Analizador de codigo con PLY LEX

Este programa es un **analizador léxico y semántico de código** que puede procesar archivos de C++ y Python línea por línea. Su objetivo principal es clasificar cada línea de código según su propósito semántico (qué tipo de instrucción representa).

Componentes Principales

Analizadores Léxicos (Lexers)

- **BaseLexer**: Clase base que define tokens comunes como operadores, números, cadenas, paréntesis, etc.
- **CPPLexer**: Extiende BaseLexer para reconocer tokens específicos de C++ como `#include`, `cout`, `cin`, `class`, etc.
- **PythonLexer**: Extiende BaseLexer para reconocer tokens específicos de Python como `def`, `print`, `input`, `class`, etc.

Analizador de Líneas (LineAnalyzer)

- Recibe una lista de tokens de una línea específica
- Aplica reglas heurísticas para clasificar la línea en categorías semánticas
- Mapea los tipos técnicos a descripciones en español comprensibles

Analizador Principal (CodeAnalyzer)

- Integra el lexer correspondiente al lenguaje con el analizador de líneas
- Lee archivos línea por línea y produce un análisis completo

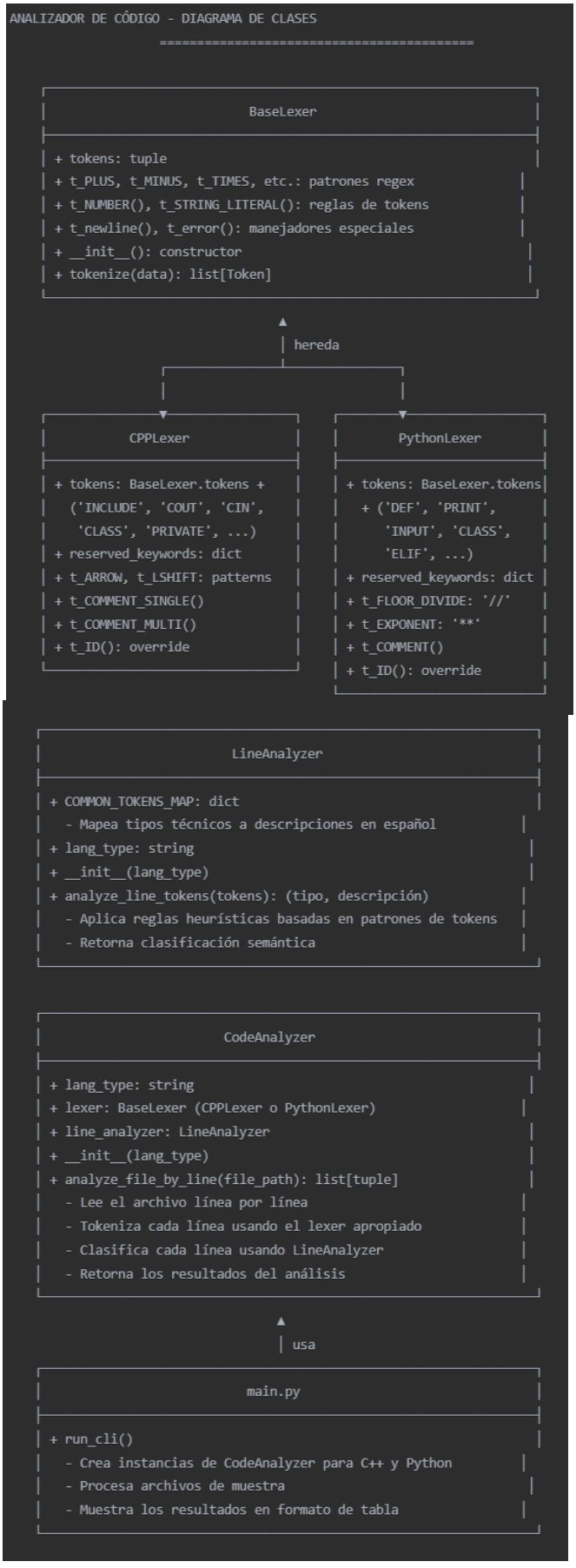
Proceso de Análisis

1. **Tokenización**: Cada línea se descompone en tokens léxicos básicos
2. **Clasificación Semántica**: Los tokens se analizan para determinar qué tipo de instrucción representan
3. **Presentación**: Los resultados se muestran en una tabla con la línea original, el tipo semántico y su descripción

Tipos de Clasificación Semántica

El programa puede identificar:

- Definiciones y llamadas a funciones
- Declaraciones y asignaciones de variables
- Estructuras de control (if/else, bucles)
- Sentencias de entrada/salida
- Comentarios y líneas vacías
- Definiciones de clases
- Directivas del preprocesador (C++)


