1.6 MAPA CONCEPTUAL

PROCESADORES DE LENGUAJE

SON PROGRAMAS QUE ANALIZAN

Interpretan o traducen lenguajes de programación. Incluyen:

- Compiladores: traducen código fuente a código máquina.
- Intérpretes: ejecutan el código directamente.
- Ensambladores y traductores: convierten entre distintos niveles de lenguaje.

ANÁLISIS LÉXICO

TOKENS

- Identifica los tokens del código fuente (palabras clave, identificadores, operadores).
- Elimina espacios y comentarios.
- Usa autómatas finitos para reconocer patrones.

ANÁLIS

ANÁLISIS SINTÁCTICO

- Verifica la estructura gramatical del código.
- Construye árboles de derivación o árboles sintácticos.
- Usa algoritmos como LL, LR, SLR para el análisis.

ANÁLISIS SEMÁNTICO

- Comprueba el significado del código.
- Detecta errores como tipos incompatibles o variables no declaradas.
- Usa gramáticas atribuidas para asociar significado a la sintaxis.

DISEÑAR E IMPLEMENTAR COMPILADORES

IMPLICA CONSTRUIR SOFTWARE

transforme código fuente en ejecutable. Requiere:

- Definir la gramática del lenguaje.
- Implementar cada fase del compilador.
- Validar que el código generado sea correcto y eficiente.

IMPACTOS EN EL COMPILADOR

COMPILADOR

- Cambios en hardware, paradigmas de programación y lenguajes influyen en el diseño del compilador.
- La evolución hacia lenguajes funcionales, concurrentes o interpretados exige nuevas técnicas de análisis y generación.

AVANCE A LOS LENGUAJES DE ALTO NIVEL

- Los lenguajes modernos (Python, Java, etc.) requieren compiladores más complejos.
- Se enfocan en abstracción, seguridad y portabilidad.

HERRAMIENTAS DE CONSTRUCCIÓN DE COMPILADORES

- Lex/Yacc, Flex/Bison: para análisis léxico y sintáctico.
- ANTLR, LLVM, GCC: frameworks modernos para compiladores.

ESTRUCTURA DE UN COMPILADOR

SE DIVIDE EN FASES PRINCIPALES:

1. Análisis léxico 2. Análisis sintáctico 3. Análisis semántico 4. Generación de código intermedio 5. Optimización de código 6. Generación de código final 7. Administración de la tabla de símbolos

GENERACIÓN DE CÓDIGO INTERMEDIO

CARACTERISTICAS

- Traduce el árbol sintáctico a una representación abstracta (como tres direcciones o código tipo TAC).
- Facilita la optimización y portabilidad.

GENERACIÓN DE CÓDIGO

- Convierte el código intermedio en instrucciones específicas del procesador.
- Considera registros, memoria y arquitectura del sistema.

OPTIMIZACIÓN DE CÓDIGO

IDEA

- Mejora el rendimiento sin alterar el comportamiento.
- Ejemplos: eliminación de código muerto, propagación de constantes, reducción de fuerza.

Eduardo Daniel Segura Páramo