



Informe Técnico

Proyecto Compiladores I

(Compilador Rust)

Número de Cuenta:

11811146

Nombre: Tiffanny Alexa

Varela Banegas

Catedrático: Román Arturo

Pineda Soto

Clase: Compiladores I

Sección: 2689

Fecha de Entrega: 28 de

septiembre de 2025

Contenido

1.	Introducción	1
2.	Contenido	2
2.1.	Descripción Del Proyecto Y Herramientas Utilizadas.....	2
2.1.1.	Descripción del Proyecto.....	2
2.1.2.	Herramientas Utilizadas	2
2.2.	Explicación De La Gramática Utilizada.....	4
2.2.1.	Lexer.....	4
2.2.2.	Parser	6
2.3.	Ejemplos De Entrada Y Salida Del Lexer Y Del Parser.	7
	Prueba2.rs.....	7
	Prueba4.rs.....	11
	Prueba2_error.rs	15
	Prueba4_error.rs	19
2.4.	Descripción De Las Pruebas Realizadas.....	23
	Prueba2.rs.....	23
	Prueba4.rs.....	25
	Prueba2_error.rs	26
	Prueba4_error.rs	29
3.	Conclusiones	32

1. Introducción

Este informe es la documentación para el proyecto final de la clase de Compiladores I, el cual es la creación de un compilador básico que realiza el análisis léxico y sintáctico de programas hechos en lenguaje RUST.

El objetivo del proyecto es aprender cómo funcionan los primeros pasos de un compilador, aprender que estructuras se usan en cada uno de estos pasos y su implementación dentro de los mismos.

Este trabajo se basa en hacer los dos primeros pasos que realiza un compilador:

1. Análisis léxico: divide el código fuente dado en “tokens”.
2. Análisis sintáctico: verifica que estos tokens sigan las reglas gramaticales del lenguaje.

Se aplicaron en la practica conceptos de clase como el uso de gramáticas libres de contexto, el uso del stack, entre otros. Al poder entender mejor estos conceptos ayudo a la implementación de detección de errores en el código y dar mensajes sobre estos que ayuden a su corrección.

2. Contenido

2.1. Descripción Del Proyecto Y Herramientas Utilizadas.

2.1.1. Descripción del Proyecto

Este proyecto consiste en la implementación de un analizador léxico y sintáctico para el lenguaje RUST. Su propósito principal es reconocer la estructura del código a analizar, identificación de tokens, construcción del árbol sintáctico y registro de errores encontrados dentro del análisis.

Este sigue las primeras fases clásicas de un compilador:

1. Lectura y procesamiento del código fuente
2. Análisis Léxico (Lexer): identificación de lexemas y generación de tokens.
3. Análisis Sintáctico (Parser): construcción del AST a partir de los tokens.
4. Registro de resultados y errores en archivos de salida

Se implemento la creación de archivos de tipo .txt para la salida de los tokens, AST y errores (si estos existen dentro del código) para poder llevar una bitácora de los mismos.

2.1.2. Herramientas Utilizadas

- VS Code: Editor principal configurado con extensiones *CMake Tools* y depuradores de C++ para la integración de escritura, compilación y ejecución en un mismo entorno.
- C++ 17: Se uso por su soporte en características actuales y la implementación de la librería *filesystem*.
- MSYS2 y MinGW64: proveen un entorno de compilación en Windows con soporte para c++ y *gdb*, lo que permite la compilación y depuración del proyecto.

- CMake: automatiza la compilación y la organización del proyecto, evitando problemas de configuración manual y ayudando a su portabilidad.
- Filesystem: librería estándar que se usa para crear, leer, escribir carpetas y archivos. Esto ayudo a poder mantener un orden por la creación de los archivos resultantes luego de cada análisis (lexer, parser y errores)
- Git: usado para el control de versiones para registrar cambios, trabajo con ramas y mantener un historial del proyecto.
- GitHub: repositorio remoto usado para respaldo del proyecto.

2.2. Explicación De La Gramática Utilizada

2.2.1. Lexer

- Uso de Gramática Libre de contexto: su uso se refleja en las funciones con el prefijo *parse* (parseFunc, parseSelf, parseFor, etc).
- Usa expresiones regulares para identificar los tokens (palabras reservadas, identificadores, ciclos, números).
- Devuelve el vector<Token> que es el que alimenta al parser.

Programa ::= (Funcion | Sentencia)*

Funcion ::= "fn" Identificador "(" ParamOpt ")" Bloque

ParamOpt ::= (Parametro ("," Parametro)*)?

Parametro ::= Identificador

Bloque ::= "{" Sentencia* "}"

Sentencia ::= Declaracion

| Asignacion

| If

| For

| While

| Loop

| Match

| Expresion ";;"

Declaracion ::= "let" Identificador ("=" Expresion)? ";"

Asignacion ::= Identificador "=" Expresion ";"

If ::= "if" Expresion Bloque ("else" Bloque)?

For ::= "for" Identificador "in" Expresion Bloque

While ::= "while" Expresion Bloque

Loop ::= "loop" Bloque

Match ::= "match" Expresion "{" (Patron "=>" Bloque)+ "}"

Expresion ::= Comparacion ((" & & " | " || ") Comparacion)*

Comparacion ::= Suma (("==" | "!=" | "<" | ">" | "<=" | ">=") Suma)*

Suma ::= Producto (("+" | "-") Producto)*

Producto ::= Primario (("*" | "/") Primario)*

Primario ::= Identificador

| Numero

| Cadena

| "true" | "false"

| "(" Expresion ")"

Identificador ::= [a-zA-Z_][a-zA-Z0-9_]*

Numero ::= [0-9]+

Cadena ::= "\" . * ? \"

- Programa: puede ser funciones o sentencias

- Funciones: definida con *fn*, parámetros opcionales y un bloque
- Bloques: delimitados por llaves ({...}), contiene secuencia de sentencias.
- Sentencias: incluye declaraciones (let), asignaciones, estructuras de control o expresiones
- Expresiones: precedencia de operadores
- Primarios: elementos básicos (identificadores, literales, expresiones agrupadas)

2.2.2. Parser

- Uso de árbol de sintaxis abstracta (AST): se implementa en la clase *NodoAST*. Cada función del parser construye los nodos (`make_unique<NodoAST>`) que representan los elementos del lenguaje. Estos nodos se conectan como hijos y así se refleja la jerarquía estructuras del programa.
 - Es de tipo **LL(k)** ya que cada entrada se consume de izquierda a derecha prediciendo que producción usar basándose en el *lookahead* (token actual).
1. Recorre la lista de tokens con un puntero.
 2. Cada función corresponde a un no terminal de la gramática.
 - a. `parseFunc()` → define funciones
 - b. `parself()` → condicionales
 - c. `parseExpre()` → expresiones aritméticas/lógicas
 3. Construye el AST mediante las instancias de *NodoAST*.
 4. Se representa el AST.
 5. Cada nodo contiene un tipo (Funcion, If, etc) y sus hijos.
 6. Se procesa y recorre la estructura semántica del programa.

