Implementación de un Intérprete de Interpretes con ANTLR4 y Python

Jacobo Arroyave Pérez

David Esteban Toro Herrera

Thomas Alejandro Toro Herrera

Universidad Autónoma de Manizales

Facultad de Ingeniería de Sistemas

Paradigmas de Lenguajes de Programación

2025

**Abstract**

Este proyecto tiene como objetivo el desarrollo de un IcS (Intérprete con sentido) utilizando ANTLR4 y Python, que a su vez puede ser interpretado en: código morse, tácticas de fútbol, recetas de cocina, representaciones de información geográfica, notas musicales o instrucciones en general, en el presente caso para notas musicales. A través de un archivo .txt, el usuario podrá ingresar notas musicales con una estructura adecuada a la sintaxis definida en la gramática, el intérprete se encargará de analizarlas y luego generar un archivo .mid que contiene el audio de la nota. Además, dicho IcS será no solo orientado para tareas del mundo real como la generación de archivos de audio, sino también orientado a ser un lenguaje de programación común y corriente; esto indica que tendrá funcionalidades aritméticas de cálculos como sumas, restas, multiplicaciones, divisiones, entre otras; adicional a ello, contará con estructuras básicas de un lenguaje como condicionales y sentencias de control, obligando a cambiar la estructura de la gramática.

La principal herramienta utilizada para interpretar las fórmulas es ANTLR4, que genera los analizadores necesarios para procesar la entrada. El sistema es capaz de generar resultados precisos y presentarlos de forma clara, permitiendo su exportación en formato MID. Este proyecto no solo facilita el análisis de datos, sino que también introduce al usuario en el uso de herramientas de interpretación de lenguajes de programación, combinando teoría y práctica de manera efectiva.

**Tabla de Contenidos**

[1. Planteamiento del Problema 4](#_Toc1280417183)

[2. Análisis del Problema 9](#_Toc766934044)

[2.1 Casos de uso 10](#_Toc667817668)

[2.1.1 Casos de uso 1: Definir Archivo txt 10](#_Toc2078593852)

[2.1.2 Casos de uso 2: Cargar archivo txt 11](#_Toc1979734564)

[2.1.3 Caso de uso 3: Ejecutar código definido en el txt 11](#_Toc2135550055)

[2.1.4 Caso de uso 4: Mostrar mensajes por consola 12](#_Toc1925897961)

[2.1.5 Caso de uso 5: Solicitar entrada del usuario 12](#_Toc1778155341)

[2.1.6 Caso de uso 6: Ingresar la entrada solicitada 13](#_Toc778661218)

[2.1.7 Caso de uso 7: Leer la entrada del usuario 14](#_Toc1373266615)

[2.1.8 Caso de uso 8: Asignar variables 14](#_Toc1999869597)

[2.1.9 Caso de uso 9: Evaluar expresiones 15](#_Toc1789955809)

[2.1.10 Caso de uso 10: Ejecutar estructuras de control 15](#_Toc1537564623)

[2.1.11 Caso de uso 11: Generar archivo de música MIDIEl sistema permite 16](#_Toc1426518013)

[3. Diseño de la solución propuesta 17](#_Toc505307710)

[3.1 Diagrama de clases 17](#_Toc422854795)

[4. Documentación de la implementación 18](#_Toc581031490)

[4.1 Gramática del lenguaje 18](#_Toc1630294209)

[4.2 Visitador del lenguaje 23](#_Toc838059155)

[4.3 Interpretador de interprete 24](#_Toc1316168993)

[4.4 Manual de usuario 29](#_Toc1362782615)

[4.5 Manual técnico 34](#_Toc680727940)

[5. Conclusiones 34](#_Toc1927345521)

[6. Lista de referencias 35](#_Toc606075403)

[7. Vita 35](#_Toc1032704767)

**Lista de figuras**

[Figura 1. Ejemplo de input de notas musicales para nuestro lenguaje 2](#_Toc193211360)

[Figura 2. Ejemplo de programa 3](#_Toc193211361)

[Figura 3. Ejemplo de invocación del intérprete 5](#_Toc193211362)

[Figura 4. Diagrama de casos de uso 7](#_Toc193211363)

[Figura 5. Diagrama de clases 16](#_Toc193211364)

[Figura 6. Gramática en ANTLR4 17](#_Toc193211365)

[Figura 7. Gramática en antlr4 18](#_Toc193211366)

[Figura 8. Gramática en ANTLR4 18](#_Toc193211367)

[Figura 9. Gramática en anltr4 19](#_Toc193211368)

[Figura 10. Gramática en antlr4 20](#_Toc193211369)

[Figura 11. Gramática en antlr4 21](#_Toc193211370)

[Figura 12. Comando de generación de archivos en antlr4 22](#_Toc193211371)

[Figura 13. Función visitProgram 25](#_Toc193211372)

[Figura 14. Función visitAssignmentStatement 25](#_Toc193211373)

[Figura 15. Función visitArithmeticExpr 26](#_Toc193211374)

[Figura 16. Función visitIfStmt 26](#_Toc193211375)

[Figura 17. Función visitWhileStmt 27](#_Toc193211376)

[Figura 18. Funciones visitPrintStmt y visitReadStmt 28](#_Toc193211377)

[Figura 19. Función visitMusicStmt 29](#_Toc193211378)

[Figura 20. Consola de comandos de carpeta IcSParadigmas 30](#_Toc193211379)

[Figura 21. Comando de ejecución del programa 30](#_Toc193211380)

[Figura 22. Archivo código.ics 31](#_Toc193211381)

[Figura 23. Archivo código.ics 31](#_Toc193211382)

[Figura 24. Explorador de archivos de IcSParadigmas 33](#_Toc193211383)

[Figura 25. Output del programa ics\_interpreter.py 34](#_Toc193211384)

[Figura 26. Output del programa ics\_interpreter.py 34](#_Toc193211385)

[Figura 27. Ejecución del archivo melodía.mid 35](#_Toc193211386)

[Figura 28. Instalación de midiutil 35](#_Toc193211387)

**Lista de tablas**

[Tabla 1 Ejemplo de especificaciones del lenguaje 3](#_Toc193211280)

[Tabla 2 Alcance de validación de errores 4](#_Toc193211281)

[Tabla 3 Diagrama de secuencia en el análisis de una expresión 15](#_Toc193211282)

# Planteamiento del Problema

El intérprete debe estar orientado a la composición de algoritmos y tiene como propósito generar instrucciones en el mundo real, las cuales pueden ser almacenadas en diferentes formatos.

