



Tecnológico de Monterrey

Analizador Sintactico

Luz Patricia Hernández Ramírez A01637277

Implementación de métodos computacionales

Grupo 602

30 de mayo 2024

Analizador Sintactico

Introducción

El presente trabajo es una aplicación de la jerarquía de Chomsky, específicamente el uso de las reglas gramaticales para lograr el análisis de sintaxis en el contexto de la programación de expresiones aritméticas. Partiendo de un analizador léxico, el desarrollo de este analizador sintáctico requirió la elaboración de reglas gramaticales, una función *parser* y un árbol de derivación. A continuación, se muestran las reglas gramaticales.

Reglas Gramaticales (regex)

```
Program    -> "Programa" "{" Principal "}"
Principal  -> "principal" "{" Block "}"
Block      -> { Statement }
Statement  -> VarDecl | Expr ";"
VarDecl    -> ("Entero" | "Real") Variable "=" Expr ";"
Expr       -> Term { ("+" | "-") Term }
Term       -> Factor { ("*" | "/") Factor }
Factor     -> Variable | Integer | Real | "(" Expr ")"
Variable   -> [a-zA-Z_][a-zA-Z0-9_]*
Integer    -> [0-9]+
Real       -> [0-9]+("." [0-9]*)?("E" ("+" | "-")? [0-9]+)?
```

Reglas en gramática libre de contexto

```
Programa -> "Programa" "{" Principal "}"
Principal -> "principal" "{" Bloque "}"
Bloque   -> Sentencia Bloque | ε
Sentencia -> DeclaracionVar | Expr ";"
```

```

DeclaracionVar -> Tipo Variable "=" Expr ";"
Tipo -> "Entero" | "Real"
Expr -> Term Expr'
Expr' -> "+" Term Expr' | "-" Term Expr' | ε
Term -> Factor Term'
Term' -> "*" Factor Term' | "/" Factor Term' | ε
Factor -> Variable | Entero | Real | "(" Expr ")"
Variable -> Letra Variable'
Letra -> [a-zA-Z_]
LetraDigito -> [a-zA-Z0-9_]
Entero -> Digito Entero'
Entero' -> Digito Entero' | ε
Entero' -> Digito Entero' | ε
Real -> Entero "." ParteDecimal ParteExponente
ParteDecimal -> Digito ParteDecimal' | ε
ParteExponente -> "E" Signo Entero | ε
Signo -> "+" | "-" | ε

```

Objetivo

El objetivo de estas reglas es lograr leer archivos de texto (como los planteados a continuación) para establecer un análisis sintáctico y demostrar su funcionamiento mediante un árbol de derivación:

Ejemplo de archivo de texto

```

Programa {

    principal ()

    {

        Entero a = 4;

        Real b = 5.8E-8;

        Entero serie = a * (b * c) + 9;

    }

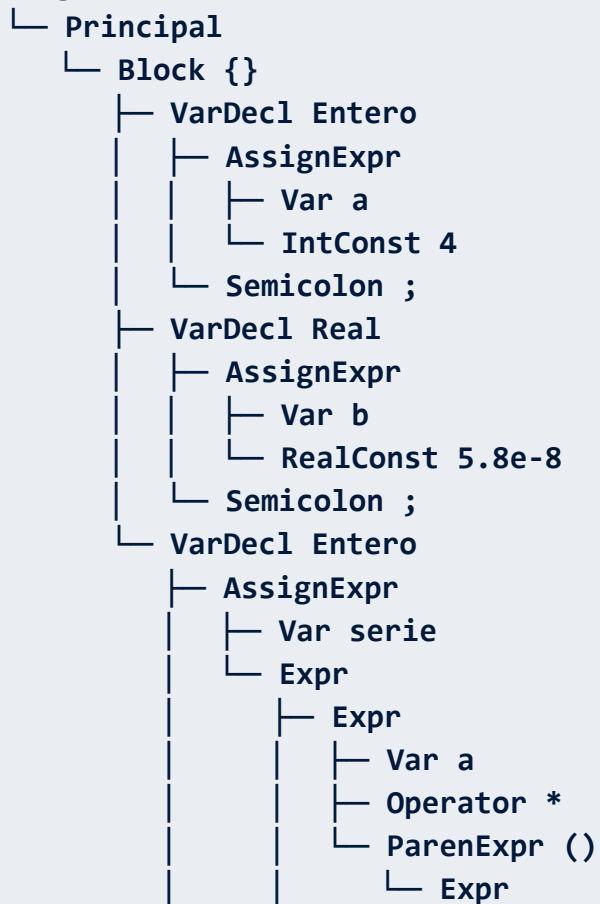
}

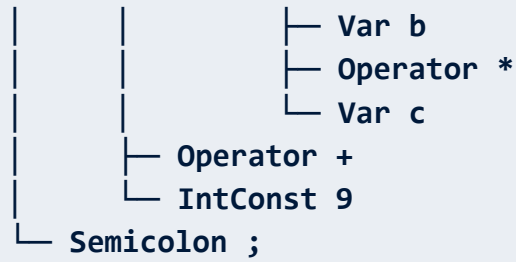
```