2.3. Ejemplos De Entrada Y Salida Del Lexer Y Del Parser.

Entrada:

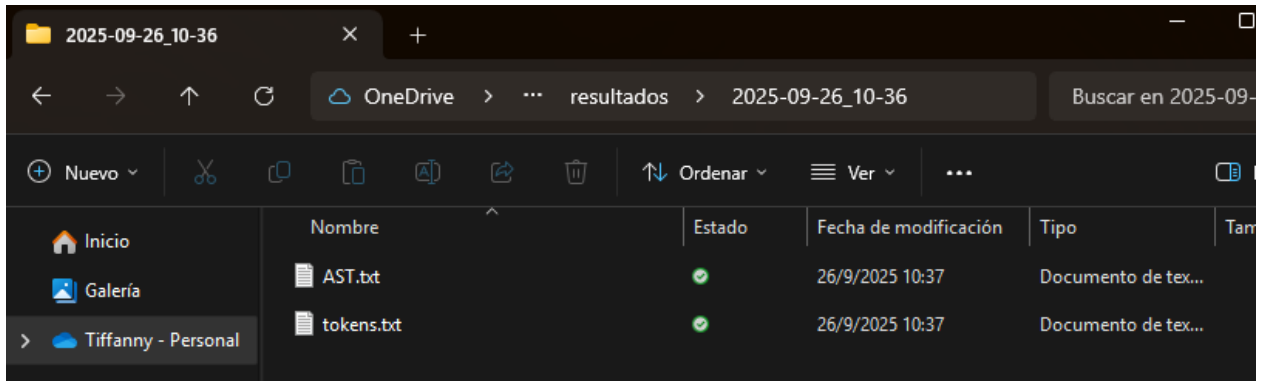
Prueba2.rs

//Expresion aritmetica con precedencia

```
fn prueba2() {  
    let r = 2 + 3 * 4;  
}  
  
fn main() {  
    prueba2();  
}
```

salida:

Nombre	Estado	Fecha de modificación	Tipo	Tamaño
 2025-09-26_10-34		26/9/2025 10:34	Carpeta de archivos	
 2025-09-26_10-36		26/9/2025 10:37	Carpeta de archivos	
 2025-09-26_10-37		26/9/2025 10:37	Carpeta de archivos	
 2025-09-26_10-38		26/9/2025 10:38	Carpeta de archivos	
 2025-09-26_10-40		26/9/2025 10:40	Carpeta de archivos	
 2025-09-26_10-57		26/9/2025 10:57	Carpeta de archivos	
 2025-09-26_10-59		26/9/2025 10:59	Carpeta de archivos	
 2025-09-26_11-00		26/9/2025 11:00	Carpeta de archivos	



Tokens.txt

=== TOKENS ENCONTRADOS ===

Total: 26

[L:2,C:1] PALABRA_RESERVADA: fn

[L:2,C:4] IDENTIFICADOR: prueba2

[L:2,C:11] PUNTUACION: (

[L:2,C:12] PUNTUACION:)

[L:2,C:14] PUNTUACION: {

[L:3,C:5] PALABRA_RESERVADA: let

[L:3,C:9] IDENTIFICADOR: r

[L:3,C:11] OPERADOR: =

[L:3,C:13] NUMERO_ENTERO: 2

[L:3,C:15] OPERADOR: +

[L:3,C:17] NUMERO_ENTERO: 3

[L:3,C:19] OPERADOR: *

[L:3,C:21] NUMERO_ENTERO: 4

[L:3,C:22] PUNTUACION: ;

[L:4,C:1] PUNTUACION: }

[L:6,C:1] PALABRA_RESERVADA: fn

[L:6,C:4] IDENTIFICADOR: main

[L:6,C:8] PUNTUACION: (

[L:6,C:9] PUNTUACION:)

[L:6,C:11] PUNTUACION: {

[L:7,C:5] IDENTIFICADOR: prueba2

[L:7,C:12] PUNTUACION: (

[L:7,C:13] PUNTUACION:)

[L:7,C:14] PUNTUACION: ;

[L:8,C:1] PUNTUACION: }

[L:9,C:1] FIN:

=== FIN TOKENS ===

AST.txt

=== AST ===

Programa

Funcion: prueba2

Parametros

Tipo Retorno: void

Bloque

Asignacion: r

Operador: +

Numero: 2

Operador: *

Numero: 3

Numero: 4

Funcion: main

Parametros

Tipo Retorno: void

Bloque

Llamada a Funcion: prueba2

=== FIN AST ===

Entrada:

Prueba4.rs

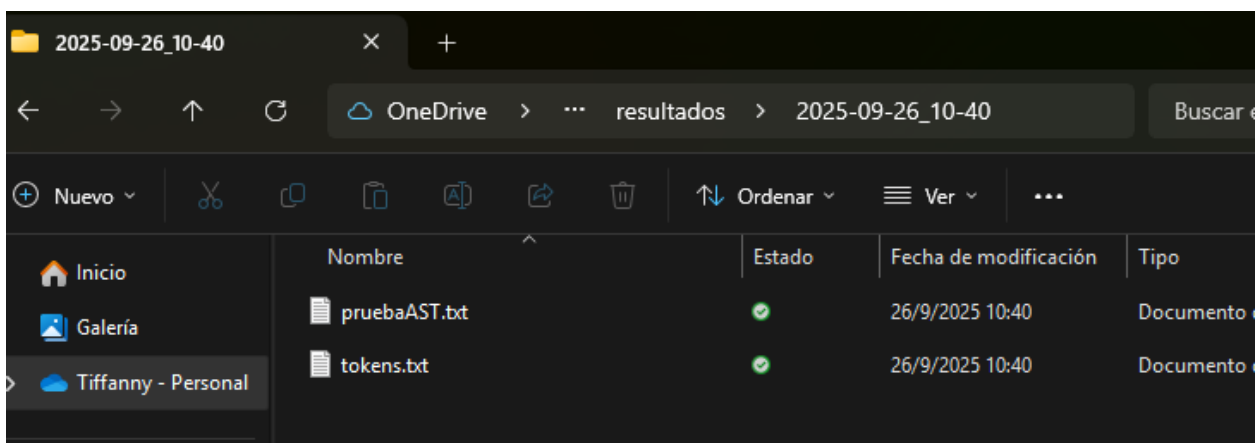
//Macro con varios argumentos

```
fn prueba4() {  
    println!("a", 10, 20 + 30);  
}
```

```
fn main() {  
    prueba4();  
}
```

Salida:

2025-09-26_10-34	✓	26/9/2025 10:34	Carpeta de archivos
2025-09-26_10-36	✓	26/9/2025 10:37	Carpeta de archivos
2025-09-26_10-38	✓	26/9/2025 10:38	Carpeta de archivos
2025-09-26_10-40	✓	26/9/2025 10:40	Carpeta de archivos
2025-09-26_10-57	✓	26/9/2025 10:57	Carpeta de archivos
2025-09-26_10-59	✓	26/9/2025 10:59	Carpeta de archivos
2025-09-26_11-00	✓	26/9/2025 11:00	Carpeta de archivos



Tokens.txt

=== TOKENS ENCONTRADOS ===

Total: 29

[L:2,C:1] PALABRA_RESERVADA: fn

[L:2,C:4] IDENTIFICADOR: prueba4

[L:2,C:11] PUNTUACION: (

[L:2,C:12] PUNTUACION:)

[L:2,C:14] PUNTUACION: {

[L:3,C:5] IDENTIFICADOR: println

[L:3,C:12] SIMBOLO: !

