INFORME DE TRABAJO 1 (TP)

1ACC0218 – TEORÍA DE COMPILADORES - PRESENCIAL
Carrera de Ciencias de la Computación

Sección: 276
Docente: Peter Jonathan Montalvo García

Integrantes:

Código	Apellidos	Nombres
u202220230	Chipana Rios	Andre Angel
u202316342	Vásquez Trujillano	Edson Fabrizio

Ingreso al Github:: https://github.com/Andre-11c/PharmaLex

2025

ÍNDICE

Problemática y motivación

Los errores en la medicación de los pacientes pueden ser dañinos o incluso mortales. Según AMCP (s.f.), estos errores son los más comunes entre los errores médicos y perjudican a más de un millón de pacientes anualmente. Algunos de estos errores incluyen problemas con productos de la salud defectuosos, fallas en la administración de medicamentos, la deficiente educación del paciente ante el uso de medicamentos, errores en la prescripción dada por el especialista, entre otros.

Teniendo en cuenta este contexto, en los últimos años se han observado errores frecuentes en la prescripción de medicamentos debido a recetas ilegibles. Por ejemplo, Redacción Mag (2023) comparte un caso de un farmacéutico que no fue capaz de interpretar la receta médica de un cliente debido a que la letra era ilegible e incluso pudo estar escrita en jerga médica o abreviaturas específicas. Este caso, al igual que muchos otros, demuestra el desafío que enfrentan los farmacéuticos para comprender las indicaciones médicas de las recetas y proveer a las personas de los medicamentos correctos.

Ante esta problemática, proponemos una solución enfocada al desarrollo y uso de una gramática que sirva como validación para la prescripción de medicamentos. Esta gramática se implementará en el sistema del centro médico, permitiendo que el especialista de la salud ingrese la información de la receta médica o el tratamiento para su registro, de forma que cada dato registrado coincida con las reglas establecidas en la gramática. Por ejemplo, si se desea registrar el nombre de un medicamento, solo será aceptado si posee caracteres alfabéticos; en cambio, si se registra una dosis, el formato debe contener caracteres numéricos para la medida y caracteres alfabéticos para la unidad de medida.

Objetivos

El objetivo principal de este proyecto es solucionar o reducir los errores causados por prescripciones inadecuadas. Para ello, proponemos la creación de un lenguaje de programación que contenga reglas gramaticales destinadas a validar cada campo de la prescripción médica. Una vez la prescripción sea validada y registrada en el sistema del centro médico, la propia entidad puede entregar al paciente una versión impresa o digital, legible y clara, de su receta médica.

Además, como objetivo secundario, la propuesta contempla otras aplicaciones, tales como su integración con asistentes de voz para agilizar el registro de datos en comparación con el proceso manual tradicional, así como su uso en la generación automática de archivos para sistemas de base de datos (por ejemplo, en formatos .json, .csv, entre otros).

Gramática en ANTLR4

A fin de establecer las reglas gramaticales del lenguaje de programación, es necesario conocer los campos de una receta médica. Según el Instituto Nacional de Salud Mental (s.f.), la información que debe contener una receta médica es la siguiente:

Campo	**Tipo de carácteres**	**Descripción**
Paciente	Alfabético	Nombre completo del paciente
DNI	Numérico	Documento Nacional de Identidad o Código Único de Identificación de Extranjeros
Diagnóstico	Alfabético	Descripción de la enfermedad o estado de salud del paciente.
Medicamento	Alfabético	Nombre del producto farmacéutico recetado.
Concentración	Alfanumérico	Medida y unidad de medida de medicamento (por ejemplo, 500 mg).
Forma	Alfabético	Forma farmacéutica del medicamento (por ejemplo, tableta, jarabe, ampolleta, etc.).
Cantidad	Numérico	Cantidad total de medicamento que debe ingerir el paciente.
Expedición	Numérico y símbolos	Fecha de expedición en formato dd-mm-aa.
Caducidad	Numérico y símbolos	Fecha de vigencia de la receta en formato dd-mm-aa.
Indicación	Alfanumérico	Detalle de cómo y cuándo debe administrarse el medicamento
Vía	Alfabético	Vía de administración de medicamento.

En ese sentido, establecemos las reglas léxicas de la gramática siguiendo las convenciones descritas en la tabla.

