Informe: Trabajo Práctico Especial

72.39 - Autómatas, Teoría de Lenguajes y Compiladores

Grupo 10

Cupitó, Felipe	60058
Finucci Roca, Hernán	60178
Kim, Azul Candelaria	60264
Vásquez Currie, Malena	60072

Tabla de Contenidos

Introducción

El presente informe busca familiarizar al lector con Music++, lenguaje creado como parte del trabajo práctico de la materia. La idea de este proyecto fue desarrollar un lenguaje de programación que le permita al usuario operar con notas musicales, convirtiendo notas a acordes (y viceversa) o generando partituras en formato LaTeX (MusixteX).
Está diseñado tanto para aficionados de la música como para aquellos que no tienen dotes musicales pero aspiran a aprender.

Consideraciones y Limitaciones

La única consideración o limitación que se debe destacar es que los acordes contemplados en el trabajo tienen notación de tono estándar estadounidense (ASPN). A su vez las notas deben estar en notación musical latino[1]. Es importante que el usuario considere estos aspectos a la hora de definir variables y correr programas.

Gramática/Sintaxis del Lenguaje

Para comenzar y finalizar cada programa es obligatorio colocar las siguientes instrucciones:

```
start
// Codigo
end
```

Cabe aclarar que cada fin de línea del cuerpo debe ser finalizada con el delimitador ;

Declaración y Asignación de Variables

Music++ acepta los siguientes tipos de datos:

- string
- integer
- chord
- note

Para declarar una variable:

- string [nombre]
- integer [nombre]
- chord [nombre]
- note [nombre]

Para asignarle un valor a una variable:

- string [nombre] is [valor]
- integer [nombre] is [valor]
- chord [nombre] is [valor]
- note [nombre] is [valor]

[1] https://artsandculture.google.com/entity/m0nl12?hl=es

Instrucciones Disponibles

A continuación se detallan algunas de las instrucciones permitidas:

Instrucción	Funcion
print [string | integer | note]	imprime en pantalla la variable
print_to_chords [note] [note] [note] …	imprime el acorde correspondiente a las notas dadas

compute
// código
while (condición)
	equivalente a un ciclo do-while: ejecuta el código mientras la condición dada sea verdadera

if (condición)
then
// código
else
// código
end_if
	equivalente a un if-else tradicional

[variable] sum [variable] [variable] minus [variable] [variable] times [variable] [variable] by [variable]	permiten realizar operaciones aritméticas como suma, resta, multiplicación o división. solo pueden ser usados con integers.
[variable] > [variable] [variable] < [variable] [variable] = [variable] [variable] != [variable] [variable] and [variable] [variable] or [variable] [variable] not [variable]	permiten realizar operaciones lógicas.
to_notes [chord]	convierte un acorde válido a notas
to_chord [note] [note] [note]	convierte notas a su acorde
is_note [note]	verifica si la nota dada es válida
is_chord [chord]	verifica si el acorde dada es válida
concat_notes [note] [note] [note]	concatena notas
create_music [note | chord | variable]	crea una partitura

Todas las instrucciones (declaraciones y asignaciones incluidas) a excepción de start y end deben finalizar con el delimitar “;”.

Desarrollo del Proyecto y Fases del Compilador

La arquitectura básica de un compilador está representada por la siguiente imagen:

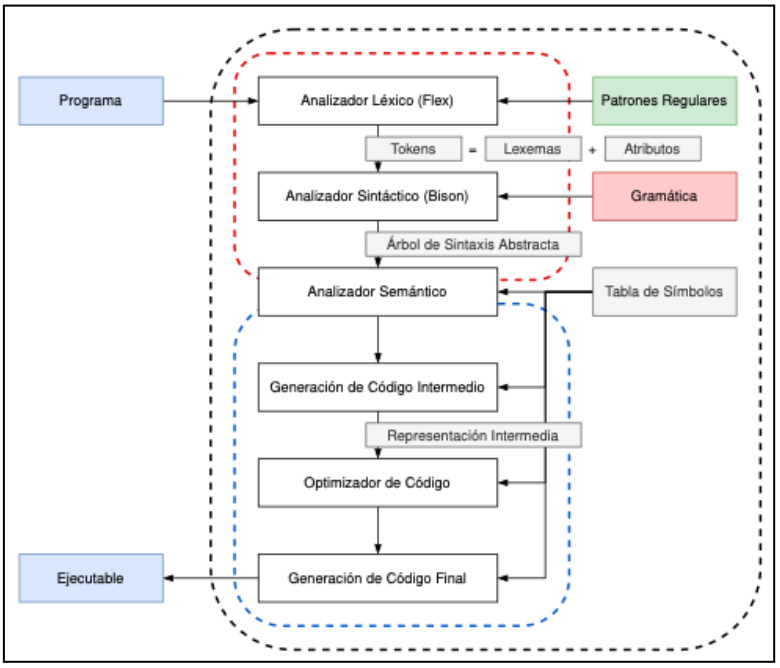

Figura 1. La arquitectura de un compilador y sus componentes principales. (Imagen tomada del enunciado)

Cuando recibe un programa de entrada, se aplican transformaciones denominadas *fases*. El grado de dependencia que posee la representación que producen con respecto a la plataforma final sobre la que el programa será ejecutado determina si la fase pertenece al *frontend* o *backend*. [2]

Esta división entre front y back marcó el desarrollo del proyecto. En un primer lugar se construyó el frontend, en el cual se crearon patrones/expresiones regulares, la gramática y un recognizer que determinaba la validez de un programa. Esta parte también incluyó la definición de tipos de datos (string, integer, note, chord) y funciones necesarias para poder reconocer que programas respetaban la sintaxis de lenguajes.

Luego se avanzó al backend en donde se construyó sobre lo diseñado previamente y se completaron las fases restantes. Se utilizó C para que el compilador pueda funcionar como analizador semántico. Una vez logrado eso, y para la fase de generación de código, se implementó un árbol de sintaxis abstracta (AST).

Para finalizar el proyecto se realizaron varios programas de prueba. El objetivo detrás de esto fue verificar el correcto funcionamiento del compilador.

[2] Enunciado v2.0.0 - campus de la materia

Dificultades

La mayor dificultad que se presentó a la hora de desarrollar el proyecto fue la implementación de la idea original (reproducción de sonidos a partir de notas y acordes). Para llevarla a cabo, se debía instalar una librería específica (CFugue) que solamente funciona en Linux nativo. Además, al reproducir sonidos a partir de un archivo de prueba, no sonaba bien. Por lo tanto, se reemplazó esto por creación de partituras en LaTeX.

La creación de la gramática fue un aspecto complicado del proyecto ya que al principio, no teníamos los conocimientos necesarios para comprender los errores que el compilador devolvía. Estos errores eran del estilo “shift/shift” o “shift/reduce”. No obstante, pudimos comprenderlos después de tener la clase teórica relacionada con este tema.

Futuras extensiones

Una extensión posible para este proyecto sería la implementación de los sonidos que no se pudieron hacer. Se podría utilizar la partitura como referencia para crear la música. Para esto, se debería preguntar si es que se puede instalar en Pampero la librería mencionada previamente. Si es que esta idea no es viable, existen otras librerías que producen música las cuales son más potentes que la investigada; sin embargo, se debería migrar el proyecto a otro lenguaje (como Java) para poder probarlo.

Se podría agregar más estilos de ciclos, como un *for*, o también estructuras (*struct*). Sin embargo, la segunda es más compleja porque se debería por lo tanto implementar punteros en nuestro lenguaje.

Referencias

Para la escritura del archivo .tex - MusiXTEX - TeXDochttps://texdoc.org › serve › musixtex
Enunciado v2.0.0 - Campus ITBA (Subido por la cátedra)