En la actualidad, el análisis de grandes cantidades de datos se ha vuelto crucial en diversos campos como la estadística, la ingeniería y la investigación. Sin embargo, muchas veces se requiere realizar cálculos complejos sobre esos datos de manera rápida y eficiente, lo cual puede ser un desafío si no se dispone de herramientas accesibles y personalizables. En este contexto, surge la necesidad de un sistema que permita a los usuarios ingresar notas musicales a través de un archivo de texto con la sintaxis adecuada, aplicarlas a conjuntos de datos y obtener resultados de manera automatizada.

Este proyecto busca abordar esta necesidad mediante el desarrollo de un sistema basado en ANTLR4 **y** Python que a su vez puede ser interpretado en: código morse, tácticas de fútbol, recetas de cocina, representaciones de información geográfica, notas musicales o instrucciones en general, en el presente caso para notas musicales.

El desafío principal de este proyecto radica en diseñar una gramática que pueda interpretar una pequeña variedad de notas musicales, instrucciones mediante sentencias de control y funcionalidades de un lenguaje común y corriente.

El intérprete debe reconocer notas musicales dadas en particular desde la función Main. ejemplo:

|  |
| --- |
| \*\*\* Input de notas musicales\*\*\*  Main {  <w> "Reproduciendo escala mayor de C";  [:] { C D E F G A B C };  } |

Figura 1. Ejemplo de input de notas musicales para nuestro lenguaje

Adicionalmente, el intérprete debe permitir escribir programas simples haciendo uso de números enteros. Por ejemplo, el siguiente programa lee dos números, los suma, muestra un mensaje según el resultado y reproduce una secuencia de notas según la condición. Luego, cuenta regresivamente desde el resultado hasta 0.

|  |
| --- |
| \*\*\* Un programa IcS \*\*\*  Main {  "Bienvenido al intérprete IcS";  <w> "Ingrese dos números enteros:";  <r> a;  <r> b;  <w> "Este es el primer numero:";  <w> a;   c := a \* b;  <w> "Este es el valor de c:";  <w> c;  contador := 1;   If c > 10 ::  <w> "La suma es mayor que 10";  [:] { D D E F G A B C# D# F# G# A# A };  else ::  <w> "La suma es 10 o menor";  [:] { D D D D };   While contador <= 5 ::  <w> contador;  contador := contador + 1;  } |

Figura 2. Ejemplo de programa

A continuación, se muestran las especificaciones del lenguaje, las cuales son soportadas por el programa elaborado.

Tabla Ejemplo de especificaciones del lenguaje

| **Instrucción** | **Símbolo** | **Ejemplo** | **Explicación** |
| --- | --- | --- | --- |
| Asignación de variables | := | c := a + b | Define una variable o almacena el resultado de una operación. |
| Operaciones aritméticas | + - \* / | (a + b) \* 2 | Realiza operaciones matemáticas entre variables o números. |
| Referencia a notas | ID | C | Hace referencia a una nota musical (ej. C, D#, Bb). |
| Llamado a la función | Nombre\_funcion(parametros) | Suma(a, b) | Ejecuta una función con parámetros. |
| Funciones musicales | [:]{notas} | [:]{C D E} | Ejecuta una secuencia de notas musicales. |
| Condicionales | if ... ; else ; | if c > 10 ; [:]{C D E} ; else ; [:]{A G F} ; | Evalúa una condición y ejecuta una acción según el resultado. |
| Bucles | while ... ; | while c > 0 ; c := c - 1 ; | Ejecuta un bloque de código mientras se cumpla una condición. |

En la siguiente tabla se podrán apreciar distintos ejemplos de errores posibles en el sistema.

Tabla 2 Alcance de validación de errores

|  |  |
| --- | --- |
| Error 1 | Error al procesar la entrada del usuario: <r> variableNoValida |
| Error 2 | La nota ingresada no es válida: NotaInvalida |
| Error 3 | Error al evaluar una expresión matemática: [error específico] |
| Error 4 | Función no soportada: funcionNoDefinida |
| Error 5 | Falta una condición en if o while |

El intérprete deberá poder ser invocado de las siguientes formas:



Figura . Ejemplo de invocación del intérprete

# Análisis del Problema

El presente sistema permite la interpretación de notas musicales importadas desde un archivo de texto y la generación del audio de la nota respectiva, así como la evaluación de operaciones aritméticas como sumas o restas, que a partir de ciertas estructuras de control, permite la ejecución de notas musicales dependiendo del resultado de la operación. En el contexto de análisis de datos, los usuarios requieren una herramienta que procese expresiones matemáticas y notas musicales definidas en una sintaxis apta para la gramática. Para ello, el sistema facilita la carga de archivos .txt y el procesamiento de expresiones contenidas en un archivo de texto.

El procesamiento de las instrucciones se realiza mediante ANTLR4, que permite interpretar y evaluar expresiones matemáticas y notas musicales. El sistema extrae los valores de la entrada y aplica las operaciones tanto aritméticas como de conversión a notas musicales especificadas en el archivo de texto. Como resultado, se obtiene una salida de audio de una nota específica en formato .mid, el cual es guardado en la carpeta de ejecución del proyecto.

## Casos de uso

A continuación, se muestra el caso de uso general del problema abordado.

