



INSTITUTO TECNOLÓGICO DE COSTA RICA

Proyecto #2: Parser

Escuela de Ingeniería en Computación
Compiladores e Intérpretes IC-5701

Alonso Navarro Carrillo, c. 2022236435

Carlos Venegas Masis, c. 2022153870

Valeria Gómez Acuña, c. 2022173229

Ing. Ericka Marín Schumann
II Semestre 2024

Tabla de contenidos

Introducción	2
Estrategia de solución	2
Análisis de resultados	4
Lecciones aprendidas	4
Casos de prueba	6
Casos de prueba 1: Errores en Expresiones Aritméticas y Asignación	6
Casos de prueba 2: Errores de Paréntesis, Puntos y Comas, y Variables No Declaradas	8
Caso de prueba 3: Errores en Declaraciones de Variables	10
Caso de prueba 4: Errores en Ciclos	12
Caso de prueba 5: Errores en Funciones y Parámetros	14
Caso de prueba 6: General	16
Manual de usuario	17
Bitácora	18
Bibliografía	20

Introducción

Este proyecto se ubica en la segunda etapa de la creación de un compilador para el lenguaje de programación C, conocida como el Parsing o Análisis Sintáctico. El propósito principal de esta etapa es diseñar y desarrollar un parser que logre identificar como se relacionan los tokens que se obtienen de la fase de análisis léxico y determinar si estos tokens forman expresiones, sentencias o estructuras válidas para un programa escrito en lenguaje C. Para lograr esto, se utilizaron dos herramientas Java CUP y JFlex, Java CUP permitió definir reglas gramaticales que describen como deben combinarse los tokens para ser reconocidos por el parser como correctos. Mientras que JFlex fue utilizado en la etapa de análisis Léxico, esta etapa se desarrolló en la primera fase del proyecto.

A lo largo de este documento se detalla el proceso de desarrollo del scanner, desde la planificación hasta la implementación y evaluación de los resultados obtenidos. Se describen decisiones técnicas tomadas para garantizar el correcto funcionamiento del proyecto, así como las estrategias empleadas para la solución de los problemas encontrados durante su desarrollo.

Estrategia de solución

Después de leer detenidamente la documentación de JFlex, se comenzó a diseñar las expresiones regulares para los tokens que debía reconocer el escáner. Aquí surgió el primer problema: el escáner reconoce los tokens según el orden de prioridad. Es decir, si la primera expresión regular es un punto, ningún otro token será reconocido, ya que este metacaracter coincide con cualquier carácter, interpretándolo como un error. Por lo tanto, fue crucial definir adecuadamente el orden de las expresiones regulares.

Una vez definido el orden de las expresiones regulares, procedimos a diseñar la estructura de los tokens y sus tipos. Para ello, decidimos crear un mapa que facilitara la búsqueda de tokens repetidos en el documento. De manera similar, cada token guarda las líneas en las que aparece en un mapa, lo que permite incrementar el contador de ocurrencias de un token en una misma línea. Finalmente, los tipos de tokens se almacenan en un enum.

Por último, se implementaron errores definidos, como un número seguido de un identificador o números flotantes que comienzan con un punto. Era necesario definir estos errores como tokens para que pudieran ser reconocidos por el escáner. Los errores no se almacenan en una estructura de datos; en su lugar, se imprimen

directamente y no se agregan al mapa de tokens.

Análisis de resultados

Actividad	Porcentaje realizado	Justificación
Desplegar lista de errores léxicos	100%	
Desplegar lista de errores sintácticos	100%	
Evitar desplegar errores sintácticos en cascada	50%	En situaciones muy específicas el parser se puede caer o dar errores poco precisos. Esto debido a complicaciones en la gramática.
Implementar las funciones de read and write	100%	
Declaración de variables, constantes y lista de variables	100%	
Reconocer la estructura del programa	100%	
Identificar funciones	100%	
Analizar expresiones con operadores aritméticos o booleanos	100%	
Identificar todas las estructuras de control	100%	
Definir buenos mensajes de error	80%	La mayoría de errores están bien definidos, pero si el flujo es erróneo no se obtiene un buen mensaje.