Reglas léxicas originales:

```
PACIENTE : [a-zA-ZáéíóúÁÉÍÓÚñÑ]+ ( ' ' [a-zA-ZáéíóúÁÉÍÓÚñÑ]+)* ;
DNI : [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9] [0-9];
DIAGNOSTICO : [a-zA-ZáéíóúÁÉÍÓÚñÑ]+ ( ' ' [a-zA-ZáéíóúÁÉÍÓÚñÑ]+ )* ;
MEDICAMENTO : [a-zA-ZáéíóúÁÉÍÓÚñÑ]+ ;
CONCENTRACION : [0-9]+ ' ' [a-zA-ZáéíóúÁÉÍÓÚñÑ]+ ;
FORMA : [a-zA-ZáéíóúÁÉÍÓÚñÑ]+ ;
```

CANTIDAD : [0-9]+ ;
EXPEDICION : [0-9][0-9] '-' [0-9][0-9] '-' [0-9][0-9] ;
CADUCIDAD : [0-9][0-9] '-' [0-9][0-9] '-' [0-9][0-9] ;
INDICACION : ([0-9]+ | [a-zA-ZáéíóúÁÉÍÓÚñÑ]+ | ',' | '.' | ' ')+ ;
VIA : [a-zA-ZáéíóúÁÉÍÓÚñÑ]+ ;

Sin embargo, estas reglas presentan definiciones muy similares o, en algunos casos, completamente idénticas. Esto genera ambigüedad léxica, ya que, si una entrada cumple con las reglas léxicas de varios tokens, el analizador léxico no podrá identificar correctamente a qué token pertenece. Por ejemplo, si se ingresa la palabra “Hola”, el lexer no podrá determinar si corresponde al token PACIENTE, DIAGNOSTICO, MEDICAMENTO, FORMA, INDICACION o VIA, ya que todos permiten el ingreso de una sola palabra.

Para resolver este problema, optamos por establecer reglas más simples, usando descripciones más generales que eviten la ambigüedad y de forma que permitan establecer las reglas sintácticas esperadas.

Reglas léxicas:

PALABRAS : [a-zA-ZáéíóúÁÉÍÓÚñÑ]+ (' ' [a-zA-ZáéíóúÁÉÍÓÚñÑ]+)* ;
INT : [0-9] ;
CONCENTRACION : INT+ ' ' PALABRAS ;
FECHA : INT INT '-' INT INT '-' INT INT ;

Una vez establecidas las reglas léxicas, procedemos a describir las reglas sintácticas que dan sentido a nuestra gramática al crear definiciones de cómo se combinan los tokens para construir expresiones. A continuación, se muestran las subreglas sintácticas diseñadas para cada oración que debe reconocer el sistema:

Subreglas sintácticas:

- datosPacienteSentence:

  Arquitectura que describe la acción de ingresar el nombre completo y DNI de un paciente.

  Definición: 'Paciente:' WS* PALABRAS ',' WS* INT INT INT INT INT INT INT INT
  Ejemplo: “Paciente: Jose Raúl Ramirez Borgoña, 12345678”

- diagnosticoSentence:

  Arquitectura que describe la acción de ingresar el diagnóstico del paciente.

Definición: 'Diagnostico:' WS* PALABRAS
Ejemplo: “"Diagnostico: Arritmia severa”

- fechasSentence:

  Arquitectura que describe el ingreso de una fecha. En este caso, es la base para ingresar la fecha de expedición y la fecha de caducidad de la receta médica.

  Definición: 'Expede:' WS* FECHA WS* 'Caduce:' WS* FECHA
  Ejemplo: “Expede: 10-05-2025 Caduce: 18-10-2025”

- medicamentoSentence:

  Arquitectura que describe el ingreso de los datos de un medicamento en el siguiente orden: nombre, concentración, forma, vía, cantidad e indicación.

  Definición: 'Medicamento:' WS* PALABRAS ',' WS* CONCENTRACION ',' WS* PALABRAS ',' WS* PALABRAS ',' WS* INT+ ',' WS* PALABRAS '.'
  Ejemplo: "Medicamento: Paracetamol, 500 mg, Tableta, Oral, 5, Tomar cada ocho horas."

Finalmente, establecemos unas reglas sintácticas simples para poner a prueba la gramática creada.

Regla sintáctica:

datosPacienteSentence EOF
| diagnosticoSentence EOF
| medicamentoSentence EOF
| fechasSentence EOF

A continuación, se adjuntan imágenes de la gramática implementada en Google Colab, por medio del lenguaje de programación Python.

Link del Google Colab:

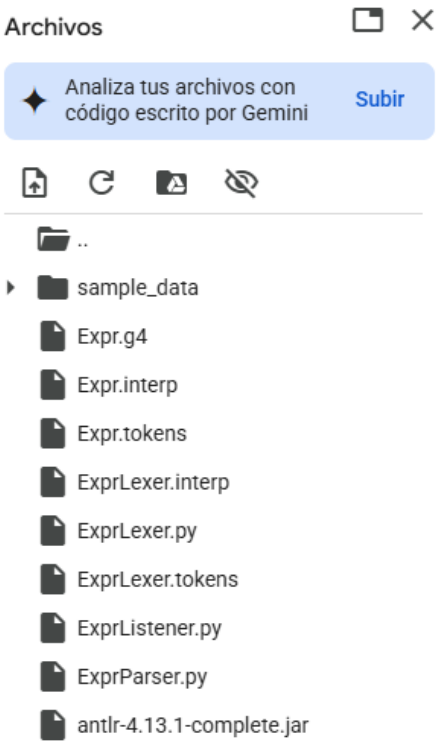
https://colab.research.google.com/drive/1I1O0Vng96My7Mp1aHi-eGVADGl71wcnM?usp=sharing