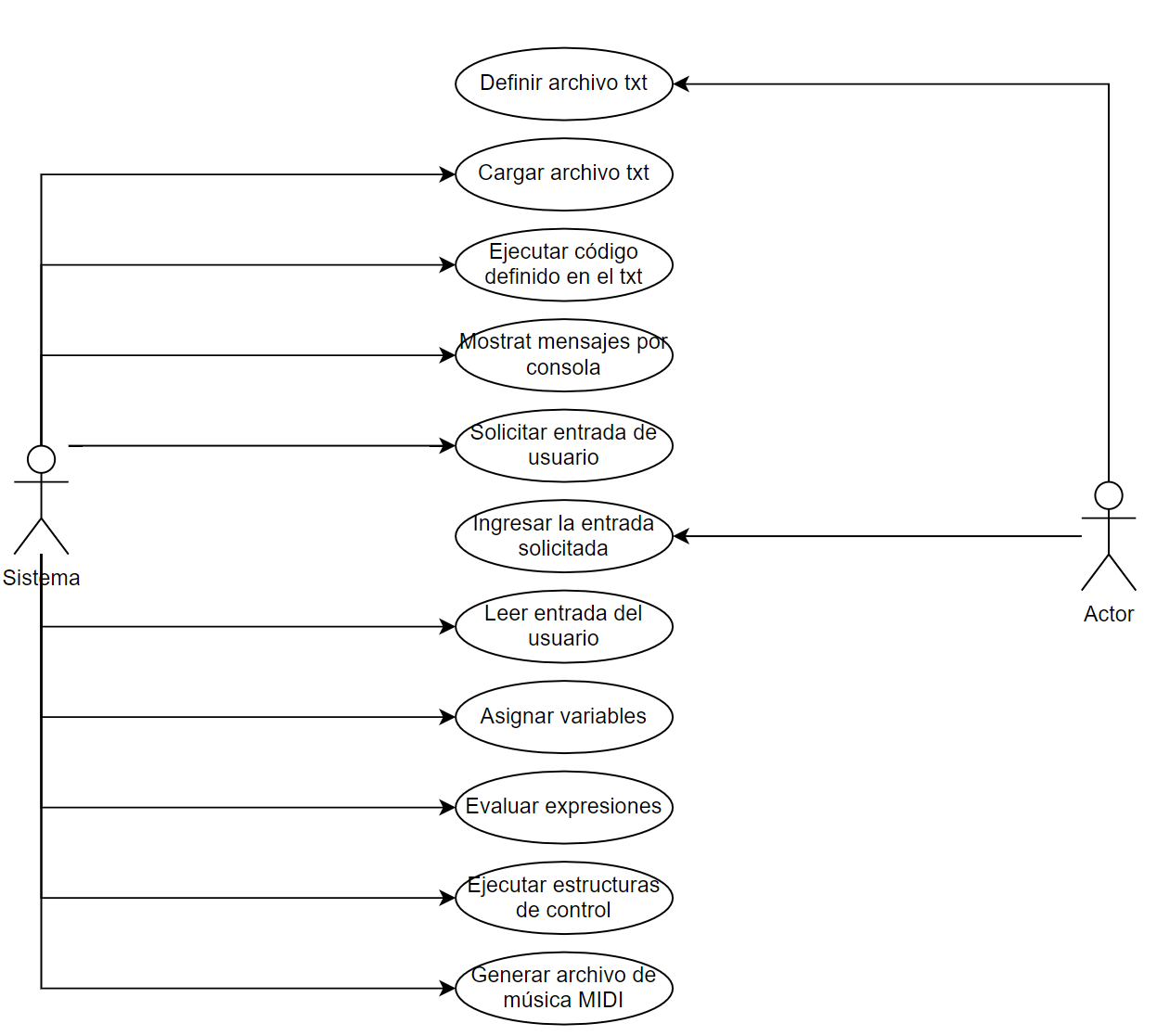


Figura . Diagrama de casos de uso

### Casos de uso 1: Definir Archivo txt

**Descripción**  
 El usuario crea un archivo de texto que contiene el código del programa a ejecutar. En este archivo, el usuario puede definir variables, asignar valores, crear notas musicales, utilizar estructuras de control y ciclos, entre otras operaciones.

**Actor principal**: Usuario

**Flujo principal**

1. El usuario redacta el código en un archivo de texto con la sintaxis del lenguaje definido.
2. Guarda el archivo con la extensión adecuada (.ics).

### Casos de uso 2: Cargar archivo txt

**Descripción**  
 El sistema carga el archivo de texto especificado por el usuario y extrae su contenido para su posterior análisis léxico y sintáctico mediante ANTLR (Parr & Quong, 1995).

**Actor principal**: Usuario

**Flujo principal**

1. El usuario selecciona el archivo a cargar.
2. El sistema lee el archivo y extrae su contenido.
3. El sistema realiza el análisis léxico y sintáctico utilizando ANTLR.
4. Si hay errores en la sintaxis, se notifican al usuario.

### 2.1.3 Caso de uso 3: Ejecutar código definido en el txt

**Descripción:**  
 Una vez validado el código del archivo, el sistema ejecuta las instrucciones paso a paso, incluyendo la generación de sonidos musicales.

**Actor Principal:** Usuario

**Flujo Principal:**

1. El sistema interpreta y ejecuta las instrucciones del archivo.
2. Si el código contiene notas musicales, las procesa y genera sonido.
3. Si hay errores en la ejecución, se notifican al usuario.

### 2.1.4 Caso de uso 4: Mostrar mensajes por consola

**Descripción:**  
 El usuario puede incluir en su código instrucciones para que el sistema muestre mensajes en consola, como valores de variables o textos personalizados.

**Actor Principal:** Usuario

**Flujo Principal:**

1. El usuario escribe una instrucción de salida en el archivo.
2. Durante la ejecución, el sistema muestra el mensaje correspondiente en la consola.

### 2.1.5 Caso de uso 5: Solicitar entrada del usuario

**Descripción:**  
 El usuario puede incluir en su código instrucciones para que el sistema muestre mensajes en consola, como valores de variables o textos personalizados.

**Actor Principal:** Usuario

**Flujo Principal:**

1. El usuario escribe una instrucción de salida en el archivo.
2. Durante la ejecución, el sistema muestra el mensaje correspondiente en la consola.

### 2.1.6 Caso de uso 6: Ingresar la entrada solicitada

**Descripción:**  
 El usuario ingresa el valor solicitado por el sistema cuando una instrucción de lectura es ejecutada.