Ejemplo árbol de derivación

Abstract Syntax Tree:

Program





Manual de usuario

Instalar

```
cabal parser
```

Compilar

```
ghc -o analizadorS AnalizadorSintactico.hs
```

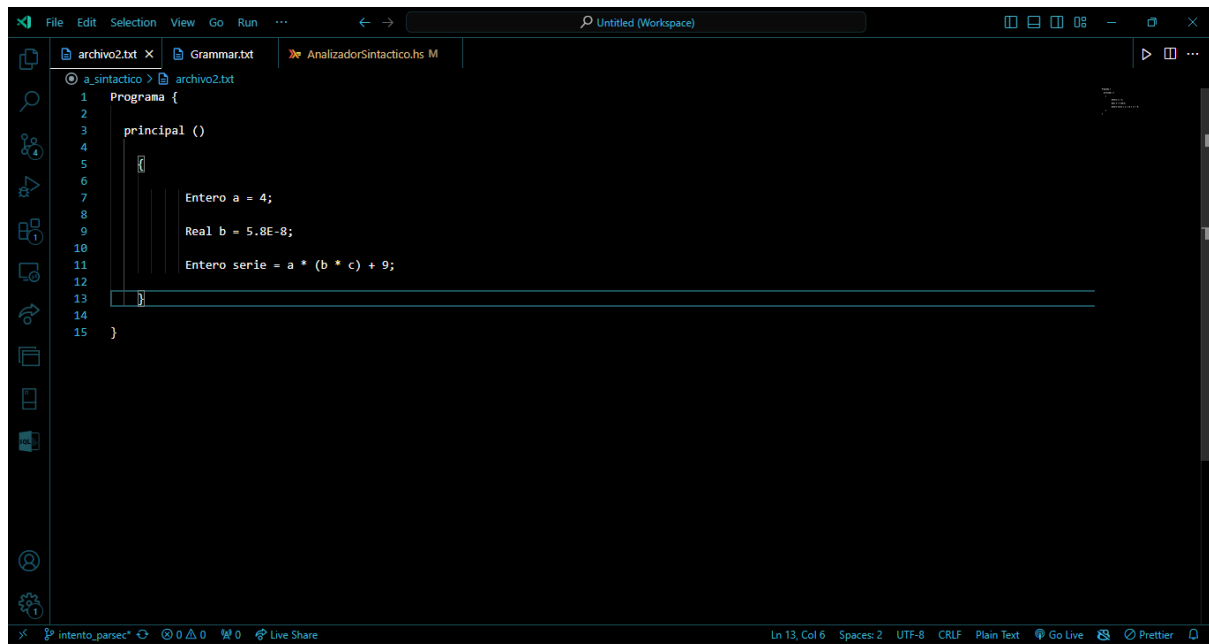
Correr

```
./analizadorS archivo2.txt
```