Lecciones aprendidas

Durante el desarrollo del proyecto, el grupo pudo observar que en el parser es fundamental definir correctamente la gramática y las reglas de producción. La estructura de las reglas y el uso adecuado de precedencia y asociatividad permitieron reducir conflictos de análisis, como conflictos shift/reduce o reduce/reduce, que dificultan el proceso de análisis sintáctico y provocan errores de interpretación. Además, organizar adecuadamente las producciones y sus alternativas evitó que ciertas estructuras se interpretaran de forma ambigua, lo cual agregó robustez al parser al generar una mejor comprensión de la jerarquía entre expresiones y bloques de código.

Definir mensajes de error claros fue un reto, especialmente al manejar errores sintácticos complejos. En muchos casos, el parser debía identificar la causa específica de un error (como la falta de un punto y coma o el uso incorrecto de una palabra reservada) y generar un mensaje informativo que ayudara a identificar el problema en la línea correspondiente. Esto fue esencial para mejorar la experiencia del usuario al facilitar la corrección de errores en el código fuente. Asimismo, el equipo experimentó con la recuperación de errores para continuar el análisis a pesar de errores sintácticos, lo cual permitió al parser brindar un reporte de múltiples errores en una sola ejecución, aumentando la eficiencia del proceso de depuración.

Al enfrentarse a estructuras sintácticas complejas como anidación de bloques y condiciones, el equipo comprendió la importancia de anticipar casos especiales en las reglas de producción. Definir reglas claras para estos casos evitó ambigüedades en el análisis y permitió un procesamiento más preciso del código fuente. Esto también permitió al parser identificar y manejar constructos incompletos o mal formados de manera más efectiva, brindando así una base sólida para la interpretación y compilación del código.

Casos de prueba

Caso de prueba 1: Errores en Expresiones Aritméticas y Asignación

```
void testAsignacionAritmetica() {  
    int a = 5;  
    int b = 10;  
  
    // Error: Falta operando después del operador '+'  
    int c = a + ;  
  
    // Error: Falta operando antes del operador '-'  
    int d = - b;  
  
    // Error: Operador de asignación en una expresión aritmética  
    a + b = c;  
  
    // Error: Uso de variable no declarada en una operación  
    int e = a + f;  
  
    // Error: Operación aritmética sin un operando válido  
    int g = + + b;  
  
    // Error: Falta el punto y coma en la asignación  
    int h = a + c  
  
    // Expresión correcta (para contraste)  
    int i = a * b;  
}
```

Errores esperados:

- Error de sintaxis en línea 6: Error en la expresión
- Error de sintaxis en línea 9: Falta operando antes de '-'.
- Error de sintaxis en línea 12: Error antes de '=' en asignación.
- Error de sintaxis en línea 15: Falta el tipo de dato en la declaración.
- Error de sintaxis en línea 18: Falta operando antes de '+'.

- Error de sintaxis en línea 18: Falta operando antes de '+'.
- Error de sintaxis en línea 21: Error en la expresión

Resultados:

Starting parsing process...

Error de sintaxis en línea 6: Error en la expresión

Error de sintaxis en línea 9: Falta operando antes de '-'.

Error de sintaxis en línea 12: Error antes de '=' en asignación.

Error de sintaxis en línea 15: Falta el tipo de dato en la declaración.

Error de sintaxis en línea 18: Falta operando antes de '+'.

Error de sintaxis en línea 18: Falta operando antes de '+'.

Error de sintaxis en línea 21: Error en la expresión

Caso de prueba 2: Errores en Estructuras de Control

```
void testEstructurasControl() {
    int x = 5;
    int y = 10;
    // Error: Falta paréntesis de cierre en la condición del 'if'
    if (x < y {
        x = x + 1;
    }
    // Error: Falta paréntesis de apertura en la condición del 'while'
    while x < y) {
        y = y - 1;
    }
    // Error: Falta llave de apertura en el bloque del 'else'
    if (x > y) {
        print(x);
    } else
        print(y);
    }
    // Error: Condición en 'if' sin paréntesis alrededor
    if x == y {
        x = x * 2;
    }
    // Error: Falta el punto y coma al final de la condición del 'while'
    while (x < y) {
        x = x + 1
    }
    // Expresión correcta (para contraste)
    if (x != y) {
        return;
    } else {
        return;
    }
}
```

Errores esperados:

- Error de sintaxis en línea 8: Error en el paréntesis de cierre del if.
- Error de sintaxis en línea 13: Falta paréntesis de apertura en el while.