ANALIZADOR DE CÓDIGO - DIAGRAMA DE CLASES
BaseLexer
+ tokens: tuple
+ t_PLUS, t_MINUS, t_TIMES, etc.: patrones regex
+ t_NUMBER(), t_STRING_LITERAL(): reglas de tokens
+ t_newline(), t_error(): manejadores especiales
+ __init__(): constructor
+ tokenize(data): list[Token]
hereda
CPPLexer
+ tokens: BaseLexer.tokens +
('INCLUDE', 'COUT', 'CIN',
'CLASS', 'PRIVATE', ...)
+ reserved_keywords: dict
+ t_ARROW, t_LSHIFT: patterns
+ t_COMMENT_SINGLE()
+ t_COMMENT_MULTI()
+ t_ID(): override
PythonLexer
+ tokens: BaseLexer.tokens
+ ('DEF', 'PRINT',
'INPUT', 'CLASS',
'ELIF', ...)
+ reserved_keywords: dict
+ t_FLOOR_DIVIDE: '//'
+ t_EXPONENT: '**'
+ t_COMMENT()
+ t_ID(): override
LineAnalyzer
+ COMMON_TOKENS_MAP: dict
- Mapea tipos técnicos a descripciones en español
+ lang_type: string
+ __init__(lang_type)
+ analyze_line_tokens(tokens): (tipo, descripción)
- Aplica reglas heurísticas basadas en patrones de tokens
- Retorna clasificación semántica
CodeAnalyzer
+ lang_type: string
+ lexer: BaseLexer (CPPLexer o PythonLexer)
+ line_analyzer: LineAnalyzer
+ __init__(lang_type)
+ analyze_file_by_line(file_path): list[tuple]
- Lee el archivo línea por línea
- Tokeniza cada línea usando el lexer apropiado
- Clasifica cada línea usando LineAnalyzer
- Retorna los resultados del análisis
usa
main.py
+ run_cli()
- Crea instancias de CodeAnalyzer para C++ y Python
- Procesa archivos de muestra
- Muestra los resultados en formato de tabla

```
                        FLUJO DE PROCESAMIENTO
                        ======================


   archivo.cpp/py  →  CodeAnalyzer  →  [línea por línea]
                          │
                          ├─→ Lexer (CPP/Python)  →  [tokens]
                          │
                          ├─→ LineAnalyzer  →  [clasificación semántica]
                          │
                          └─→ Resultados  →  [tabla formateada]

                      TIPOS DE CLASIFICACIÓN SEMÁNTICA
                      ================================

   • FUNCTION_DEF     → Definición de Función
   • FUNCTION_CALL    → Llamada a Función
   • VAR_DECL         → Declaración de Variable
   • ASSIGN           → Asignación de Variable
   • PRINT_STMT       → Sentencia de Impresión
   • INPUT_STMT       → Sentencia de Entrada
   • BRANCH           → Sentencia Condicional (if/else)
   • FOR_LOOP         → Bucle For
   • WHILE_LOOP       → Bucle While
   • CLASS_DEF        → Definición de Clase
   • RETURN           → Sentencia de Retorno
   • COMMENT          → Comentario
   • EMPTY_LINE       → Línea Vacía/Ignorada
   • CALCULATIONS     → Cálculo/Expresión
```

Probando:

```
Contenido del archivo:


using namespace std;
int main(){
    int numero;
    cout << "Escribe un numero: " ;
    cin >> numero ;
    int* ptrNumero = &numero ;
    cout << (*ptrNumero%2 == 0 ? "Es par" : "No es par") << endl ;
    cout << ptrNumero ;
}


--------------------------------------------------


--- Resultados del Análisis (C++: archivo_cpp.txt) ---
+------------------------------------------------------------+-------------------------+-------------------------+
| Línea de Código                                            | Tipo de Token Semántico | Descripción             |
+============================================================+=========================+=========================+
|                                                            | EMPTY_LINE              | Línea Vacía/Ignorada    |
+------------------------------------------------------------+-------------------------+-------------------------+
| using namespace std;                                       | USING_NAMESPACE         | Uso de Namespace        |
+------------------------------------------------------------+-------------------------+-------------------------+
| int main(){                                                | FUNCTION_DEF            | Definición de Función   |
+------------------------------------------------------------+-------------------------+-------------------------+
| int numero;                                                | VAR_DECL                | Declaración de Variable |
+------------------------------------------------------------+-------------------------+-------------------------+
| cout << "Escribe un numero: " ;                            | PRINT_STMT              | Sentencia de Impresión  |
+------------------------------------------------------------+-------------------------+-------------------------+
| cin >> numero ;                                            | INPUT_STMT              | Sentencia de Entrada    |
+------------------------------------------------------------+-------------------------+-------------------------+
| int* ptrNumero = &numero ;                                 | VAR_DECL                | Declaración de Variable |
+------------------------------------------------------------+-------------------------+-------------------------+
| cout << (*ptrNumero%2 == 0 ? "Es par" : "No es par") << endl ; | PRINT_STMT              | Sentencia de Impresión  |
+------------------------------------------------------------+-------------------------+-------------------------+
| cout << ptrNumero ;                                        | PRINT_STMT              | Sentencia de Impresión  |
+------------------------------------------------------------+-------------------------+-------------------------+
| }                                                          | UNKNOWN                 | Desconocido             |
+------------------------------------------------------------+-------------------------+-------------------------+
--------------------------------------------------
```

```
--- Analizando el archivo 'archivo_python.txt' (python) ---

Contenido del archivo:

import sys

def greet(name):
    message = "Hello, " + name + "!"
    if name == "world":
        print(message)
    else:
        print("Nice to meet you, " + name)
    return message

class MyClass:
    def __init__(self, value):
        self.value = value
        self.other_value = value * 2

my_name = "Alice"
result = greet(my_name) # Llamada a función
# Esto es un comentario en Python
# Otra línea de comentario
print("Analysis complete.")

--------------------------------------------------

Ocurrió un error al leer o analizar el archivo: D:\COMPILADORES\src\lexer_python.py:63: Rule 't_ID' returned an unknown token type 'IF'
```