**PROCESADORES DE LENGUAJES**

**TEMA 1. INTRODUCCIÓN**

* **Compilación**: proceso de traducción que convierte un programa escrito en un lenguaje (Java, C, …) a un programa equivalente en otro lenguaje (ensamblador).
* **Intérprete**: compilador paso a paso.
* **Ensamblador**: compilador de lenguaje ensamblador.
* **Nº de pasadas:** nº de lecturas desde el inicio del fichero hasta el final.

**LENGUAJES Y GRAMÁTICAS:**

* **GRAMÁTICA (N, T, P, S)**

**N:** Símbolos no terminales - A, B, D

**T:** Símbolos terminales - a, for, 1

**P:** reglas de producción - A -> aD

**S:** Axioma o símbolo inicial

* Analizador sintáctico -> gramática tipo 2
* Analizador léxico -> gramática tipo 3
* **2 módulos:**
  + T.S.: Tabla de símbolos, guarda las variables del usuario
  + G.E.: Gestor de errores
* **Objetivo:** llegar a código máquina.

**FASES DEL PROCESO DE COMPILACIÓN**

Fichero fuente -> An. Léxico -> An. Sintático -> An. Semántico (Análisis sintáctico anotado) ->

1. **Análisis Léxico:** lee el fichero, lo recorre carácter a carácter y extrae sus “palabras”.

**Ejemplo:** límite:=largo x alto = 5

<var, 1> -> identificador=variable

<asig, - >  
 <MUL, - >

<menos, - >

<NUM, 1>

1. **Análisis Sintáctico:** leer los tokens y agruparlos -> generar el árbol sintáctico.

**1º identifica** < var, 2 >

5º

3º

**2º identifica** < var, 3 >

6º

**4º identifica** < num, 1 >

< var, 1 >

< asig, >

1. **Análisis Semántico:** recorre el árbol de arriba abajo y de izquierda a derecha y comprueba si es correcto.

**FASE DE ANÁLISIS**

Generador de Código Intermedio (Programa Intermedio) -> Optimizador de Código -> Generador de código (Lenguaje Objeto) -> Programa Objeto

* 3 pasos de análisis -> procesador de lenguajes
* 3 pasos de síntesis -> traductor de lenguajes

1. **Generador de código intermedio:** recibe el árbol sintáctico por a. semántico, lo traduce a un lenguaje intermedio más sencillo.

t1: = largo \* alto

t2: = ent – real

t3: = t1 – t2

límite: = t3

1. **Optimizador de Código:** optimiza Código intermedio -> Código de más calidad

t2: = 1.0 | t3: = t1 – 1.0 | t1:= largo\*alto

límite: = t1 – 1.0

1. **Generador de código:** genera las instrucciones y las traduce a ensamblador

MOV largo, R1

MUL alto, R1

MOV R1, E1

MOV E1, R1

SUB 1.0, R1

MOV R1, límite

1. **Optimizador de código:** optimiza el código final
2. **Tabla de símbolos.**

**TEMA 2: ANALISIS LÉXICO**

**Objetivo**: proporcionar un token al analizador sintáctico

1. **FUNCIÓN DEL ANALIZADOR LÉXICO**

Encargado de leer el fichero fuente.

* + Lee uno a uno los caracteres y envía las palabras el analizador sintáctico en forma de token.
  + Se encarga de eliminar los elementos que no aportan información (espacio en blanco, salto de línea, comentarios, …).
  + Rellenar parte de la información en la T.S.
  + Relacionar errores con su posición en el texto fuente.
* Cadena de caracteres: empiezan por comillas “”.
* Identificador: no empieza por comillas.
* Token: cada elemento del lenguaje que tiene significado propio (símbolos terminales).
* Patrón: regla o norma que nos dice cómo se escriben todas las palabras correspondientes a un determinado token.
* Lexema: sentencias de caracteres en el fichero de entrada que concuerda con un patrón de un determinado token.

Ejemplo: x = 0

3 tokens, 1 símbolo en la TS (la x, que es el único identificador).

* **TOKEN**

Tipo\_token: análisis sintáctico

Atributo: análisis semántico, traductor

< Tipo\_token, Atributo >

1. **ERRORES**

Error léxico cuando no se logra equiparar una secuencia de caracteres con alguno de los patrones del lenguaje -> No se corresponde con un tipo.

Errores típicos: - carácter no válido en el lenguaje de token válido.

- Nombre de usuario (identificador, variable) no válido.

- Constante real no válida.

1. **DISEÑO ANALIZADOR LÉXICO**
2. Identificar los tokens de un lenguaje.
3. Diseñar la gramática (Regular, Tipo 3, gramática que reconozca cada lexema).
4. Construir autómata finito determinista.
5. Completar autómata con acciones semánticas.
6. Identificar situaciones de error.
7. TOKENS: enteros y reales son dos tokens diferentes, siempre se pone atributo “valor” al nº real o entero.
8. Gramática tipo 3:

{ A-> aB

A -> a }

A -> l B | : c | \* | del A

B -> l B | d B | λ

C -> =

La gramática se termina cuando se genera un token.

1. AFD:
   * Determinista: sabiendo el carácter de entrada, sabes donde transita.

Ejemplos:

* ¿Nº de tokens identificadores por An. Léxico?

int índice = 1; -> 5 tokens

while (n<=10){ } -> 8 tokens

c = b; -> 4 tokens

* ¿Son pares Tipo\_token, Atributo correctos?

<identificador, punteroTS> -> si

<constante\_entera, > -> no

<abre\_parentesis, valor> -> no

<punto\_y\_coma, > -> si

<identificador, > -> no

<constante\_entera, valor> -> si

<FOR, > -> si

<operador\_aritmetico, 2> -> si

<operador\_logico, valor> -> no

<lexema, punteroTS> -> no

<operador\_relacional, 2> -> si

<punto\_coma, punto\_coma> -> no

**TEMA 3: TABLA DE SÍMBOLOS**

Estructura de datos donde se almacena la información relevante sobre los identificadores del programa.

Todos los módulos del compilador tienen acceso.

* Analizador Léxico: función llamada del An. Sintáctico que retorna un token.
* Analizador Léxico: tiene acceso al programa. Le pasa un token al An. sintáctico.
* Analizador Sintáctico: comprobar si la estructura recibida del léxico es correcta.

Cuando el A. Léxico lee un identificador, lo lleva a la T.S.

* Analizador Semántico: crea la TS y rellena la información.

Para cada entrada hay atributo:

· Lexema: nombre del identificador (el an. léxico solo mete el lexema).

· Tipo: entero, real, cadena, array… (dependiendo del tipo, se tienen unos atributos determinados).

* Analizador léxico no sabe en qué tabla meter el lexema. No tiene visión de contexto. Eso lo hace el Analizador Semántico.
* VAR indica que el parámetro se pasa por referencia y no por valor.

Operaciones que podemos hacer:

1. Insertar ID.
2. Consultar ID.
3. Crear T.S. Vacía.
4. Cambiar el valor de un atributo.
5. Meter un atributo.
6. Borrar la tabla.

* Analizador Léxico: añadir un nuevo lexema. Buscar si un lexema está en la tabla -> comprobar si ya se creó una entrada para ese identificador.