- Error de sintaxis en línea 20: Error en la llave de apertura del else.
- Error de sintaxis en línea 25: Error en los paréntesis de la condición del if.
- Error de sintaxis en línea 29: Falta punto y coma.

Resultados:

Starting parsing process...

Error de sintaxis en línea 8: Error en el paréntesis de cierre del if.

Error de sintaxis en línea 13: Falta paréntesis de apertura en el while.

Error de sintaxis en línea 20: Error en la llave de apertura del else.

Error de sintaxis en línea 25: Error en los paréntesis de la condición del if.

Error de sintaxis en línea 29: Falta punto y coma.

Caso de prueba 3: Errores en Declaraciones de Variables

```
// Error: Identificador sin tipo de dato
y;
//Declaracion de una variable
int x;
// Declaracion correcta de multiples variables constantes
const int z, t, y, y;
// Error: Falta punto y coma en la declaración
int g, h, i,
void testDeclaracionesVariables() {
    // Error: Uso incorrecto de 'void' como tipo de dato en una variable
    void variableInvalida;
    // Error: Falta identificador en la declaración
    int ;
    // Error: Declaración incompleta con un tipo de dato y asignación faltante
    const int ; // Falta un identificador o asignación a la variable constante
    // Declaración correcta (para contraste)
    int valorCorrecto = 10;
    char letra = 'A';
}
```

Errores esperados:

- Error de sintaxis en línea 5: Falta el tipo de dato en la declaración.
- Error de sintaxis en línea 11: Falta punto y coma.
- Error de sintaxis en línea 16: Una variable no puede ser declarada void.
- Error de sintaxis en línea 19: Falta el identificador de la variable en la declaración.
- Error de sintaxis en línea 22: Falta el identificador de la variable en la declaración.

Resultados:

Starting parsing process...

Error de sintaxis en línea 5: Falta el tipo de dato en la declaración.

Error de sintaxis en línea 11: Falta punto y coma.

Error de sintaxis en línea 16: Una variable no puede ser declarada void.

Error de sintaxis en línea 19: Falta el identificador de la variable en la declaración.

Error de sintaxis en línea 22: Falta el identificador de la variable en la declarac

Caso de prueba 4: Errores en Ciclos

```
void testCiclos() {
    int x = 0;

    // Error: Falta punto y coma en la declaración de 'for'
    for (int i = 0 i < 10; i++) {
        x += i;
    }
    // Error: While vacio
    while () {
        x += 1;
    }
    // Error: Falta paréntesis de cierre en la condición del 'while'
    while (x < 10 {
        x += 1;
    }
    // Error: do while vacio
    do {
        x += 1;
    } while ();
    // Error: for vacio
    for () {
        x += 1;
    }
    // Error: for con una sola expresion
    for (;) {
        x += 1;
    }
    // Error: Falta paréntesis de apertura en la condición del 'do-while'
    do {
        x += 2;
    } while x < 20);
    // Error: Falta punto y coma al final de la condición en 'do-while'
    do {
        x -= 1;
    } while (x > 0)
    // Error: Falta llave de apertura en el bloque del 'for'
    for (int j = 0; j < 5; j++)
```

```

x += j;
}
// Ciclo correcto (para contraste)
for (int k = 0; k < 5; k++) {
    x += k;
}
}

```

Errores esperados:

- Error de sintaxis en línea 5: Falta punto y coma.
- Error de sintaxis en línea 12: Error en la condición del while.
- Error de sintaxis en línea 17: Error en el paréntesis de cierre del while.
- Error de sintaxis en línea 22: Error en la condición del do-while.
- Error de sintaxis en línea 27: Error en la sentencia del for.
- Error de sintaxis en línea 30: No puede haber una sola expresión.
- Error de sintaxis en línea 37: Falta paréntesis de apertura en el do-while.
- Error de sintaxis en línea 42: Falta punto y coma.
- Error de sintaxis en línea 47: Error en la llave de apertura del for.