[L:3,C:13] PUNTUACION: (

[L:3,C:14] CADENA: "a"

[L:3,C:17] PUNTUACION: ,

[L:3,C:19] NUMERO_ENTERO: 10

[L:3,C:21] PUNTUACION: ,

[L:3,C:23] NUMERO_ENTERO: 20

[L:3,C:26] OPERADOR: +

[L:3,C:28] NUMERO_ENTERO: 30

[L:3,C:30] PUNTUACION:)

[L:3,C:31] PUNTUACION: ;
[L:4,C:1] PUNTUACION: }
[L:6,C:1] PALABRA_RESERVADA: fn
[L:6,C:4] IDENTIFICADOR: main
[L:6,C:8] PUNTUACION: (
[L:6,C:9] PUNTUACION:)
[L:6,C:11] PUNTUACION: {
[L:7,C:5] IDENTIFICADOR: prueba4
[L:7,C:12] PUNTUACION: (
[L:7,C:13] PUNTUACION:)
[L:7,C:14] PUNTUACION: ;
[L:8,C:1] PUNTUACION: }
[L:9,C:1] FIN:

=== FIN TOKENS ===

PruebaAST.txt

=== AST ===

Programa

Funcion: prueba4

Parametros

Tipo Retorno: void

Bloque

Llamada a Macro: println!

Cadena de Caracteres: "a"

Numero: 10

Operador: +

Numero: 20

Numero: 30

Funcion: main

Parametros

Tipo Retorno: void

Bloque

Llamada a Funcion: prueba4

=== FIN AST ===

Entrada:















Prueba2_error.rs

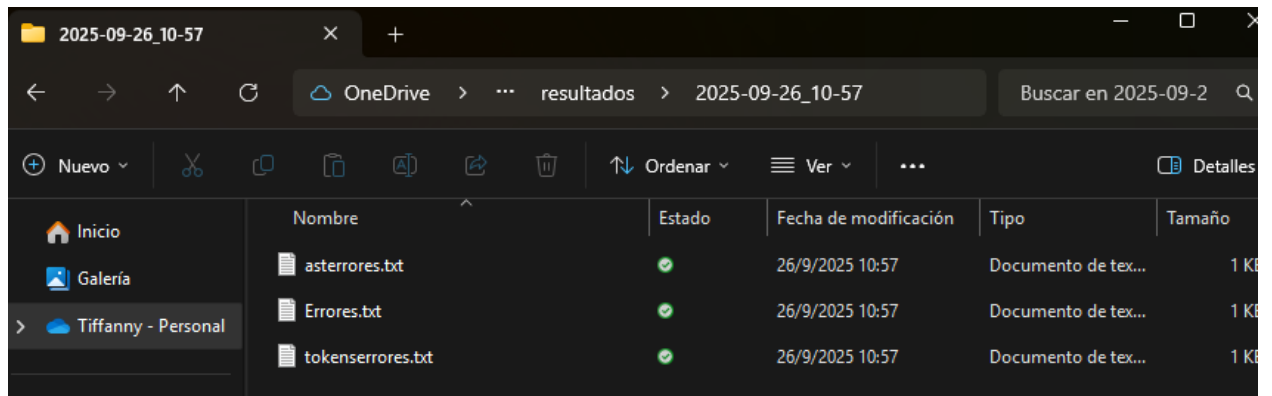
//Expresion aritmetica con caracter invalido

```
fn prueba2_error() {  
    let r = 2 + 3 * 4 ~;  
}
```

```
fn main() {  
    prueba2_error();  
}
```

Salida:

 2025-09-26_10-34		26/9/2025 10:34	Carpeta de archivos
 2025-09-26_10-36		26/9/2025 10:37	Carpeta de archivos
 2025-09-26_10-38		26/9/2025 10:38	Carpeta de archivos
 2025-09-26_10-40		26/9/2025 10:40	Carpeta de archivos
 2025-09-26_10-57		26/9/2025 12:24	Carpeta de archivos
 2025-09-26_10-59		26/9/2025 10:59	Carpeta de archivos
 2025-09-26_11-00		26/9/2025 11:00	Carpeta de archivos



Tokenserrores.txt

=== TOKENS ENCONTRADOS ===

Total: 27

[L:2,C:1] PALABRA_RESERVADA: fn

[L:2,C:4] IDENTIFICADOR: prueba2_error

[L:2,C:17] PUNTUACION: (

[L:2,C:18] PUNTUACION:)

[L:2,C:20] PUNTUACION: {

[L:3,C:5] PALABRA_RESERVADA: let

[L:3,C:9] IDENTIFICADOR: r

[L:3,C:11] OPERADOR: =

[L:3,C:13] NUMERO_ENTERO: 2

[L:3,C:15] OPERADOR: +

[L:3,C:17] NUMERO_ENTERO: 3

[L:3,C:19] OPERADOR: *

[L:3,C:21] NUMERO_ENTERO: 4

[L:3,C:23] ERROR: ~

[L:3,C:24] PUNTUACION: ;

[L:4,C:1] PUNTUACION: }

[L:6,C:1] PALABRA_RESERVADA: fn

[L:6,C:4] IDENTIFICADOR: main

[L:6,C:8] PUNTUACION: (

[L:6,C:9] PUNTUACION:)

[L:6,C:11] PUNTUACION: {

[L:7,C:5] IDENTIFICADOR: prueba2_error

[L:7,C:18] PUNTUACION: (

[L:7,C:19] PUNTUACION:)

[L:7,C:20] PUNTUACION: ;

[L:8,C:1] PUNTUACION: }

[L:9,C:1] FIN:

=== FIN TOKENS ===

Asterroses.txt

=== AST ===

Programa

Funcion: prueba2_error

Parametros

Tipo Retorno: void

Bloque

Funcion: main

Parametros

Tipo Retorno: void

Bloque

Llamada a Funcion: prueba2_error

=== FIN AST ===

Errores.txt

=== El AST esta incompleto ya que se han encontrado errores dentro del codigo ===

[L:3,C:23] ERROR: LEXICO: Simbolo invalido en L3, C23: ~

[L:3,C:23] ERROR: SINTACTICO: Error de sintaxis en el bloque: Linea: 3, Columna: 23.

Token esperado ';', pero se encontro '~'. Esperando ';' despues de la asignacion.