**Actor Principal:** Usuario

**Flujo Principal:**

1. El usuario introduce el valor solicitado.
2. El sistema captura la entrada y continúa la ejecución del programa.

### 2.1.7 Caso de uso 7: Leer la entrada del usuario

**Descripción:**  
 El sistema interpreta y almacena el valor ingresado por el usuario para su posterior uso en el programa.

**Actor Principal:** Sistema

**Flujo Principal:**

1. El sistema lee y procesa el valor ingresado.
2. Si el valor es inválido, muestra un mensaje de error y solicita nuevamente la entrada.

### 2.1.8 Caso de uso 8: Asignar variables

**Descripción:**  
 El sistema asigna valores a variables según lo definido en el código.

**Actor Principal:** Sistema

**Flujo Principal:**

1. El usuario define una variable y le asigna un valor en el código.
2. Durante la ejecución, el sistema almacena y gestiona los valores asignados.

### 2.1.9 Caso de uso 9: Evaluar expresiones

**Descripción:**  
 El sistema evalúa expresiones matemáticas y lógicas presentes en el código.

**Actor Principal:** Sistema

**Flujo Principal:**

1. El usuario escribe expresiones en el código (suma, resta, comparaciones, etc.).
2. Durante la ejecución, el sistema resuelve las expresiones y almacena o usa los resultados.

### 2.1.10 Caso de uso 10: Ejecutar estructuras de control

**Descripción:**  
 El sistema ejecuta estructuras de control como condicionales (if-else) y bucles (while).

**Actor Principal:** Sistema

**Flujo Principal:**

1. El usuario escribe estructuras de control en el código.
2. Durante la ejecución, el sistema evalúa las condiciones y ejecuta las instrucciones correspondientes.

### 2.1.11 Caso de uso 11: Generar archivo de música MIDIEl sistema permite

**Descripción:**  
 El sistema genera un archivo MIDI a partir de las notas musicales definidas en el código.

**Actor Principal:** Sistema

**Flujo Principal:**

1. El usuario escribe instrucciones para generar música en el código.
2. Durante la ejecución, el sistema convierte las notas en eventos MIDI.
3. Se guarda un archivo .mid que puede ser reproducido con software de audio.

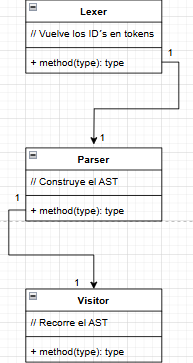
# Diseño de la solución propuesta

Crear un diagrama de clases que represente el árbol de sintaxis abstracta y un diagrama de secuencia de la forma cómo los datos se interpretaran (flujo).

Tabla 3 Diagrama de secuencia en el análisis de una expresión

|  |  |
| --- | --- |
| **Ejemplo de expresión** | **Ejemplo de árbol de sintaxis** |
| Main{  <w> "Iniciando secuencia";  [:]{ G A# B };  <w> "Finalizando";  } |  |

## Diagrama de clases



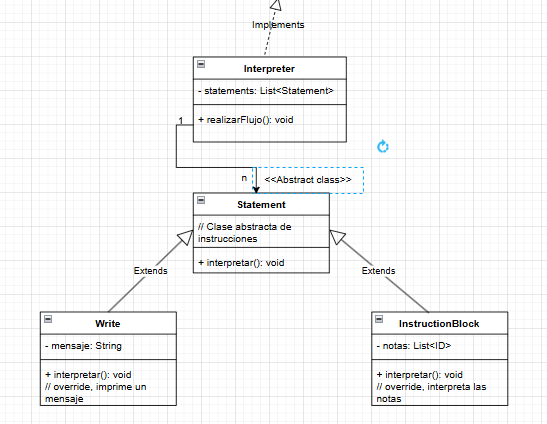
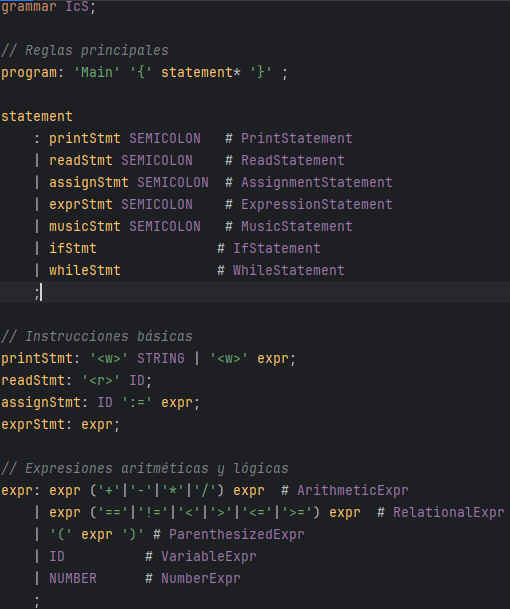


Figura . Diagrama de clases

# Documentación de la implementación

## Gramática del lenguaje



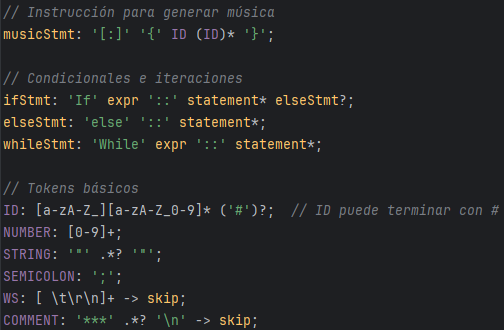


Figura . Gramática en ANTLR4

La gramática mostrada en la figura 4 es la usada en la implementación de la solución a nuestro problema.

**Regla program:**

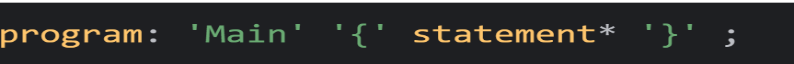


Figura 7. Gramática en antlr4

Esta es la regla principal de la gramática. Define que todo programa en IcS debe iniciar con Main y abrir llaves {}.  
 Dentro de las llaves puede haber cero o más instrucciones (statement\*).