Resultados:

Starting parsing process...

Error de sintaxis en línea 5: Falta punto y coma.

Error de sintaxis en línea 12: Error en la condición del while.

Error de sintaxis en línea 17: Error en el paréntesis de cierre del while.

Error de sintaxis en línea 22: Error en la condición del do-while.

Error de sintaxis en línea 27: Error en la sentencia del for.

Error de sintaxis en línea 30: No puede haber una sola expresión.

Error de sintaxis en línea 37: Falta paréntesis de apertura en el do-while.

Error de sintaxis en línea 42: Falta punto y coma.

Error de sintaxis en línea 47: Error en la llave de apertura del for.

Caso de prueba 5: Errores en Funciones y Parámetros

```
// Error: Falta tipo de retorno en la declaración de la función (No funciona)
suma(int a, int b) {
    return a + b;
}

// Error: Falta identificador en el parámetro de la función
int resta(int a, ) {
    return a - b;
}

// Error: Falta llave de cierre en el cuerpo de la función (Tira Falta punto y coma)
int multiplica(int a, int b)
    int resultado = a * b;
    return resultado;
}

int dividir int a, int b) {
    return a / b;
}

int dividir (int a, int b {
    return a / b;
}

void calcularSuma(int a, int b) {
    return a + b;
}

// Función correcta (para contraste)
int modulo(int a, int b) {
    return a % b;
}
```

Errores esperados:

- Error de sintaxis en línea 2: Falta el tipo de dato en la declaración de la función.
- Error de sintaxis en línea 7: Error en la coma de los parametros.
- Error de sintaxis en línea 16: Error en las llaves de la función.
- Error de sintaxis en línea 18: Error en los paréntesis de la función.
- Error de sintaxis en línea 22: Error en la coma de los parametros.

Resultados:

Starting parsing process...

Error de sintaxis en línea 2: Falta el tipo de dato en la declaración de la función.

Error de sintaxis en línea 7: Error en la coma de los parametros.

Error de sintaxis en línea 16: Error en las llaves de la función.

Error de sintaxis en línea 18: Error en los paréntesis de la función.

Error de sintaxis en línea 22: Error en la coma de los parametros.

Caso de prueba 6: General

```
void funcionGeneral() {
    int a = 5;
    int b = 10
    // Error: Declaración de variable con tipo inválido
    void varInvalida = 20;
    // Error: asignación inválida
    a + b = c;
    // Error: Falta paréntesis de cierre en la condición del 'if'
    if (a < b {
        print(a);
    } else {
        print(b)
    }
    // Error: Falta el paréntesis de apertura en el 'while'
    while a < b) {
        a = a + 1;
    }
    // Error: Falta punto y coma en la declaración de 'for'
    for (int i = 0 i < 10; i++) {
        print(i);
    }
    // Error: Paréntesis no cerrado en una expresión aritmética
    a + (b * 2 ;

    // Declaración correcta para contraste
    if (x != a) {
        print("x y a son diferentes");
    }
}
```

Errores esperados:

- Error de sintaxis en línea 3: Falta punto y coma.
- Error de sintaxis en línea 5: Una variable no puede ser declarada void.
- Error de sintaxis en línea 8: Error antes de '=' en asignación.
- Error de sintaxis en línea 14: Falta punto y coma.

- Error de sintaxis en línea 13: Error en el paréntesis de cierre del if.
- Error de sintaxis en línea 20: Falta paréntesis de apertura en el while.
- Error de sintaxis en línea 23: Falta punto y coma.
- Error de sintaxis en línea 28: Error en paréntesis de expresión.
- Error de sintaxis en línea 33: Error en la llave de apertura del if.

Resultados:

Starting parsing process...

Error de sintaxis en línea 3: Falta punto y coma.

Error de sintaxis en línea 5: Una variable no puede ser declarada void.

Error de sintaxis en línea 8: Error antes de '=' en asignación.