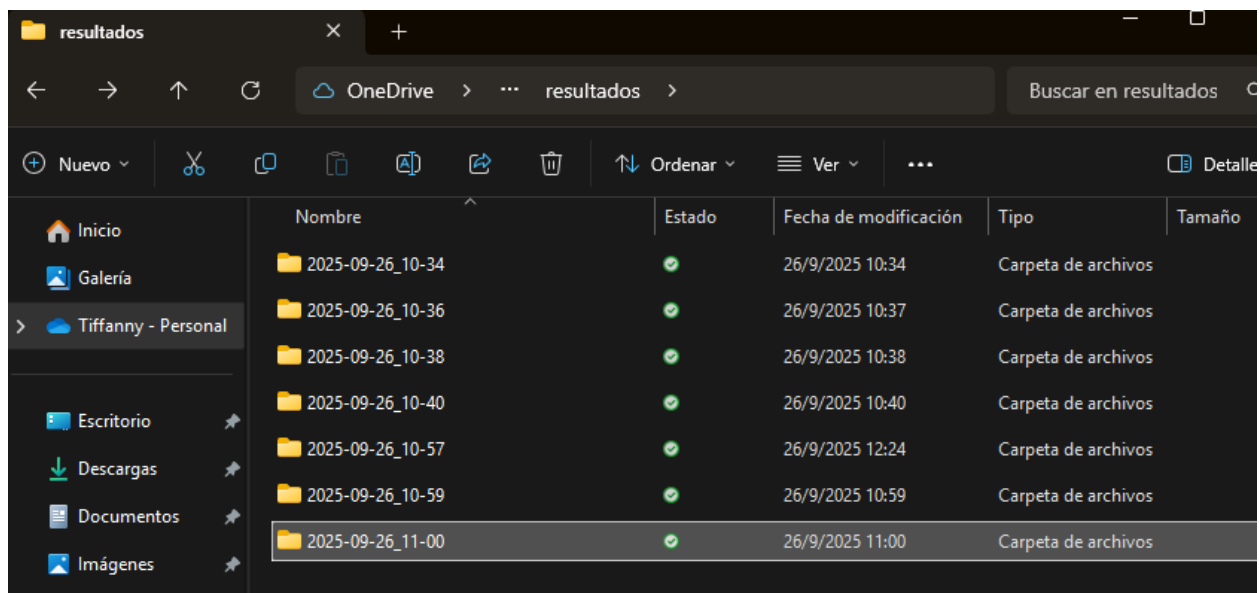
Entrada:

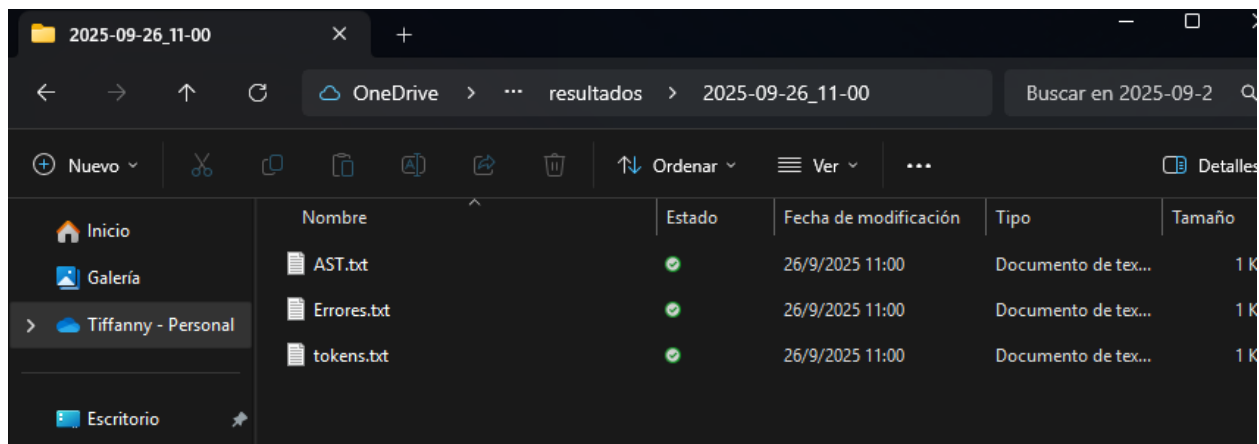
Prueba4_error.rs

//Macro con argumentos incompletos

```
fn prueba4_error() {  
    println!("a", 10, );  
}  
  
fn main() {  
    prueba4_error();  
}
```

Salida:





Tokens.txt

=== TOKENS ENCONTRADOS ===

Total: 26

[L:2,C:1] PALABRA_RESERVADA: fn

[L:2,C:4] IDENTIFICADOR: prueba4_error

[L:2,C:17] PUNTUACION: (

[L:2,C:18] PUNTUACION:)

[L:2,C:20] PUNTUACION: {

[L:3,C:5] IDENTIFICADOR: println

[L:3,C:12] SIMBOLO: !

[L:3,C:13] PUNTUACION: (

[L:3,C:14] CADENA: "a"

[L:3,C:17] PUNTUACION: ,

[L:3,C:19] NUMERO_ENTERO: 10

[L:3,C:21] PUNTUACION: ,

[L:3,C:23] PUNTUACION:)

[L:3,C:24] PUNTUACION: ;

[L:4,C:1] PUNTUACION: }

[L:6,C:1] PALABRA_RESERVADA: fn

[L:6,C:4] IDENTIFICADOR: main

[L:6,C:8] PUNTUACION: (

[L:6,C:9] PUNTUACION:)

[L:6,C:11] PUNTUACION: {

[L:7,C:5] IDENTIFICADOR: prueba4_error

[L:7,C:18] PUNTUACION: (

[L:7,C:19] PUNTUACION:)

[L:7,C:20] PUNTUACION: ;

[L:8,C:1] PUNTUACION: }

[L:9,C:1] FIN:

=== FIN TOKENS ===

AST.txt

=== AST ===

Programa

Funcion: prueba4_error

Parametros

Tipo Retorno: void

Bloque

Funcion: main

Parametros

Tipo Retorno: void

Bloque

Llamada a Funcion: prueba4_error

=== FIN AST ===

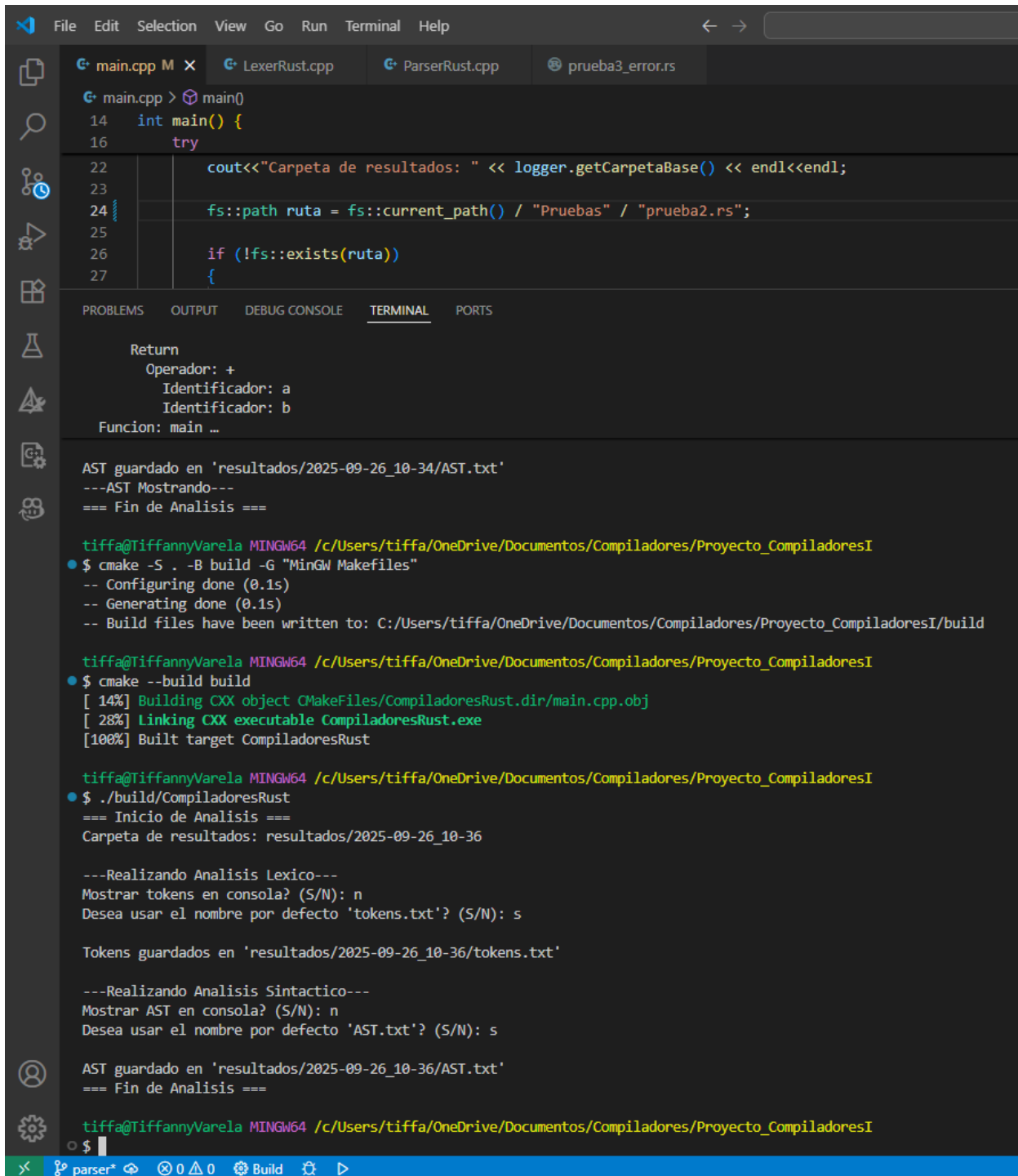
Errores.txt

=== El AST esta incompleto ya que se han encontrado errores dentro del codigo ===

[L:3,C:23] ERROR: SINTACTICO: Error de sintaxis en el bloque: Token inesperando en la expresion:) en la linea 3, columna 23

2.4. Descripción De Las Pruebas Realizadas.

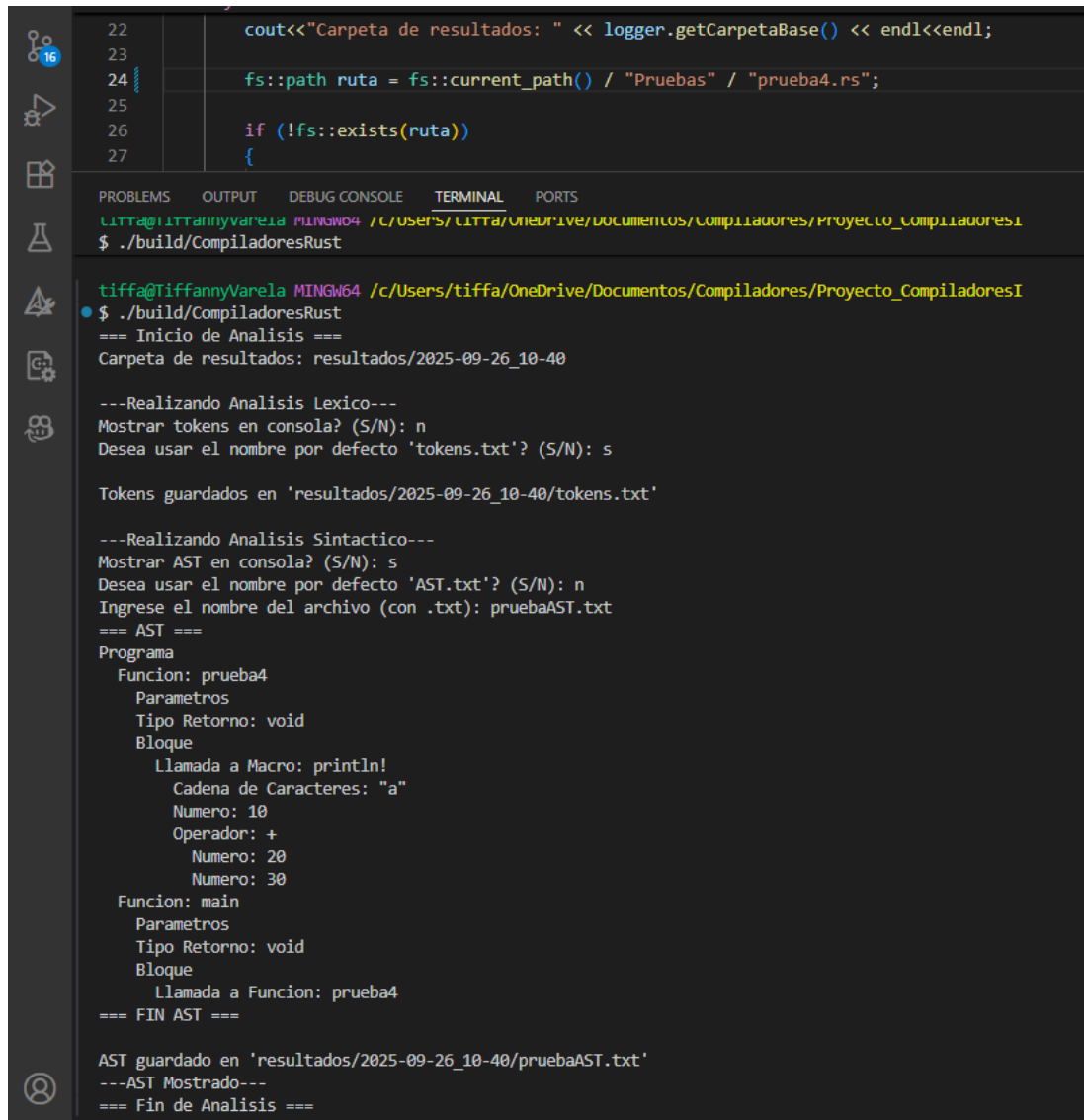
Prueba2.rs



```
File Edit Selection View Go Run Terminal Help
main.cpp M X LexerRust.cpp ParserRust.cpp prueba3_error.rs
main.cpp > main()
14 int main() {
16     try
22         cout<<"Carpeta de resultados: " << logger.getCarpetaBase() << endl<<endl;
23
24         fs::path ruta = fs::current_path() / "Pruebas" / "prueba2.rs";
25
26         if (!fs::exists(ruta))
27         {
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Return
Operador: +
Identificador: a
Identificador: b
Funcion: main ...
AST guardado en 'resultados/2025-09-26_10-34/AST.txt'
---AST Mostrando---
=== Fin de Analisis ===
tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$ cmake -S . -B build -G "MinGW Makefiles"
-- Configuring done (0.1s)
-- Generating done (0.1s)
-- Build files have been written to: C:/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI/build
tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$ cmake --build build
[ 14%] Building CXX object CMakeFiles/CompiladoresRust.dir/main.cpp.obj
[ 28%] Linking CXX executable CompiladoresRust.exe
[100%] Built target CompiladoresRust
tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$ ./build/CompiladoresRust
=== Inicio de Analisis ===
Carpeta de resultados: resultados/2025-09-26_10-36
---Realizando Analisis Lexico---
Mostrar tokens en consola? (S/N): n
Desea usar el nombre por defecto 'tokens.txt'? (S/N): s
Tokens guardados en 'resultados/2025-09-26_10-36/tokens.txt'
---Realizando Analisis Sintactico---
Mostrar AST en consola? (S/N): n
Desea usar el nombre por defecto 'AST.txt'? (S/N): s
AST guardado en 'resultados/2025-09-26_10-36/AST.txt'
=== Fin de Analisis ===
tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$
```