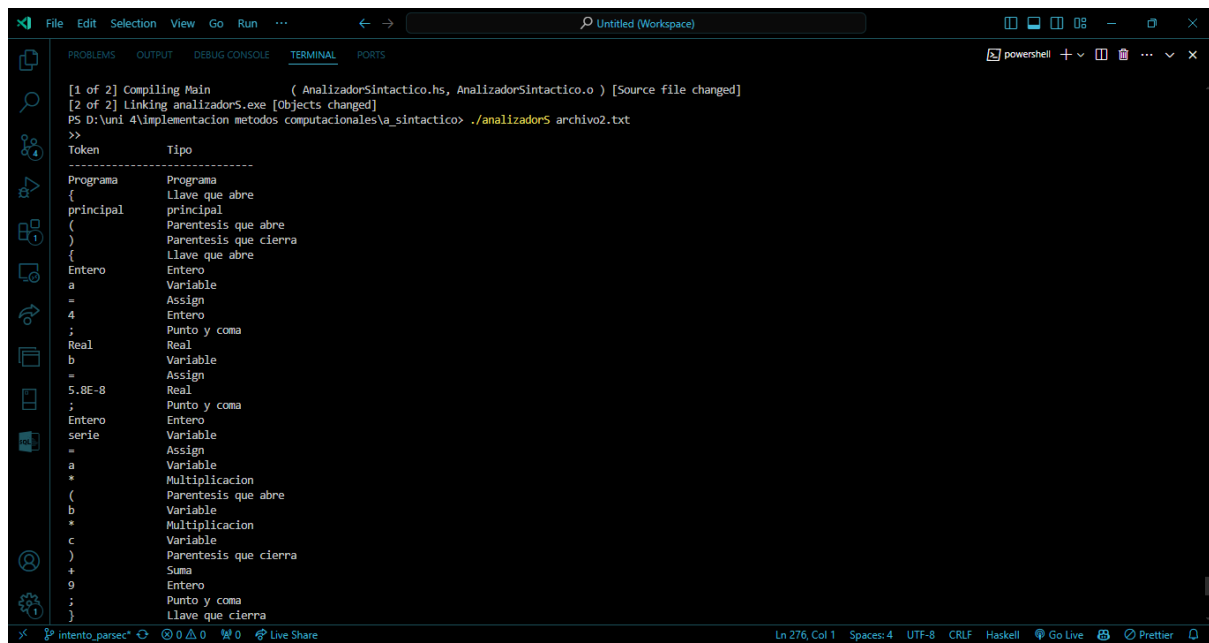
La carpeta contiene diferentes archivos txt nombrados archivo1.txt, archivo2.txt, archivo3.txt, etc. Cada uno tiene diferentes casos de prueba que pueden ser leídos en caso de que así se desee. De otro modo puede usar cualquiera de estos archivos para correr y probar el programa. En el manual de usuario específico el 2 pues contiene el ejemplo de las instrucciones.

Test cases

Case 1



```
File Edit Selection View Go Run ... Untitled (Workspace)
archivo2.txt x Grammar.txt AnalizadorSintactico.hs M
a_sintactico > archivo2.txt
1 Programa {
2
3     principal ()
4
5     {
6
7         Entero a = 4;
8
9         Real b = 5.8E-8;
10
11         Entero serie = a * (b * c) + 9;
12
13     }
14
15 }
```



```
File Edit Selection View Go Run ... Untitled (Workspace)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + - ... v x
[1 of 2] Compiling Main (AnalizadorSintactico.hs, AnalizadorSintactico.o) [Source file changed]
[2 of 2] Linking analizador5.exe [Objects changed]
PS D:\uni 4\implementacion metodos computacionales\va_sintactico> ./analizador5 archivo2.txt
>>
Token      Tipo
-----
Programa   Programa
{           Llave que abre
principal  principal
(           Parentesis que abre
)           Parentesis que cierra
{           Llave que abre
Entero      Entero
a           Variable
=           Assign
4           Entero
;           Punto y coma
Real        Real
b           Variable
=           Assign
5.8E-8      Real
;           Punto y coma
Entero      Entero
serie       Variable
=           Assign
a           Variable
*           Multiplicacion
(           Parentesis que abre
b           Variable
*           Multiplicacion
c           Variable
)           Parentesis que cierra
+           Suma
9           Entero
;           Punto y coma
}           Llave que cierra
```