**Regla statement:**

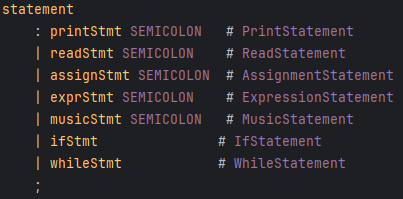


Figura 8. Gramática en ANTLR4

Cada instrucción (statement) puede ser de distintos tipos:

* printStmt → Imprimir un mensaje (<w> "texto")
* readStmt → Leer una variable (<r> nombre\_variable)
* assignStmt → Asignar un valor a una variable (x := 5)
* exprStmt → Evaluar una expresión matemática (x + 2)
* musicStmt → Tocar notas musicales ([:]{ C D E })
* ifStmt → Condicionales (If x > 0 :: ...)
* whileStmt → Ciclos (While x > 0 :: ...)

Las primeras cinco instrucciones terminan con ;, pero if y while no porque pueden contener múltiples sentencias.

**Reglas de instrucciones básicas:**

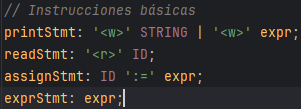


Figura 9. Gramática en anltr4

printStmt permite imprimir texto o una expresión:

* <w> "Hola"
* <w> x + 3

readStmt permite leer una variable:

* <r> x (*Lee un valor y lo guarda en x*)

assignStmt permite asignar un valor a una variable:

* x := 5
* total := a + b

exprStmt evalúa una expresión pero no la asigna ni imprime.

* x + 2; (se evalúa pero no se guarda en ninguna variable)

**Regla Expr:**

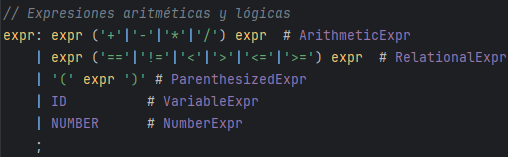


Figura 10. Gramática en antlr4

Se pueden usar operadores matemáticos y comparaciones:

* 5 + 3
* x \* (y - 2)
* a > b

**Regla MusicStmt:**

Define una secuencia de notas musicales:

* [:]{ C D E }
* [:]{ A G F# }

**Condicionales if-else y ciclo while:**

Permite estructurar el control de las instrucciones, ya que están sujetas a una condición y en el caso del ciclo hacerla cuantas veces sea verdadera dicha condición.

**Tokens:**

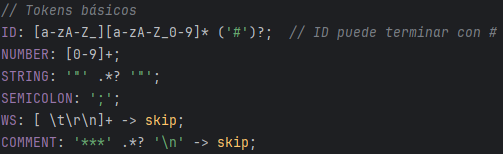


Figura 11. Gramática en antlr4

ID Representa los identificadores, como nombres de variables o funciones.

NUMBER define números enteros. Un número debe comenzar con al menos un dígito (0-9), seguido de más dígitos.

STRING cadenas de texto entre comillas dobles.

SEMICOLON representa el “;” como finalizador de instrucciones.

WS (WhiteSpaces) ignora los espacios en blanco entre las expresiones.

COMMENT ignora los comentarios que comiencen con “\*\*\*”.

**Implementación y uso de la gramática con ANTLR4:**

Para utilizar la gramática en ANTLR4 y generar los analizadores necesarios en Python, se deben seguir los siguientes pasos:

* **Generación del lexer y parser**  
  Desde la línea de comandos, ejecutar:



Figura 12. Comando de generación de archivos en antlr4

Esto generará los archivos IcSLexer.py, IcSParser.py y IcSVisitor.py, que se utilizarán para analizar expresiones.

## Visitador del lenguaje

El visitador en IcS se encarga de recorrer el árbol de sintaxis generado por ANTLR4 e interpretar las instrucciones del programa.

**Estructura del visitador:**

El visitador extiende la clase generada por ANTLR4 (IcSVisitor.py) e implementa métodos específicos para cada nodo del árbol sintáctico.

* **Expresiones aritméticas**: Evalúa operaciones como +, -, \*, /.
* **Condicionales (if-else)**: Evalúa la condición y ejecuta el bloque correspondiente.
* **Bucles (while)**: Ejecuta instrucciones repetitivas mientras la condición se cumpla.
* **Salida (<w> "texto")**: Imprime mensajes en consola.
* **Entrada (<r> variable)**: Captura datos del usuario.
* **Manejo de notas musicales ([:]{ C D E })**: Genera eventos MIDI.

**Ejemplo de implementación:**

Main {

<w> "Ingrese un número";

<r> num;

If num > 5 ::

<w> "Número mayor a 5";

[:] { C D E };

else ::

<w> "Número menor o igual a 5";

[:] { G F E };

}

**Cómo lo interpreta el visitador:**

1. **Lee** la entrada del usuario.
2. **Evalúa** la condición num > 5.
3. **Ejecuta** el bloque if o else.
4. **Genera** los sonidos MIDI correspondientes.

## Interpretador de interprete

A continuación, se podrán apreciar algunos de los métodos más importantes del archivo ics\_interpreter.py, el cual extiende del IcSVisitor.py e implementa el flujo principal del programa:



Figura 13. Función visitProgram

La función visitProgram recorre el árbol de sintaxis y ejecuta cada declaración del código en orden. Llama a visit para evaluar expresiones, asignar valores y procesar estructuras de control.

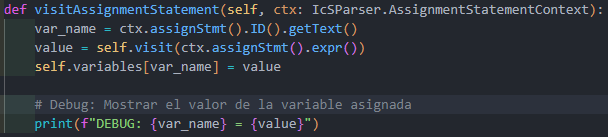


Figura 14. Función visitAssignmentStatement

La función visitAssignmentStatement procesa una asignación de variable en el código. Primero, obtiene el nombre de la variable a través del contexto del analizador. Luego, evalúa la expresión asignada utilizando la función visit. Finalmente, almacena el resultado en un diccionario que mantiene los valores de las variables para su uso posterior en la ejecución del programa.