1. Se cambia el path para utilizar el código de prueba2.rs
2. Se ejecuta el código con los comandos de CMake y MinGW64
 - a. `rm -rf build`
 - b. `cmake -S . -B build -G "MinGW Makefiles"`
 - c. `cmake --build build`
 - d. `./build/CompiladoresRust`
3. Escribimos "n" ya que no quiero mostrar los tokens en consola
4. Escribimos "s" para guardar el documento de los tokens con el nombre genérico
5. Escribimos "n" ya que no quiero el AST en consola
6. Escribimos "s" para guardar el documento del AST con el nombre genérico

Prueba4.rs



```
22     cout<<"Carpeta de resultados: " << logger.getCarpetaBase() << endl<<endl;
23
24     fs::path ruta = fs::current_path() / "Pruebas" / "prueba4.rs";
25
26     if (!fs::exists(ruta))
27     {
28
29     }
30 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$ ./build/CompiladoresRust

tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$ ./build/CompiladoresRust
=== Inicio de Analisis ===
Carpeta de resultados: resultados/2025-09-26_10-40

---Realizando Analisis Lexico---
Mostrar tokens en consola? (S/N): n
Desea usar el nombre por defecto 'tokens.txt'? (S/N): s

Tokens guardados en 'resultados/2025-09-26_10-40/tokens.txt'

---Realizando Analisis Sintactico---
Mostrar AST en consola? (S/N): s
Desea usar el nombre por defecto 'AST.txt'? (S/N): n
Ingrese el nombre del archivo (con .txt): pruebaAST.txt
=== AST ===
Programa
  Funcion: prueba4
  Parametros
  Tipo Retorno: void
  Bloque
    Llamada a Macro: println!
    Cadena de Caracteres: "a"
    Numero: 10
    Operador: +
    Numero: 20
    Numero: 30
  Funcion: main
  Parametros
  Tipo Retorno: void
  Bloque
    Llamada a Funcion: prueba4
=== FIN AST ===

AST guardado en 'resultados/2025-09-26_10-40/pruebaAST.txt'
---AST Mostrado---
=== Fin de Analisis ===
```

1. Se cambia el path para utilizar el código de prueba4.rs
2. Se ejecuta el código con los comandos de CMake y MinGW64
 - a. `rm -rf build`
 - b. `cmake -S . -B build -G "MinGW Makefiles"`
 - c. `cmake --build build`
 - d. `./build/CompiladoresRust`
3. Escribimos "n" ya que no quiero mostrar los tokens en consola

4. Escribimos “s” para guardar el documento de los tokens con el nombre genérico
5. Escribimos “s” ya que quiero el AST en consola
6. Escribimos “n” para guardar el documento del AST con un nombre específico (en este caso “pruebaAST.txt”)

Prueba2_error.rs

```

main.cpp M x prueba2_error.rs
main.cpp > main()
14 int main() {
16     try
21         cout<< "=== Inicio de Analisis ===" << endl;
22         cout<<"Carpeta de resultados: " << logger.getCarpetaBase() << endl<<endl;
23
24         fs::path ruta = fs::current_path() / "Pruebas" / "prueba2_error.rs";
25
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

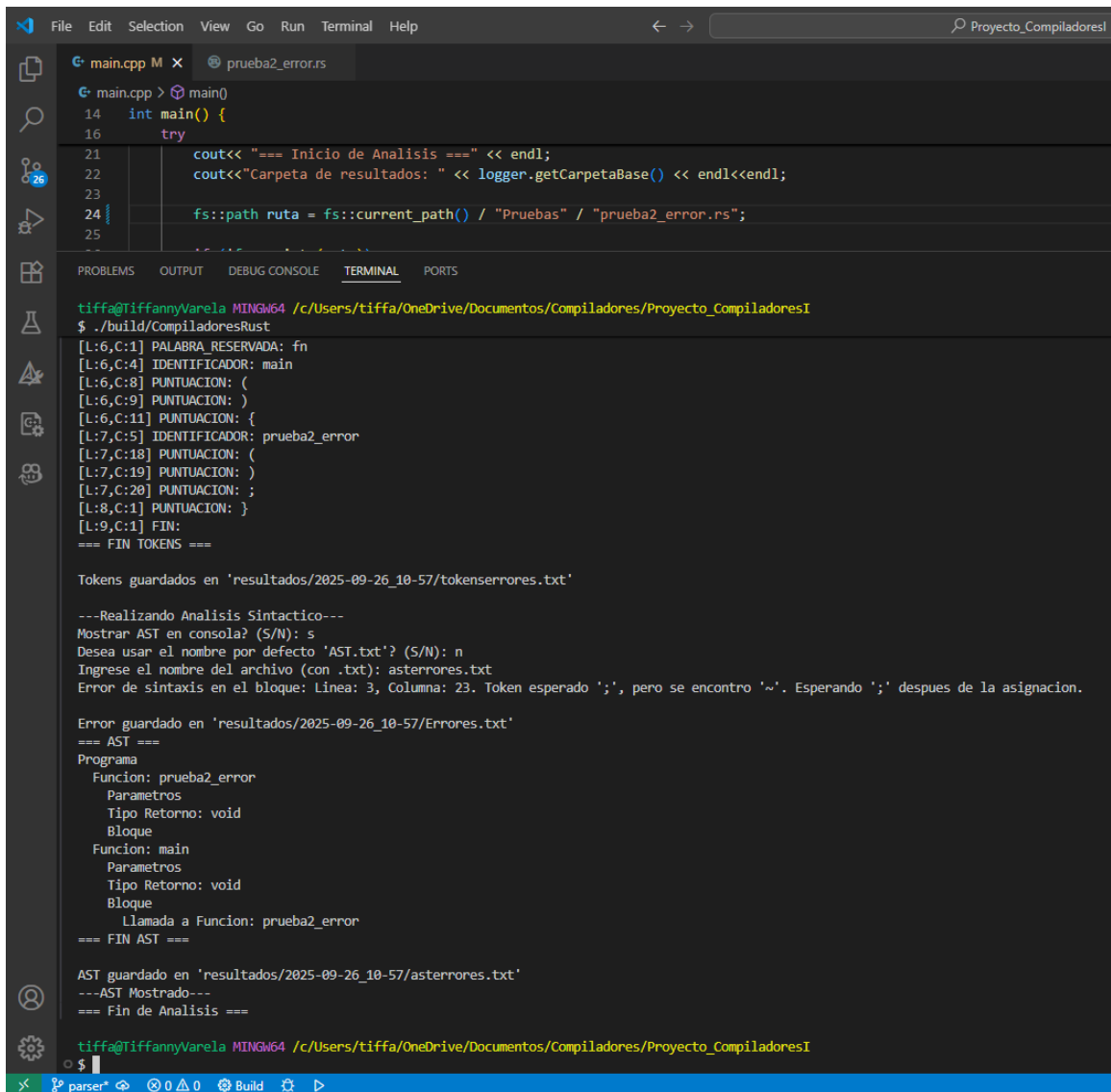
tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$ cmake --build build

tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$ ./build/CompiladoresRust
=== Inicio de Analisis ===
Carpeta de resultados: resultados/2025-09-26_10-57

---Realizando Analisis Lexico---
Mostrar tokens en consola? (S/N): s
Desea usar el nombre por defecto 'tokens.txt'? (S/N): n
Ingrese el nombre del archivo (con .txt): tokenserrores.txt
LEXICO: Símbolo invalido en L3, C23: ~

Error guardado en 'resultados/2025-09-26_10-57/Errores.txt'
=== TOKENS ENCONTRADOS ===
Total: 27
[L:2,C:1] PALABRA_RESERVADA: fn
[L:2,C:4] IDENTIFICADOR: prueba2_error
[L:2,C:17] PUNTUACION: (
[L:2,C:18] PUNTUACION: )
[L:2,C:20] PUNTUACION: {
[L:3,C:5] PALABRA_RESERVADA: let
[L:3,C:9] IDENTIFICADOR: r
[L:3,C:11] OPERADOR: =
[L:3,C:13] NUMERO_ENTERO: 2
[L:3,C:15] OPERADOR: +
[L:3,C:17] NUMERO_ENTERO: 3
[L:3,C:19] OPERADOR: *
[L:3,C:21] NUMERO_ENTERO: 4
[L:3,C:23] ERROR: ~
[L:3,C:24] PUNTUACION: ;
[L:4,C:1] PUNTUACION: }
[L:6,C:1] PALABRA_RESERVADA: fn
[L:6,C:4] IDENTIFICADOR: main
[L:6,C:8] PUNTUACION: (
[L:6,C:9] PUNTUACION: )
[L:6,C:11] PUNTUACION: {
[L:7,C:5] IDENTIFICADOR: prueba2_error
[L:7,C:18] PUNTUACION: (
[L:7,C:19] PUNTUACION: )
[L:7,C:20] PUNTUACION: ;
[L:8,C:1] PUNTUACION: }