Gramática:

```
gramatica = """
grammar Expr;

// --- Reglas Sintácticas - Parser ---
prog:
    datosPacienteSentence EOF
    | diagnosticoSentence EOF
    | medicamentoSentence EOF
    | fechasSentence EOF
    ;

// --- SubReglas Sintácticas - Parser ---
datosPacienteSentence:
    'Paciente:' WS* PALABRAS ',' WS* INT INT INT INT INT INT INT INT
    ;

diagnosticoSentence:
    'Diagnostico:' WS* PALABRAS
    ;

fechasSentence:
    'Expede:' WS* FECHA WS* 'Caduce:' WS* FECHA
    ;

medicamentoSentence:
    'Medicamento:' WS* PALABRAS ',' WS* CONCENTRACION ',' WS* PALABRAS ',' WS* PALABRAS ',' WS* INT+ ',' WS* PALABRAS '.'
    ;

// --- Reglas Léxicas (Tokens) - Lexer ---
PALABRAS :   [a-zA-ZáéíóúÁÉÍÓÚñÑ]+ ( ' ' [a-zA-ZáéíóúÁÉÍÓÚñÑ]+ )* ;
INT :      [0-9] ;
CONCENTRACION : INT+ ' ' PALABRAS ;
FECHA :    INT INT '-' INT INT '-' INT INT ;

WS  : [ \\t\\r\\n]+ -> skip ;
"""

with open("Expr.g4", "w", encoding='utf-8', newline='\n') as f:
    f.write(gramatica)
```

Generar archivos G4:

```
[5]  # Genera varios archivos Lexer.py, Parser.py, .tokens, etc.
     #   Sirven para interpretar y ejecutar texto segun las reglas de la gramatica
     !java -jar antlr-4.13.1-complete.jar -Dlanguage=Python3 Expr.g4
```



Pruebas de gramática:

```
[6]  from antlr4 import *
     from ExprLexer import ExprLexer   # Importar Lexer del compilador
     from ExprParser import ExprParser # Importar Parser del compilador

     def parse_expression(input_text): # ingresa un String - texto
         input_stream = InputStream(input_text)  # analizar texto
         lexer = ExprLexer(input_stream)         # pasar al lexer
         token_stream = CommonTokenStream(lexer) # pasar a los tokens
         parser = ExprParser(token_stream)       # pasar al parser
         tree = parser.prog()                    # obtener arbol semantico
         return tree.toStringTree(recog=parser)  # convertir arbol a String
```

```
[7]  # Prueba 1:
     print(parse_expression("Paciente:Juan Pérez Gómez, 12345678\n"))
```

```
(prog (datosPacienteSentence Paciente: Juan Pérez Gómez , 1 2 3 4 5 6 7 8) <EOF>)
```

```
[8]  # Prueba 2:
     print(parse_expression("Diagnostico: Gripe común\n"))
```

```
(prog (diagnosticoSentence Diagnostico: Gripe común) <EOF>)
```

```
[9]  # Prueba 3:
     print(parse_expression("Medicamento: Paracetamol, 500 mg, Tableta, Oral, 5, Tomar cada ocho horas.\n"))
```

```
(prog (medicamentoSentence Medicamento: Paracetamol , 500 mg , Tableta , Oral , 5 , Tomar cada ocho horas .) <EOF>)
```

```
     # Prueba 4:
     print(parse_expression("Expede: 01-05-25 Caduce: 01-06-25\n"))
```

```
(prog (fechasSentence Expede: 01-05-25 Caduce: 01-06-25) <EOF>)
```

Referencias bibliográficas

Academy of Managed Care Pharmacy (AMCP). (s.f.) Medication Errors. *AMCP*. Recuperado el 12 de mayo de 2025, de https://www-amcp-org.translate.goog/concepts-managed-care-pharmacy/medication-errors?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=sge#:~:text=Common%20causes%20of%20medication%20error,and%20lack%20of%20patient%20education.

Redacción Mag. (30 de diciembre de 2023). Un farmacéutico pide ayuda para entender lo que dice esta receta: ¿logras descifrarlo?. *El Comercio*. https://elcomercio.pe/mag/virales/video-viral-farmaceutico-pide-ayuda-para-entender-lo-que-dice-esta-receta-espana-nnda-nnrt-noticia/?ref=ecr

Instituto Nacional de Salud Mental. (s.f.). Requisitos que debe contener una receta médica. *Ministerio de Salud.* Recuperado el 14 de mayo de 2025, de https://www.insm.gob.pe/comites/farmacoterapeutico/archivos/170707%20REQUISITOS%20QUE%20DEBE%20CONTER%20UNA%20RECETA.pdf