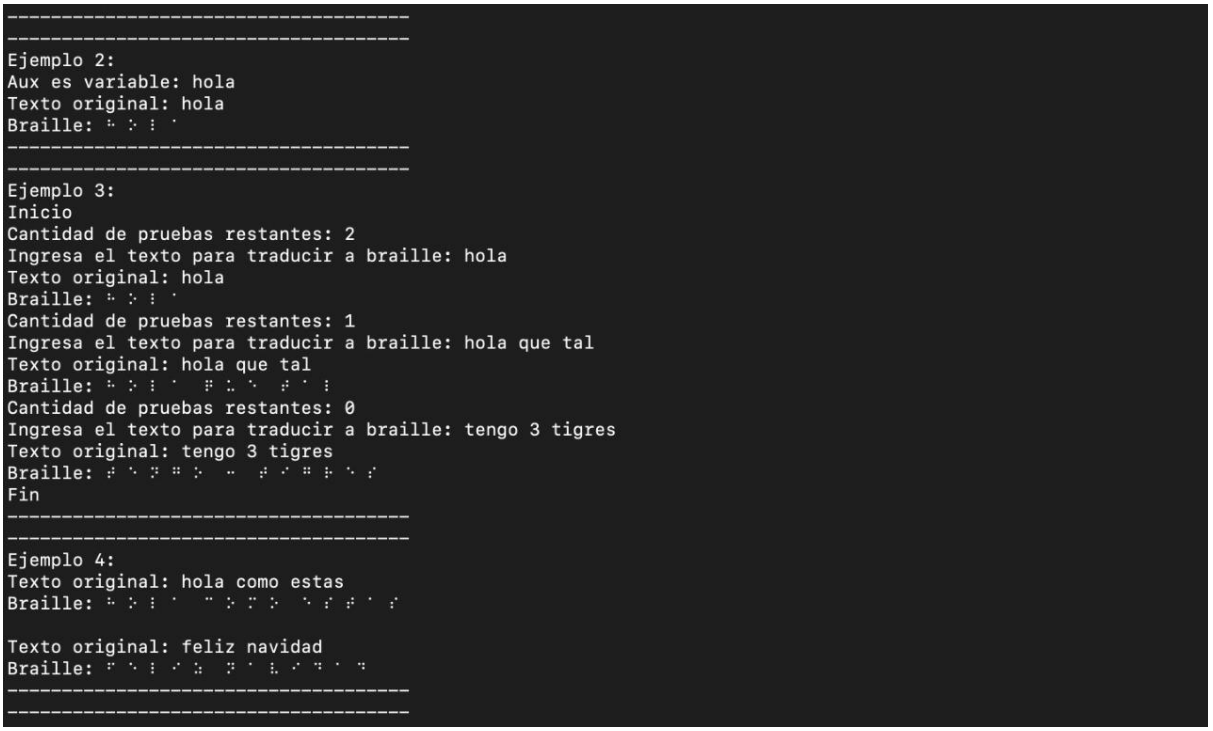
- `braille b = 234.136.15.1235.2345.15.,`
- `texto txt = traducir b,`

- `texto aux = "hola como estas",`
- `texto aux2 = "feliz navidad",`
- `texto print = concatenar aux aux2,`

Cuando se pasa de tipo texto a tipo braille en la concatenación y en la traducción se eliminan los espacios de la variable tipo texto.

Por último el delimitador utilizado al finalizar una instrucción es “,” y el comando “ imprimir línea” imprime un salto de línea.

En la siguiente imagen se muestran los ejemplos 2, 3 y 4. En los cuales se asignan variables y se imprime su traducción en braille.



5. Dificultades encontradas en el desarrollo del TP

A lo largo del trabajo nos encontramos con diferentes dificultades al querer definir la gramática. En un principio la idea fue definir un tipo string texto y un tipo string braille, pero al no poder realizarlo nos quedamos con un solo tipo de string.

Luego se dificulto la definición de la estructura. Se realizó una estructura tipo árbol pero al ver que era más complejo de lo que pensamos lo terminamos cambiando a una lista de nodos.

Por estas dificultades tuvimos que borrar y volver a empezar tres veces hasta llegar a la versión final que cumple con los requisitos y nuestra idea de base.

6. Futuras extensiones

En un futuro la idea sería crear un tipo de dato braille con el cual se podrían realizar más funcionalidades con el lenguaje, como realizar contracciones y concatenaciones de strings. Para realizar esto deberíamos crear un nuevo token en el parser y las producciones correspondientes en la gramática además de un verificador de tipo de variable, para poder separar el string texto del string braille.

7. Referencias

https://github.com/AaditT/braille/blob/master/braille.py
Se utilizó parte de este código para lo que es la impresión de los símbolos de braille.

Se utilizaron los apuntes y videos provistos por la cátedra para comprender el uso de yacc y lex.

8. Conclusión

Podemos concluir que, si bien tuvimos que comenzar de cero el desarrollo del Trabajo Práctico varias veces, estamos conformes con la versión final a la que llegamos. Simplificamos varios aspectos de la idea original para poder llevarla a cabo, pero en un futuro se podría mejorar y agregar nuevas funcionalidades para así darle más valor al lenguaje.

De este trabajo nos llevamos las herramientas Lex y YACC, que al principio fueron complicadas de comprender, pero actualmente nos sentimos cómodos con ellas y creemos que en un futuro podrán ser de gran ayuda.