```



```
File Edit Selection View Go Run Terminal Help
main.cpp M x prueba2_error.rs
main.cpp > main()
14 int main() {
16     try
21         cout<< "=== Inicio de Analisis ===" << endl;
22         cout<<"Carpeta de resultados: " << logger.getCarpetaBase() << endl<<endl;
23
24         fs::path ruta = fs::current_path() / "Pruebas" / "prueba2_error.rs";
25
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$ ./build/CompiladoresRust
[L:6,C:1] PALABRA_RESERVADA: fn
[L:6,C:4] IDENTIFICADOR: main
[L:6,C:8] PUNTUACION: (
[L:6,C:9] PUNTUACION: )
[L:6,C:11] PUNTUACION: {
[L:7,C:5] IDENTIFICADOR: prueba2_error
[L:7,C:18] PUNTUACION: (
[L:7,C:19] PUNTUACION: )
[L:7,C:20] PUNTUACION: ;
[L:8,C:1] PUNTUACION: }
[L:9,C:1] FIN:
=== FIN TOKENS ===

Tokens guardados en 'resultados/2025-09-26_10-57/tokenserrores.txt'

---Realizando Analisis Sintactico---
Mostrar AST en consola? (S/N): s
Desea usar el nombre por defecto 'AST.txt'? (S/N): n
Ingrese el nombre del archivo (con .txt): asterrores.txt
Error de sintaxis en el bloque: Linea: 3, Columna: 23. Token esperado ';', pero se encontro '~'. Esperando ';' despues de la asignacion.

Error guardado en 'resultados/2025-09-26_10-57/Errores.txt'
=== AST ===
Programa
  Funcion: prueba2_error
    Parametros
    Tipo Retorno: void
    Bloque
  Funcion: main
    Parametros
    Tipo Retorno: void
    Bloque
      Llamada a Funcion: prueba2_error
=== FIN AST ===

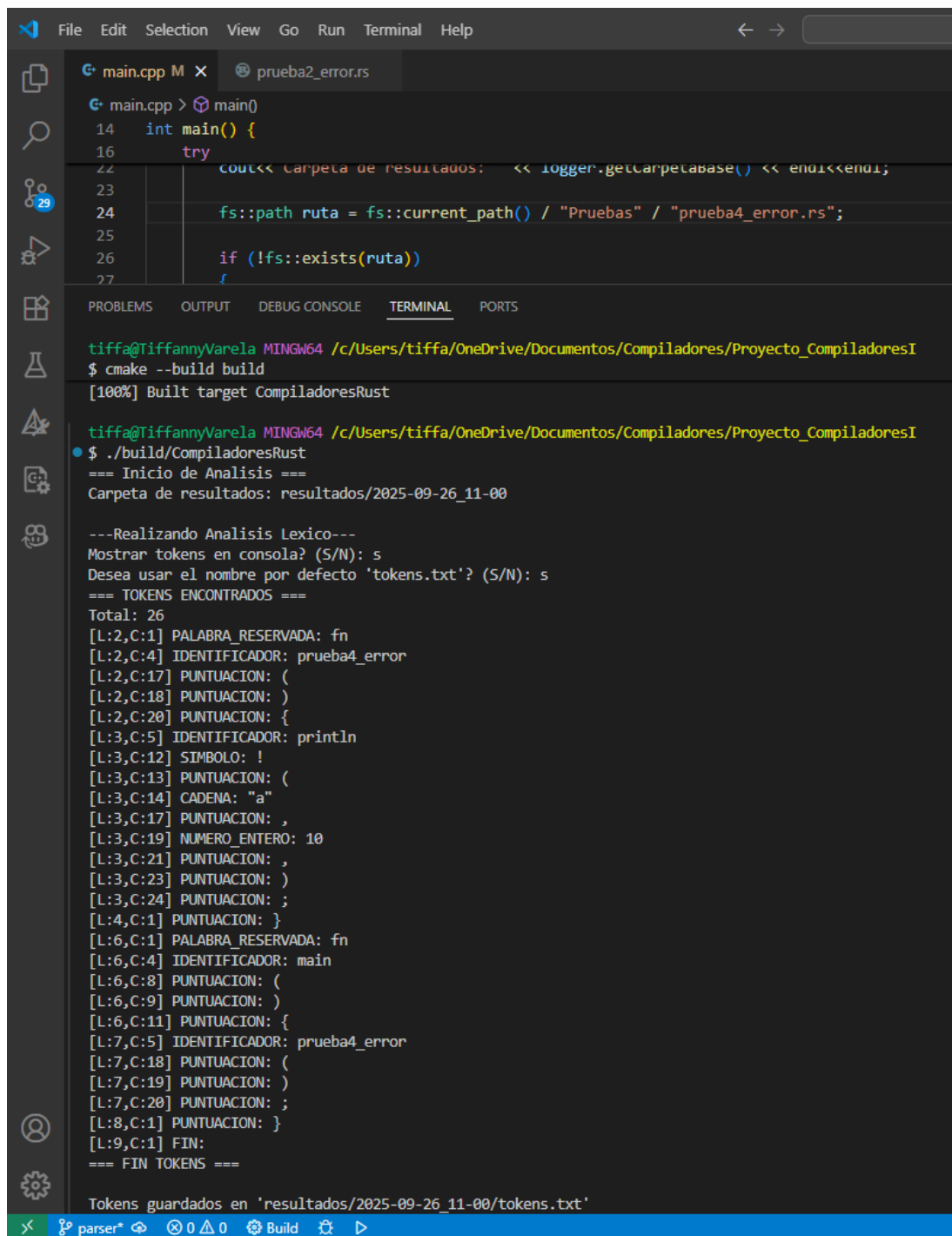
AST guardado en 'resultados/2025-09-26_10-57/asterrores.txt'
---AST Mostrado---
=== Fin de Analisis ===

tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$
```