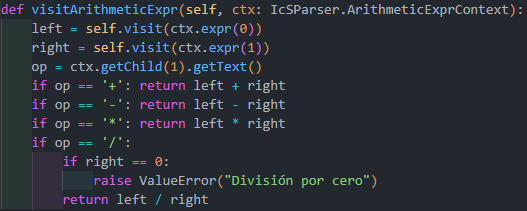


Figura 15. Función visitArithmeticExpr

La función visitArithmeticExpr evalúa expresiones aritméticas en el código. Primero, obtiene los valores de los operandos llamando a visit en los subárboles correspondientes. Luego, identifica el operador aritmético utilizado, como suma, resta, multiplicación o división. Finalmente, realiza la operación correspondiente y devuelve el resultado, manejando casos especiales como la división por cero.

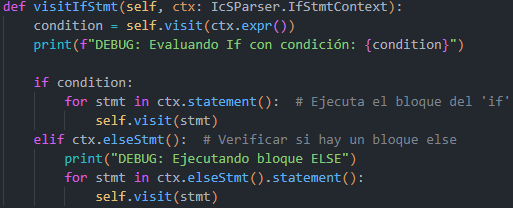


Figura 16. Función visitIfStmt

La función visitIfStmt evalúa una estructura condicional en el código. Primero, evalúa la expresión lógica dentro de la condición. Si el resultado es verdadero, ejecuta las instrucciones dentro del bloque If. Si la condición es falsa y existe un bloque Else, ejecuta las instrucciones de ese bloque. También imprime mensajes de depuración para indicar qué camino se está ejecutando.

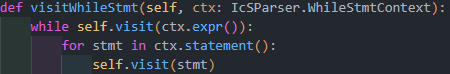


Figura 17. Función visitWhileStmt

La función visitWhileStmt evalúa y ejecuta una estructura de bucle While en el código. Primero, verifica la condición del bucle y, si es verdadera, ejecuta las instrucciones dentro del bloque repetidamente. Tras cada iteración, vuelve a evaluar la condición para determinar si debe continuar o detenerse. Esto permite la ejecución de ciclos basados en condiciones dinámicas.

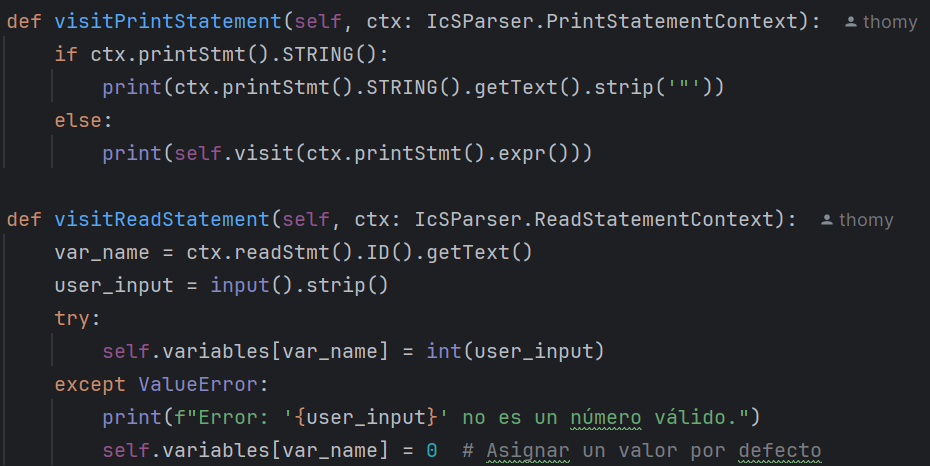


Figura 18. Funciones visitPrintStmt y visitReadStmt

La función visitAssignmentStatement se encarga de procesar las asignaciones de variables en el código. Obtiene el nombre de la variable y evalúa la expresión asociada, almacenando el resultado en un diccionario de variables. Esto permite que las variables sean utilizadas posteriormente en otras partes del programa.

La función visitArithmeticExpr evalúa expresiones matemáticas que incluyen suma, resta, multiplicación y división. Obtiene los valores de los operandos, identifica el operador y realiza la operación correspondiente. Si se detecta una división por cero, lanza un error para evitar fallos en la ejecución.

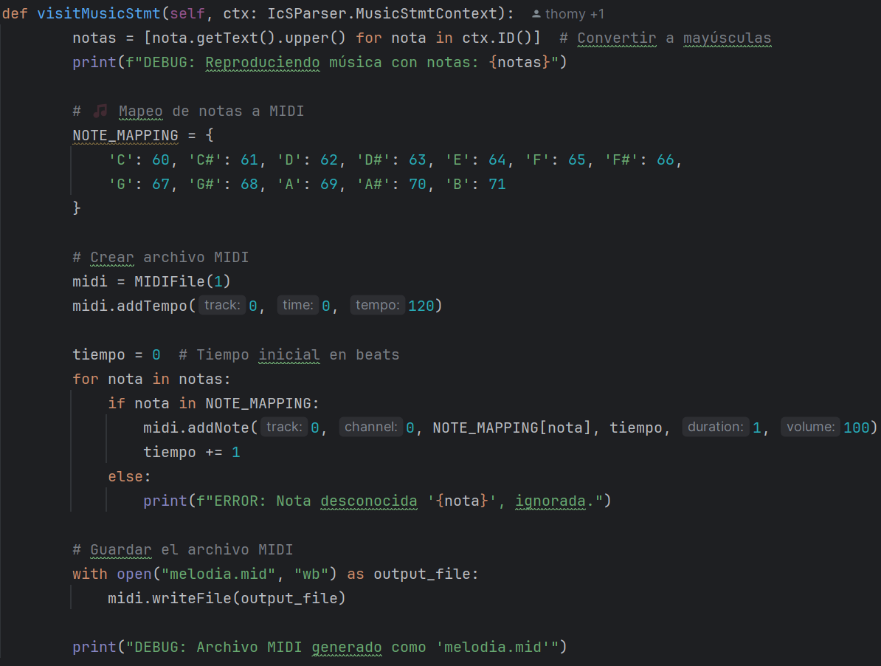


Figura 19. Función visitMusicStmt

La función visitMusicStmt procesa instrucciones para generar música a partir de una secuencia de notas. Convierte las notas en su representación MIDI y las agrega a un archivo utilizando la biblioteca MIDIUtil. Cada nota es traducida a su número MIDI correspondiente y se almacena en un archivo llamado melodia.mid. Si una nota no es reconocida, se muestra un mensaje de error.