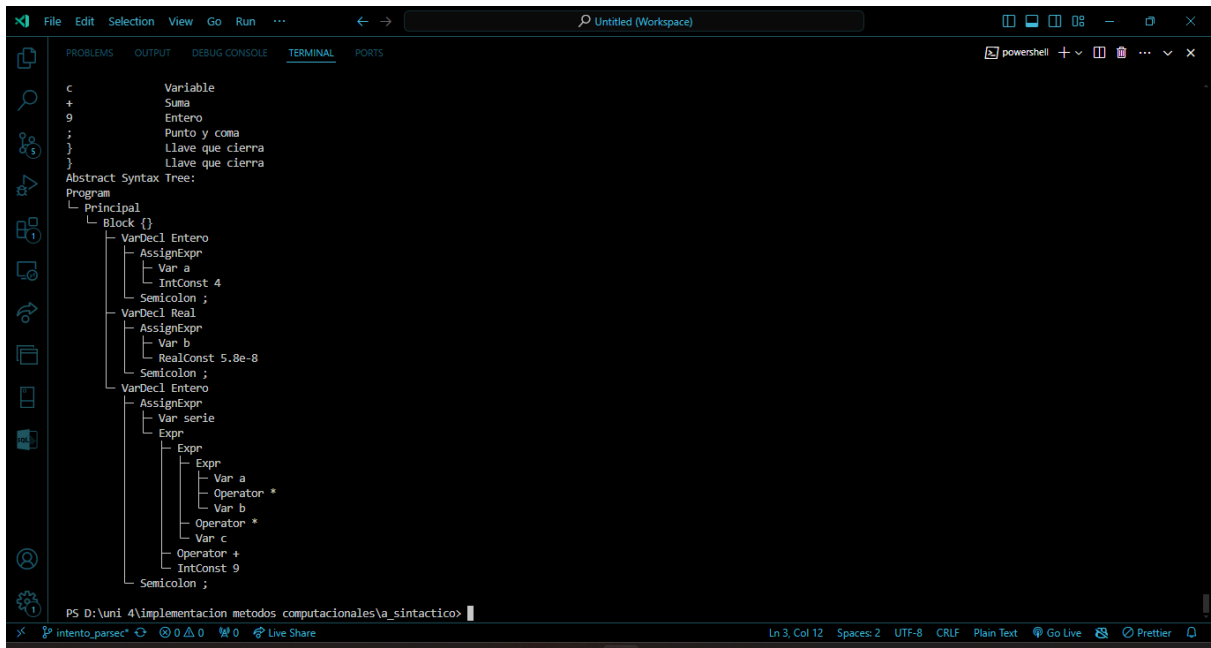
```
+ Suma
9 Entero
; Punto y coma
} Llave que cierra
} Llave que cierra

Abstract Syntax Tree:
Program
├── Principal
│   └── Block {}
│       ├── VarDecl Entero
│       │   ├── AssignExpr
│       │   │   ├── Var a
│       │   │   └── IntConst 4
│       │   └── Semicolon ;
│       ├── VarDecl Real
│       │   ├── AssignExpr
│       │   │   ├── Var b
│       │   │   └── RealConst 5.8e-8
│       │   └── Semicolon ;
│       └── VarDecl Entero
│           ├── AssignExpr
│           │   ├── Var serie
│           │   └── Expr
│           │       ├── Expr
│           │       │   ├── Var a
│           │       │   ├── Operator *
│           │       │   └── ParenExpr ()
│           │       │       ├── Expr
│           │       │       │   ├── Var b
│           │       │       │   ├── Operator *
│           │       │       │   └── Var c
│           │       └── Operator +
│           │           └── IntConst 9
│           └── Semicolon ;
```