1. Se cambia el path para utilizar el código de prueba2_error.rs
2. Se ejecuta el código con los comandos de CMake y MinGW64
 - a. `rm -rf build`
 - b. `cmake -S . -B build -G "MinGW Makefiles"`
 - c. `cmake --build build`
 - d. `./build/CompiladoresRust`
3. Escribimos “s” ya que quiero mostrar los tokens en consola

4. Escribimos “n” para guardar el documento de los tokens con un nombre específico (“tokenserrores.txt”)
5. Escribimos “s” ya que quiero el AST en consola
6. Escribimos “n” para guardar el documento del AST con un nombre específico (“asteerrores.txt”)

Prueba4_error.rs



The screenshot shows an IDE with two tabs: `main.cpp` and `prueba2_error.rs`. The `main.cpp` tab is active, showing a C++ program that uses `cout` to print a directory path and checks if a file exists. The terminal window at the bottom shows the execution of a Rust program that performs a lexical analysis of the `prueba4_error.rs` file. The output includes the directory path, the start of the analysis, and a list of tokens found in the file, such as `fn`, `prueba4_error`, `{`, `}`, `println`, `!`, `(`, `"a"`, `,`, `10`, `main`, and `FIN`.

```
main.cpp > main()
14 int main() {
16     try
22         cout<< "Carpeta de resultados: " << logger.getCarpetabase() << endl<<endl;
23
24         fs::path ruta = fs::current_path() / "Pruebas" / "prueba4_error.rs";
25
26         if (!fs::exists(ruta))
27     {
28
29     }
30 }
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$ cmake --build build
[100%] Built target CompiladoresRust

tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$ ./build/CompiladoresRust
=== Inicio de Analisis ===
Carpeta de resultados: resultados/2025-09-26_11-00

---Realizando Analisis Lexico---
Mostrar tokens en consola? (S/N): s
Desea usar el nombre por defecto 'tokens.txt'? (S/N): s
=== TOKENS ENCONTRADOS ===
Total: 26
[L:2,C:1] PALABRA_RESERVADA: fn
[L:2,C:4] IDENTIFICADOR: prueba4_error
[L:2,C:17] PUNTUACION: (
[L:2,C:18] PUNTUACION: )
[L:2,C:20] PUNTUACION: {
[L:3,C:5] IDENTIFICADOR: println
[L:3,C:12] SIMBOLO: !
[L:3,C:13] PUNTUACION: (
[L:3,C:14] CADENA: "a"
[L:3,C:17] PUNTUACION: ,
[L:3,C:19] NUMERO_ENTERO: 10
[L:3,C:21] PUNTUACION: ,
[L:3,C:23] PUNTUACION: )
[L:3,C:24] PUNTUACION: ;
[L:4,C:1] PUNTUACION: }
[L:6,C:1] PALABRA_RESERVADA: fn
[L:6,C:4] IDENTIFICADOR: main
[L:6,C:8] PUNTUACION: (
[L:6,C:9] PUNTUACION: )
[L:6,C:11] PUNTUACION: {
[L:7,C:5] IDENTIFICADOR: prueba4_error
[L:7,C:18] PUNTUACION: (
[L:7,C:19] PUNTUACION: )
[L:7,C:20] PUNTUACION: ;
[L:8,C:1] PUNTUACION: }
[L:9,C:1] FIN:
=== FIN TOKENS ===

Tokens guardados en 'resultados/2025-09-26_11-00/tokens.txt'
```

The screenshot shows the Visual Studio Code interface. The editor has two tabs: `main.cpp` and `prueba2_error.rs`. The `main.cpp` file contains the following code:

```
14 int main() {
15     try
16     {
17         cout<< "Carpetas de resultados: " << logger.getCarpetabase() << endl<<endl;
18     }
19     catch (...)
20     {
21     }
22     fs::path ruta = fs::current_path() / "Pruebas" / "prueba4_error.rs";
23     if (!fs::exists(ruta))
24     {
25     }
26 }
```

The `prueba2_error.rs` file is currently selected and shows the following code:

```
14 int main() {
15     try
16     {
17         cout<< "Carpetas de resultados: " << logger.getCarpetabase() << endl<<endl;
18     }
19     catch (...)
20     {
21     }
22     fs::path ruta = fs::current_path() / "Pruebas" / "prueba4_error.rs";
23     if (!fs::exists(ruta))
24     {
25     }
26 }
```

The `TERMINAL` tab is active, showing the output of the compilation process. The output includes the following text:

```
tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$ ./build/CompiladoresRust

[L:4,C:1] PUNTUACION: }
[L:6,C:1] PALABRA_RESERVADA: fn
[L:6,C:4] IDENTIFICADOR: main
[L:6,C:8] PUNTUACION: (
[L:6,C:9] PUNTUACION: )
[L:6,C:11] PUNTUACION: {
[L:7,C:5] IDENTIFICADOR: prueba4_error
[L:7,C:18] PUNTUACION: (
[L:7,C:19] PUNTUACION: )
[L:7,C:20] PUNTUACION: ;
[L:8,C:1] PUNTUACION: }
[L:9,C:1] FIN:
=== FIN TOKENS ===

Tokens guardados en 'resultados/2025-09-26_11-00/tokens.txt'

---Realizando Analisis Sintactico---
Mostrar AST en consola? (S/N): s
Desea usar el nombre por defecto 'AST.txt'? (S/N): s
Error de sintaxis en el bloque: Token inesperando en la expresion: ) en la linea 3, columna 23

Error guardado en 'resultados/2025-09-26_11-00/Errores.txt'
=== AST ===
Programa
  Funcion: prueba4_error
    Parametros
    Tipo Retorno: void
    Bloque
  Funcion: main
    Parametros
    Tipo Retorno: void
    Bloque
      Llamada a Funcion: prueba4_error
=== FIN AST ===

AST guardado en 'resultados/2025-09-26_11-00/AST.txt'
---AST Mostrado---
=== Fin de Analisis ===

tiffa@TiffanyVarela MINGW64 /c/Users/tiffa/OneDrive/Documentos/Compiladores/Proyecto_CompiladoresI
$
```

1. Se cambia el path para utilizar el código de prueba4_error.rs
2. Se ejecuta el código con los comandos de CMake y MinGW64
 - a. `rm -rf build`
 - b. `cmake -S . -B build -G "MinGW Makefiles"`

- c. `cmake --build build`
 - d. `./build/CompiladoresRust`
3. Escribimos “s” ya que quiero mostrar los tokens en consola
 4. Escribimos “s” para guardar el documento de los tokens con el nombre genérico
 5. Escribimos “s” ya que quiero el AST en consola
 6. Escribimos “s” para guardar el documento del AST con el nombre genérico

3. Conclusiones

- Se permitió comprender de forma practica como funcionan las fases iniciales de un compilador
- Se demostró que el uso de gramáticas libres de contexto y la construcción de un AST son herramientas fundamentales para la representación y procesamiento de la estructura de un lenguaje de programación
- Se experimento con herramientas para automatización de procesos como CMake
- Se logro conectar la teoría vista en clase con una implementación funcional