## Manual de usuario

Para hacer uso del programa desarrollado en este trabajo, se deben seguir los pasos a mencionar a continuación.

Como primer paso, al descomprimir el archivo, se deberá abrir la consola de comandos y ubicarse justo en la carpeta que resultó del proceso de descompresión, de la forma que se muestra en la siguiente figura.



Figura 20. Consola de comandos de carpeta IcSParadigmas

Una vez se encuentre dentro de la carpeta indicada, se deberá ejecutar el siguiente comando en la consola para realizar la ejecución de las instrucciones especificadas en el archivo llamado codigo.ics, tal como se muestra en la siguiente imagen.



Figura 21. Comando de ejecución del programa

Cabe resaltar que el usuario puede hacer las modificaciones que desee en el archivo que se mostrará a continuación

Haciendo clic en este archivo, desde visual Studio, el usuario podrá realizar las modificaciones pertinentes en el lenguaje de programación.



Figura 22. Archivo código.ics

A continuación, se realizará un ejemplo de lo que sucede al momento de que el usuario escribe la instrucción en consola refiriéndose a la ejecución del programa, tal como se indicó anteriormente. Como ejemplo se usará lo que está especificado en el siguiente archivo, que ya fue seleccionado por el usuario.

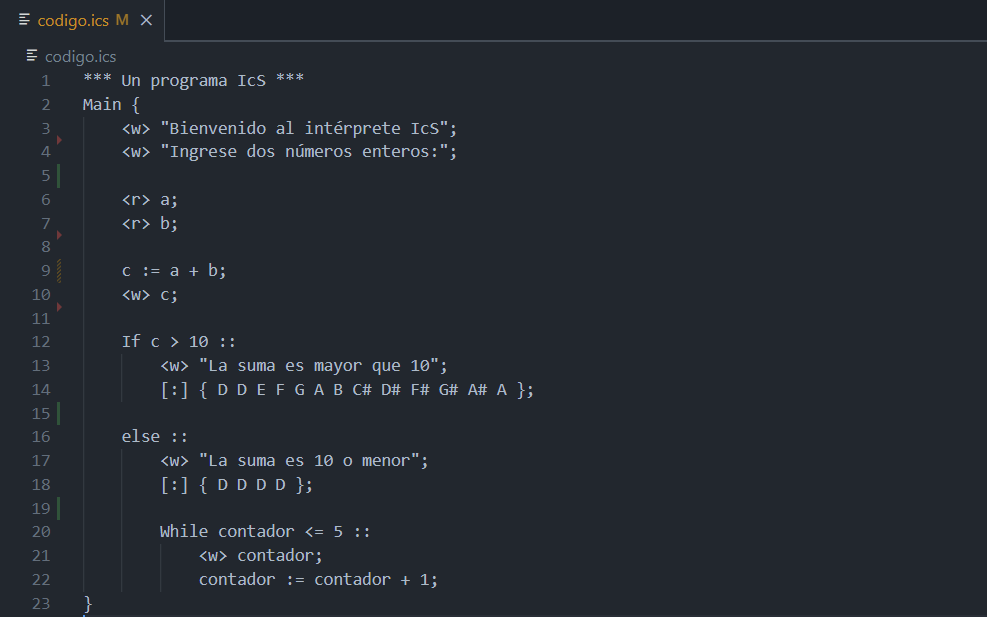


Figura 23. Archivo código.ics

Como breve explicación de lo que se ve en la imagen del archivo ‘codigo.ics’ hay que resaltar que el usuario recibirá un mensaje de bienvenida, tendrá que escribir dos números enteros, uno seguido del otro en la consola de Visual Studio Code y posterior a esto, por consola, se mostrará el valor de la suma de los dos números enteros digitados con anterioridad, que es el valor de c; luego se define el valor de un contador, el cual será usado en el ‘else’.

Como paso siguiente, lo que sucede es que el valor de ‘c’ es valuado en un ‘if’, el cual analiza si el valor de c es mayor que 10 y en el caso de que sea cierto, imprime un mensaje de éxito en consola que idica cuáles son las notas generadas en el archivo de melodías y genera un archivo de tipo Secuencia MIDI llamado ‘melodia.mid’, que es el que se guarda en la carpeta IcSParadigmas , y está al mismo nivel de todos los archivos de la carpeta, como se muestra a continuación.

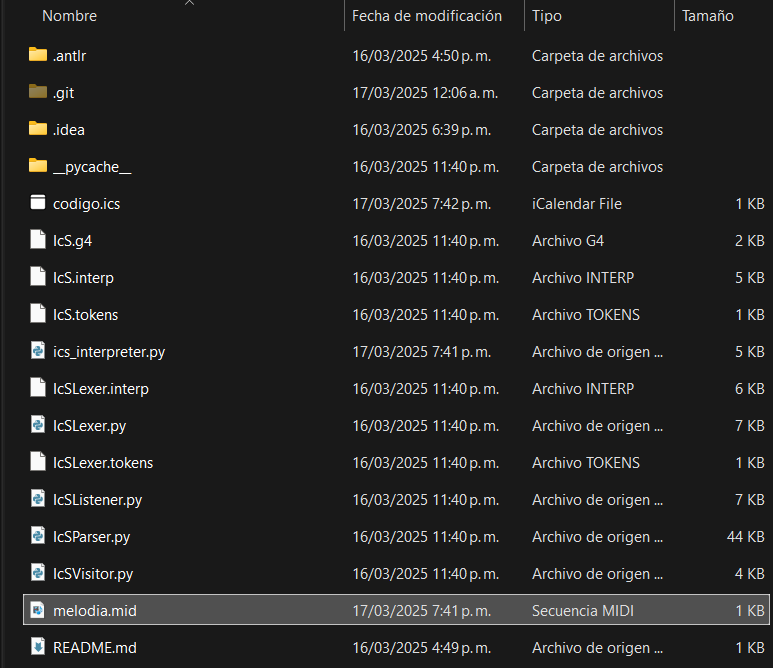


Figura 24. Explorador de archivos de IcSParadigmas

Para ejecutar el archivo y escuchar la melodía, solamente debe hacer doble clic en el archivo generado.