Error de sintaxis en línea 14: Falta punto y coma.

Error de sintaxis en línea 13: Error en el paréntesis de cierre del if.

Error de sintaxis en línea 20: Falta paréntesis de apertura en el while.

Error de sintaxis en línea 23: Falta punto y coma.

Error de sintaxis en línea 28: Error en paréntesis de expresión.

Error de sintaxis en línea 33: Error en la llave de apertura del if.

Manual de usuario

Instalación

Para construir y ejecutar el proyecto, es necesario tener Java instalado en tu sistema. Sigue estos pasos para configurar el proyecto:

1. Clona el repositorio:

```
git clone https://github.com/AlonsoNav/CCompilerJFlex.git
cd your-repo
```

2. Genera el archivo CLexer:

```
java -jar lib/jflex-full-1.9.1.jar src/scanner/CLexer.flex
```

3. Genera el archivo Parser:

```
java -jar lib/java-cup-11b.jar -parser Parser -symbols Symbol
src/parser/Parser.cup
```

4. Compila el proyecto:

```
javac -d bin -sourcepath src -cp lib/java-cup-11b.jar
src/app/ParserMain.java src/app/ScannerMain.java
src/scanner/CLexer.java src/scanner/Token.java
src/scanner/TokenType.java src/parser/Parser.java
src/parser/Sym.java
```

Uso

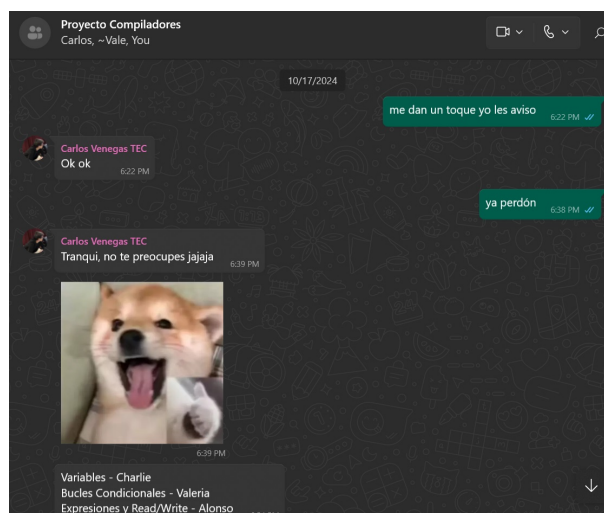
Para ejecutar el compilador con un archivo de entrada, utiliza el siguiente comando:

```
java -cp bin app.ParserMain input_file
```

Bitácora

Fecha: 17-10-2024

Para iniciar el proyecto hicimos una pequeña reunión en la que se estableció que CV se encargaba de las variables, declaraciones y constantes. Por otro lado, VG iba a trabajar las estructuras de control como condicionales y bucles. AN diseñaría las gramáticas para las expresiones, y las funciones como read y write. CV también trabajó en la conversión de los tokens del scanners a Symbol y además de dar la estructura básica del programa. A continuación se adjunta evidencia de lo hablado en un chat de WhatsApp:



Fecha: 26-10-2024

En este día nos conectamos un rato en Discord para revisar lo que tenemos trabajado hasta el momento y discutir las próximas etapas del proyecto. Para el manejo de errores se hizo otra distribución donde CV se encargaba de errores de expresiones y de read and write, AN de errores de estructuras de control y VG de errores de variables, constantes y funciones.

Fecha: 04-09-2024

Para este punto se tienen muchas complicaciones con el manejo de errores, por lo que a partir de este día se tienen reuniones consecutivas donde se hace trabajo cooperativo con el fin de sacar las funcionalidades. Por último, se divide las partes de la documentación que serán redactadas por todos los miembros del equipo.

Bibliografía

- [1] Klein, G., Rowe, S., & Décamps, R. (marzo de 2023). *JFlex User's Manual*. JFlex Team. En: <https://www.jflex.de/manual.html>.
- [2] Hudson, S. (julio de 1999). *CUP User's Manual*. Cup. En: <https://www.cs.princeton.edu/~appel/modern/java/CUP/manual.html#intro>.