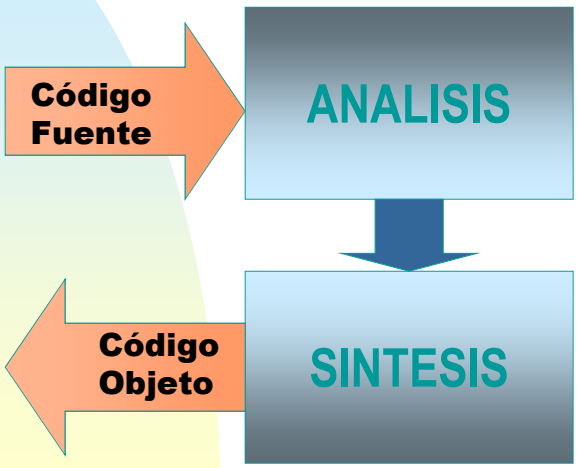
Lenguajes de programación

¿Cómo se construye un compilador?

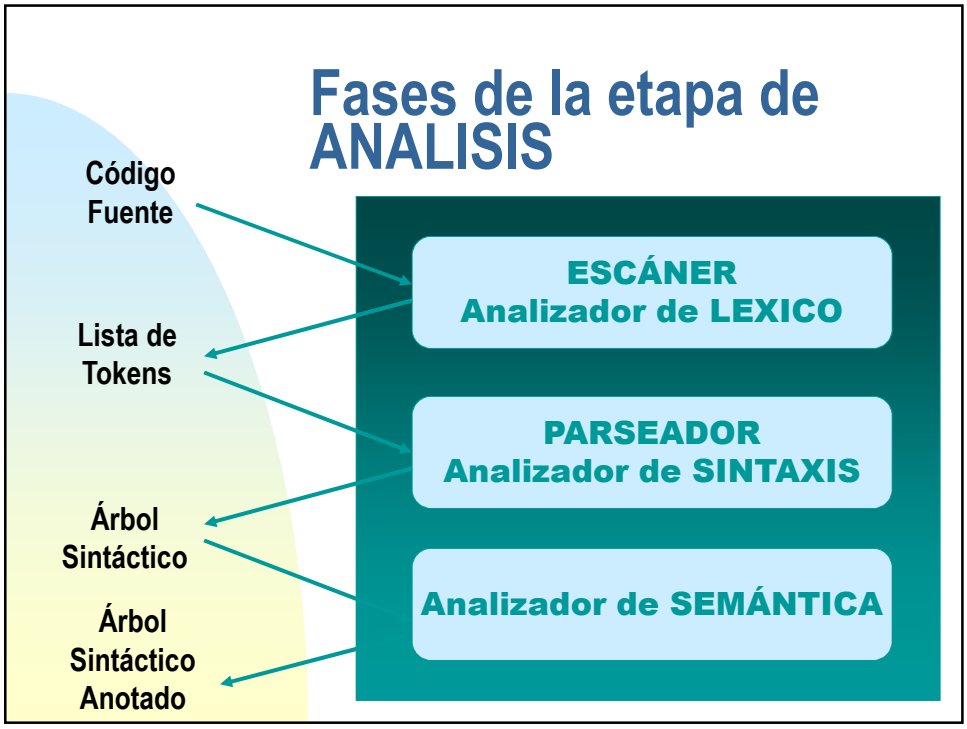
=> ANÁLISIS LÉXICO

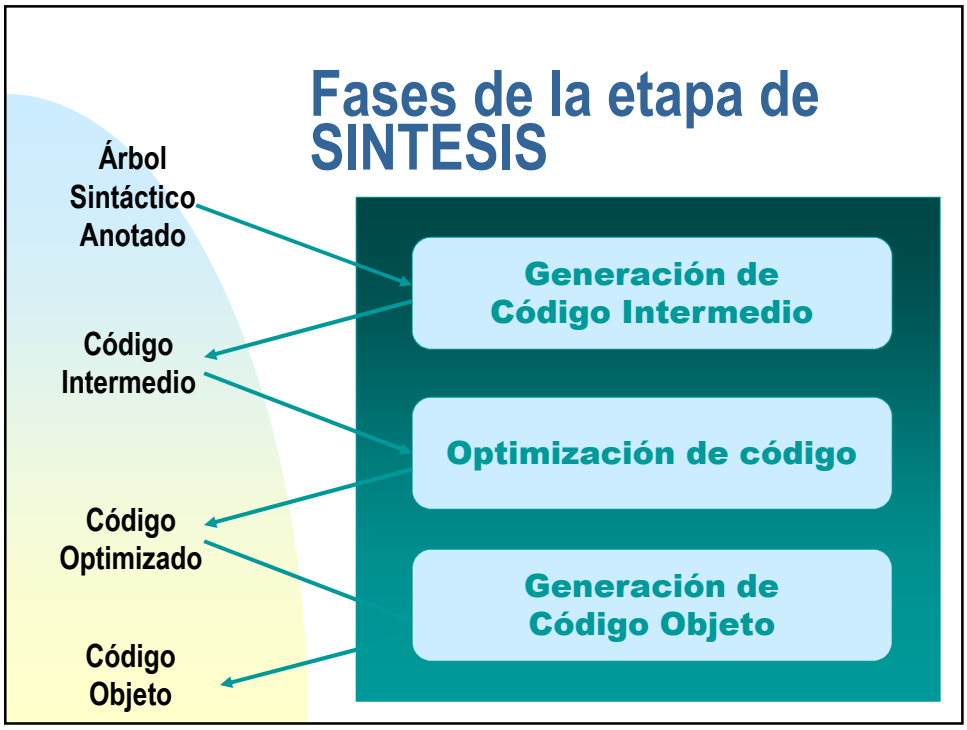
Etapas del proceso de compilación



Descompone el código fuente para verificar el uso correcto del lenguaje

Transforma a los elementos reconocidos del lenguaje en el código objeto


Fases de la etapa de ANALISIS
Código Fuente
Lista de Tokens
Árbol Sintáctico
Árbol Sintáctico Anotado
ESCÁNER Analizador de LEXICO
PARSEADOR Analizador de SINTAXIS
Analizador de SEMÁNTICA


Fases de la etapa de SINTESIS
Árbol Sintáctico Anotado
Código Intermedio
Código Optimizado
Código Objeto
Generación de Código Intermedio
Optimización de código
Generación de Código Objeto

Analizador de LÉXICO

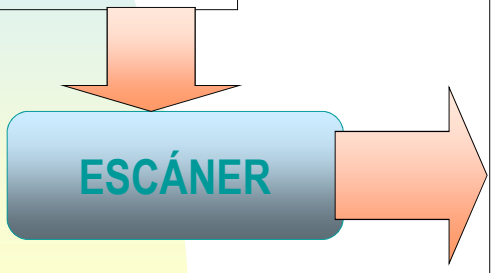
- Lee caracter por caracter el texto de entrada, y los agrupa tratando de reconocer elementos válidos del lenguaje.
- Elimina espacios y líneas en blanco, tabuladores y los comentarios del programa.
- Asocia a cada elemento reconocido con una clave llamada TOKEN.
- Genera mensajes de error cuando no reconoce elementos válidos.

Asignación de TOKENS