Ahora, en caso de que ‘c’ sea un número igual o menor a 10, se imprimirá un mensaje de éxito en consola, que indica cuáles son las notas generadas en la melodía y además de ello, se entra en un ciclo ‘while’, el cual se encarga de aumentar un contador e imprimirlo a medida que se hacen las iteraciones.

A continuación, se muestra el output final resultante de la ejecución completa del programa si la suma de los dos números enteros es mayor que 10.

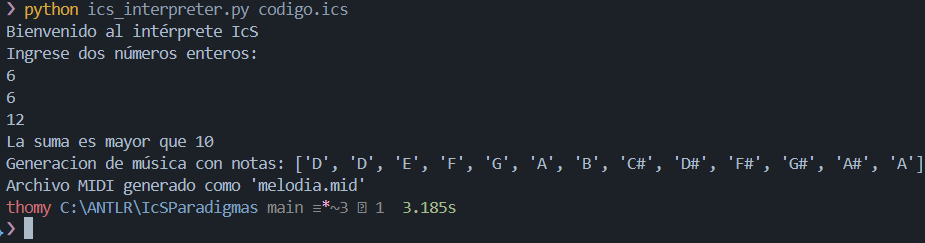


Figura 25. Output del programa ics\_interpreter.py

A continuación, se muestra el output final resultante de la ejecución completa del programa si la suma de los dos números enteros es menor o igual que 10.

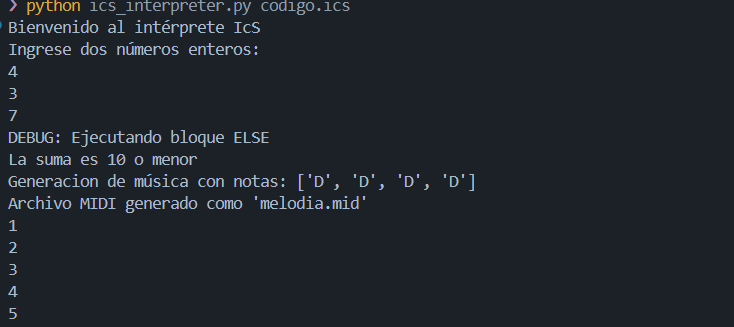


Figura 26. Output del programa ics\_interpreter.py

Finalmente, puede verificar que las notas se han generado, haciendo doble clic en el archivo ‘melodia.mid’ de modo que se abrirá el reproductor de música del equipo, como se muestra en la siguiente imagen.

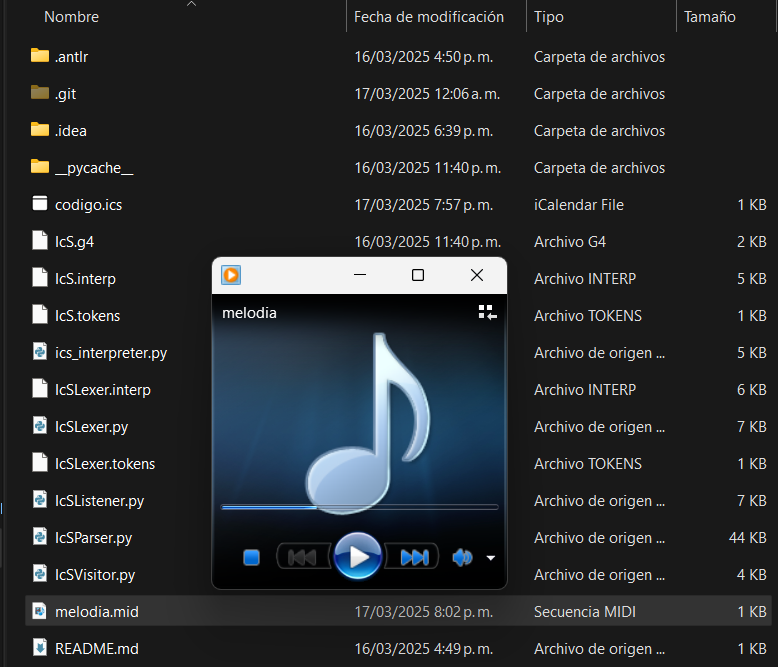


Figura 27. Ejecución del archivo melodía.mid

## Manual técnico

Para poder hacer uso del programa, lo único que se requiere es instalar lo siguiente en la consola de comandos, la cual es una herramienta para generar y manipular archivos MIDI (Musical Instrument Digital Interface) de forma programática.



Figura 28. Instalación de midiutil

# Conclusiones

El desarrollo del intérprete permitió ejecutar correctamente los programas escritos en el lenguaje definido por la gramática, validando su estructura y evaluando sus expresiones. Se logró procesar instrucciones como asignaciones, expresiones aritméticas y condicionales, asegurando una correcta interpretación de las operaciones. Además, la implementación de generación de archivos MIDI confirmó la capacidad del intérprete para procesar comandos musicales. A lo largo del desarrollo, se adquirió un mayor conocimiento sobre el uso del patrón Visitor en ANTLR y la manera en que se recorren árboles de sintaxis abstracta. Finalmente, la depuración y optimización del código facilitaron la comprensión de la interacción entre el lexer, el parser y el visitante, consolidando habilidades en la construcción de lenguajes de programación.

# 

# Lista de referencias

ANTLR; (s.f.). *ANTLR 4 Documentation*. Disponible en : [https://www.antlr.org](https://www.antlr.org/).

MIDI Manufacturers Association; (s.f.). *MIDI 1.0 Specification*. Disponible en : [https://www.midi.org](https://www.midi.org/).

# Vita

**Thomas Alejandro Toro Herrera**: Estudiante de Ingeniería de Sistemas, cursando 5.º semestre, con interés en las áreas de inteligencia artificial y desarrollo de software. Habilidades en Java, JavaScript y TypeScript, además de experiencia en bases de datos y desarrollo web. Estilos de aprendizaje: Estilo Teórico - Pragmático.