Case 2

```
>> D:\uni 4\implementacion metodos computacionales\va_sintactico>
Token
Tipo
-----
Programa Programa
{ Llave que abre
principal principal
{ Llave que abre
Entero Entero
a Variable
= Assign
4 Entero
; Punto y coma
Real Real
b Variable
= Assign
5.8E-8 Real
; Punto y coma
Entero Entero
serie Variable
= Assign
a Variable
* Multiplicacion
b Variable
* Multiplicacion
c Variable
+ Suma
9 Entero
; Punto y coma
} Llave que cierra
} Llave que cierra

Abstract Syntax Tree:
Program
├── Principal
│   └── Block {}
│       ├── VarDecl Entero
│       │   ├── AssignExpr
```



The screenshot shows a Visual Studio Code window with a terminal. The terminal is running a command to parse a file named 'archivo2.txt' using a parser named 'analizadorS'. The output shows a list of tokens and their types, followed by an Abstract Syntax Tree (AST) for the program.

```

>> D:\uni 4\implementacion metodos computacionales\va_sintactico> ./analizadorS archivo2.txt
Token      Tipo
-----
Programa   Programa
{          Llave que abre
principal  principal
{          Llave que abre
Entero     Entero
a          Variable
=          Assign
4          Entero
;          Punto y coma
Real       Real
b          Variable
=          Assign
5.8E-8     Real
;          Punto y coma
Entero     Entero
serie      Variable
=          Assign
a          Variable
*          Multiplicacion
b          Variable
*          Multiplicacion
c          Variable
+          Suma
9          Entero
;          Punto y coma
}          Llave que cierra
}          Llave que cierra
Abstract Syntax Tree:

```

The screenshot shows a Visual Studio Code window with a terminal displaying a detailed Abstract Syntax Tree (AST) for a program. The AST is a tree structure representing the syntactic structure of the code. The root node is 'Program', which has a child 'Principal', which in turn has a child 'Block {}'. The 'Block {}' node has three children: 'VarDecl Entero', 'VarDecl Real', and 'VarDecl Entero'. Each 'VarDecl' node has an 'AssignExpr' child, which further branches into 'Var' and 'Expr' nodes. The 'Expr' nodes represent the expressions assigned to the variables, such as 'IntConst 4', 'RealConst 5.8e-8', and a more complex expression involving multiplication and addition of variables and constants.

```

c          Variable
+          Suma
9          Entero
;          Punto y coma
}          Llave que cierra
}          Llave que cierra
Abstract Syntax Tree:
Program
├── Principal
│   └── Block {}
│       ├── VarDecl Entero
│       │   ├── AssignExpr
│       │   │   ├── Var a
│       │   │   │   ├── IntConst 4
│       │   │   │   └── Semicolon ;
│       │   └── Semicolon ;
│       ├── VarDecl Real
│       │   ├── AssignExpr
│       │   │   ├── Var b
│       │   │   │   ├── RealConst 5.8e-8
│       │   │   │   └── Semicolon ;
│       │   └── Semicolon ;
│       └── VarDecl Entero
│           ├── AssignExpr
│           │   ├── Var serie
│           │   │   ├── Expr
│           │   │   │   ├── Expr
│           │   │   │   │   ├── Expr
│           │   │   │   │   │   ├── Var a
│           │   │   │   │   │   ├── Operator *
│           │   │   │   │   │   ├── Var b
│           │   │   │   │   │   ├── Operator *
│           │   │   │   │   ├── Var c
│           │   │   │   │   ├── Operator +
│           │   │   │   │   └── IntConst 9
│           │   │   └── Semicolon ;

```

Case 3

```
1 Programa
2
3 principal
4 {
5
6
7     a = 4;
8
9     Real b = 5.8E-8;
10
11     Entero serie = a * * c;
12
13 }
14
15
```

```
graph LR
    L1[Programa] --> L2[Var c]
    L1 --> L3[Operator +]
    L1 --> L4[IntConst 9]
    L1 --> L5[Semicolon ;]

PS D:\uni 4\implementacion metodos computacionales\va_sintactico> ./analizadorS archivo3.txt
>>
Token      Tipo
-----
Programa   Programa
{          Llave que abre
principal  Llave que abre
{          Llave que abre
a          Variable
=          Assign
4          Entero
;          Punto y coma
Real       Real
b          Variable
=          Assign
5.8E-8     Real
;          Punto y coma
Entero     Entero
serie      Variable
=          Assign
a          Variable
*          Multiplicacion
*          Multiplicacion
c          Variable
;          Punto y coma
}          Llave que cierra
}          Llave que cierra

Syntax error:
(line 1, column 1):
unexpected Token {tokenType = Assign, tokenValue = "="}
PS D:\uni 4\implementacion metodos computacionales\va_sintactico>
```

