Fase 2 del Proyecto de Compiladores – Análisis Sintáctico

Materia: Compiladores I

Docente: Dra. Blanca Guadalupe Estrada Rentería

Semestre: 8º ISC

Fecha de entrega: 19 y 20 de Junio

Objetivo General

El objetivo de esta segunda fase es diseñar e implementar un analizador sintáctico descendente recursivo que lea y procese los tokens ya validados y generados por el analizador léxico desarrollado en la Fase 1. A partir de estos tokens, el analizador sintáctico construirá el árbol sintáctico abstracto (AST) del programa, validará su estructura gramatical y reportará cualquier error sintáctico con detalles de línea y columna.

Gramática Base

programa → `main { lista_declaracion }`
lista_declaracion → `lista_declaracion declaracion | declaracion`
declaracion → `declaracion_variable | lista_sentencias`
declaracion_variable → `tipo identificador ;`
identificador → `id | identificador , id`
tipo → `int | float | bool`
lista_sentencias → `lista_sentencias sentencia | ε`
sentencia → `seleccion | iteracion | repeticion | sent_in | sent_out | asignacion`
asignacion → `id = sent_expresion`
sent_expresion → `expresion ; | ;`
seleccion → `if expresion then` **lista_sentencias** `[ else` **lista_sentencias** `] end`
iteracion → `while expresion` **lista_sentencias** `end`
repeticion → `do` **lista_sentencias** `while expresion`
sent_in → `cin >> id ;`
sent_out → `cout << salida`
salida → `cadena | expresion | cadena << expresion | expresion << cadena`
expresion → `expresion_simple [ rel_op expresion_simple ]`
rel_op → `< | <= | > | >= | == | !=`

expresion_simple → `expresion_simple suma_op termino | termino`
suma_op → `+ | - | ++ | --`
termino → `termino mult_op factor | factor`
mult_op → `* | / | %`
factor → `factor pot_op componente | componente`
pot_op → `^`
componente → `( expresion ) | número | id | bool | op_logico componente`
op_logico → `&& | || | !`
cadena → `"cualquier texto"`

🛠 Requisitos Técnicos

1. Entrada desde archivo de tokens:
    - El analizador debe leer un archivo de texto con los tokens generados por el analizador léxico.
    - El archivo debe incluir tipo de token, lexema, línea y columna.

2. Análisis sintáctico y construcción del AST:

- Validar que el programa cumple con la gramática.
- Construir un Árbol Sintáctico Abstracto (AST).

3. Visualización del árbol (obligatorio):

- El árbol sintáctico debe mostrarse de forma gráfica y colapsable, como una estructura de carpetas.

4. Manejo de errores:

- Reportar errores con tipo, línea y columna. Intentar continuar si no son fatales.

📦 Entregables

- Código fuente del analizador sintáctico
- Archivo con visualización gráfica del árbol sintáctico
- Archivo con lista de errores sintácticos detectados
- Documento PDF con descripción técnica, gramática, ejemplos y capturas
- Archivos de prueba (mínimo 2 válidos y 1 con errores)
- Block de notas con nombres y matrículas de los integrantes

Validación esperada

El analizador debe validarse con programas que incluyan estructuras como declaraciones múltiples, condicionales, ciclos, operaciones aritméticas y lógicas, incrementos, sentencias de entrada/salida, etc.

Nota Importante sobre la entrega y penalización

- La calificación máxima es 10 si se entrega el día asignado (19 o 20 de junio).
- Por cada día de retraso se descuenta 1 punto completo.
- La entrega solo se acepta si el programa está 100% funcional.
- Incluso si la calificación llega a cero, la entrega del proyecto es obligatoria.