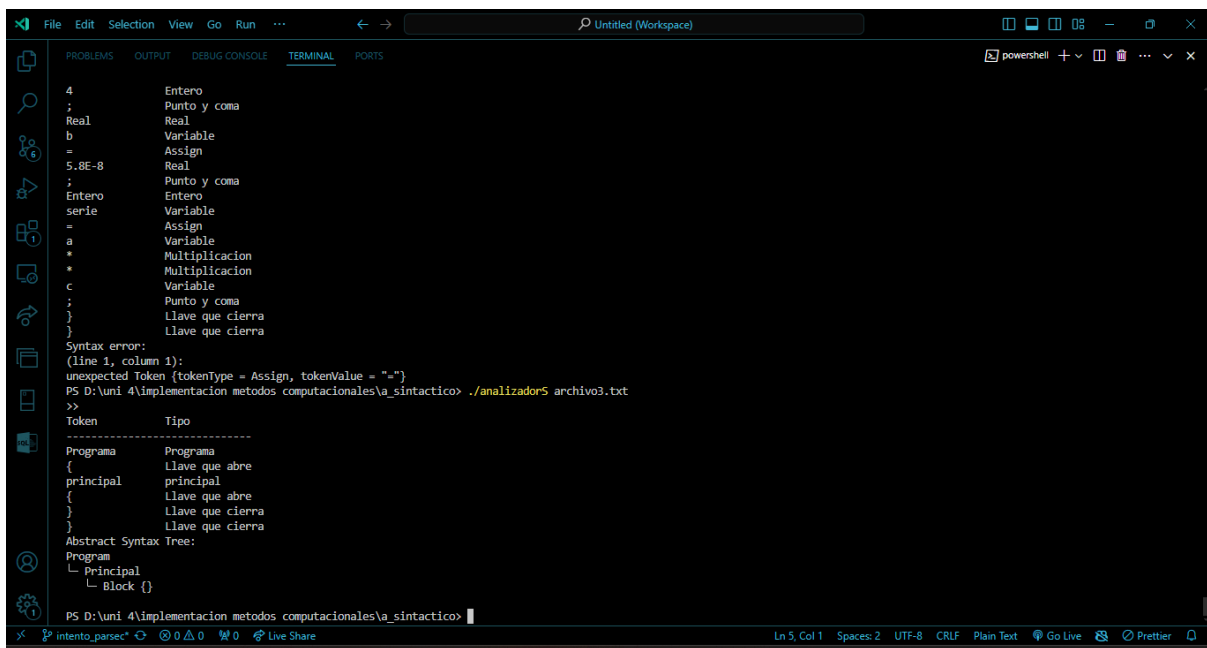
Case 4



Visual Studio Code editor interface. The file explorer on the left shows a project named 'a_sintactico' containing 'archivo2.txt', 'archivo3.txt', 'Grammar.txt', and 'AnalizadorSintactico.hs'. The editor window displays 'archivo3.txt' with the following code:

```
1 Programa {  
2     principal {  
3     }  
4 }  
5
```

The status bar at the bottom indicates 'Ln 5, Col 1', 'Spaces: 2', 'UTF-8', 'CRLF', 'Plain Text', and 'Go Live'.

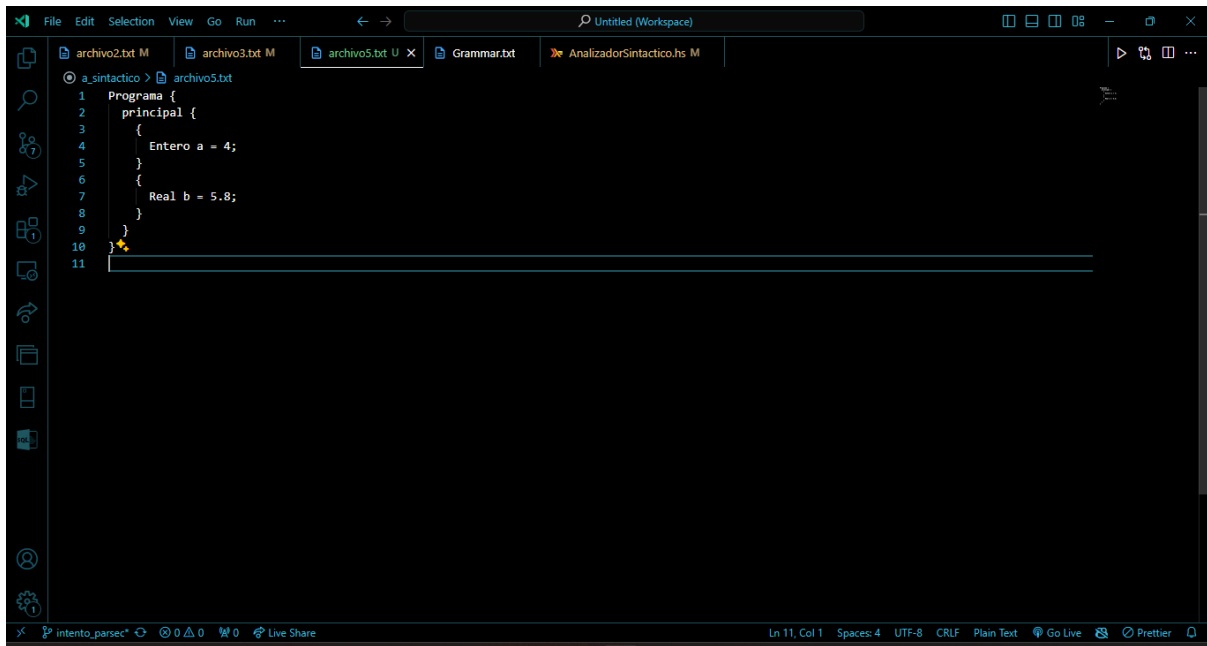


Visual Studio Code editor interface showing the output of a parser. The 'TERMINAL' tab is active, displaying the following output:

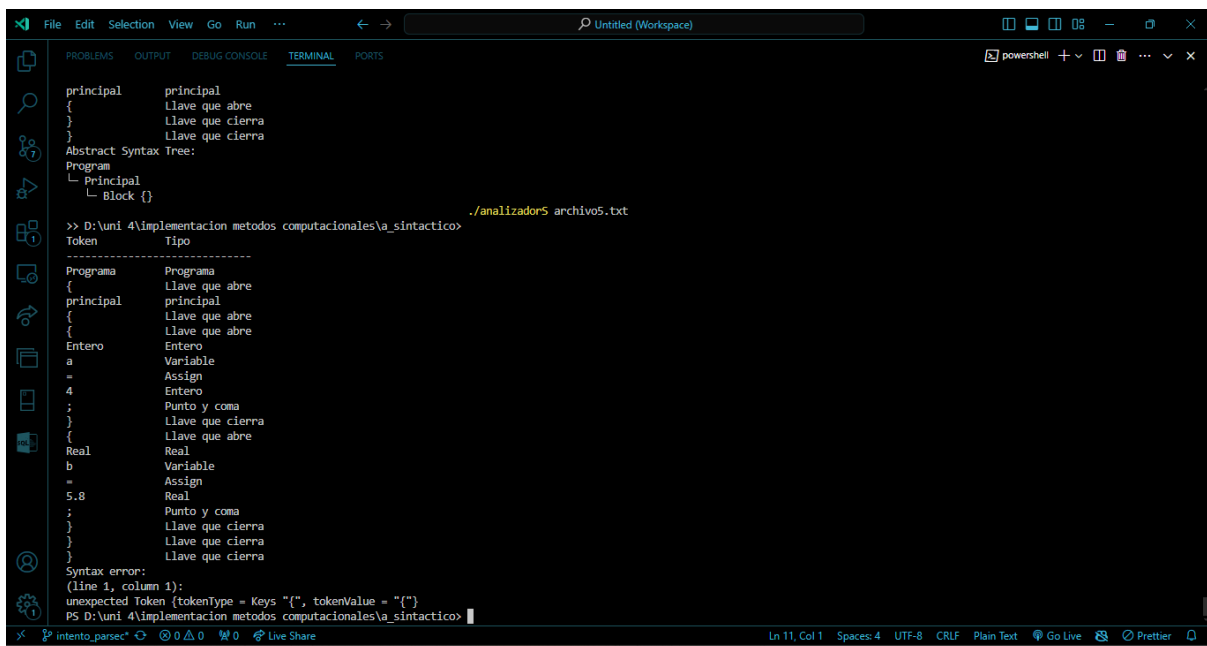
```
4      Entero  
;      Punto y coma  
Real   Real  
b      Variable  
=      Assign  
5.8E-8 Real  
;      Punto y coma  
Entero Entero  
serie  Variable  
=      Assign  
a      Variable  
*      Multiplicacion  
*      Multiplicacion  
c      Variable  
;      Punto y coma  
}      Llave que cierra  
}      Llave que cierra  
Syntax error:  
(line 1, column 1):  
unexpected Token {tokenType = Assign, tokenValue = "="}  
PS D:\uni 4\implementacion metodos computacionales\va_sintactico> ./analizador5 archivo3.txt  
>>  
Token      Tipo  
-----  
Programa   Programa  
{          Llave que abre  
principal  {  
{          Llave que abre  
}          Llave que cierra  
}          Llave que cierra  
Abstract Syntax Tree:  
Program  
├─ Principal  
└─ Block {}  
PS D:\uni 4\implementacion metodos computacionales\va_sintactico>
```

The status bar at the bottom indicates 'Ln 5, Col 1', 'Spaces: 2', 'UTF-8', 'CRLF', 'Plain Text', and 'Go Live'.

Case 5



```
1 Programa {
2   principal {
3   {
4     Entero a = 4;
5   }
6   {
7     Real b = 5.8;
8   }
9 }
10 }
11
```



```
principal      principal
{              {
  Llave que abre
}              }
  Llave que cierra
}              }
  Llave que cierra

Abstract Syntax Tree:
Program
├── Principal
│   └── Block {}
└──

>> D:\uni 4\implementacion metodos computacionales\la_sintactico> ./analizador5 archivo5.txt
Token
-----
Programa      Programa
{              {
  Llave que abre
principal     principal
{              {
  Llave que abre
Entero        Entero
a              Variable
=              Assign
4              Entero
;              Punto y coma
}              }
  Llave que cierra
{              {
Real          Real
b              Variable
=              Assign
5.8            Real
;              Punto y coma
}              }
  Llave que cierra
}              }
  Llave que cierra
}              }
  Llave que cierra

Syntax error:
(line 1, column 1):
unexpected Token {tokenType = Keys "(", tokenValue = "("}
PS D:\uni 4\implementacion metodos computacionales\la_sintactico>
```

Conclusiones

El desarrollo eficiente y efectivo del analizador como aplicación práctica de la teoría de lenguajes formales implicó considerar la estructura previa del tokenizador, por lo que se decidió construir los parsers de manera modular. Se hizo uso de la librería Parsec en el análisis de las cadenas de texto, la cual, al enfocarse en el análisis de cadenas, facilita la construcción de parsers que operan directamente sobre el texto fuente, proporciona

herramientas para manejar diferentes tipos de tokens y configurar cómo se deben reconocer, permitiendo flexibilidad en las declaraciones de elementos como palabras clave, operadores y delimitadores, centralizando y simplificando la gestión de tokens en el analizador sintáctico.

En conjunto, el analizador léxico y este proyecto proporcionan un desarrollo exhaustivo de nuestras bases de lenguajes formales. El uso de Parsec ha demostrado ser una herramienta poderosa y versátil que, combinada con un diseño modular y una buena estructuración del código, ha permitido crear un analizador sintáctico robusto y fácil de mantener, que puede ser adaptado y extendido según sea necesario.