- Los símbolos y palabras con significado único, reciben una clave de token particular.
- Los elementos que son del mismo tipo, reciben una clave de token genérico, y se guardan en memoria (tabla de símbolos). Ejemplo: identificadores y constantes

Ejemplo

```
main()
{ //programa que no hace nada
   int a;
   a = a * 2;
}
```



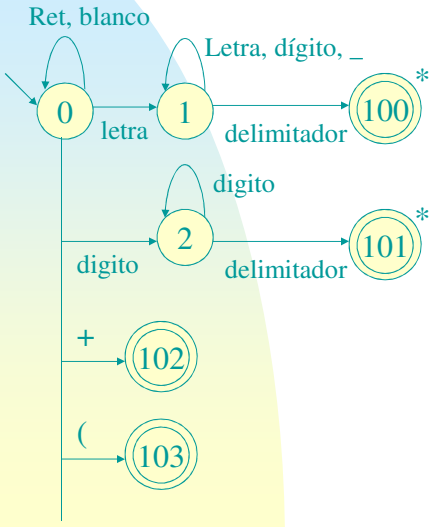
Lista de Tokens:
PR_main (5)
Paréntesis_abierto (12)
Paréntesis_cerrado (13)
Llave_abierta (18)
PR_int(4)
identificador (20, dirX)
Puntoycoma (33)
identificador (20, dirX)
Op= (67)
identificador (20,dirX)
Op* (58)
Cte_entera (25, dirZ)
Puntoycoma (33)
Llave_cerrada (19)

¿Cómo se construye un Analizador de Léxico?

- Diseñar un autómata de estados finitos determinístico que reconozca a todos los elementos del lenguaje.
- Representar el autómata en memoria.
  - Matriz de transiciones.
- Implementar un manejador de la matriz.

- Utilizar un generador automático de scanners: *LEX* o *FLEX*.
  - ER->AEFND->AEFD->Matriz

Ejemplo



Estado \ Símbolo	letra	digito	_	+	(
0	1	2	E	102	103
1	1	1	1	100	100
2	E	2	E	101	101

Algoritmo general para manejar la matriz

- Inicializar en Estado=0
- Mientras el Estado no sea ACEPTOR o ERROR:
  - **Leer un caracter del archivo**
  - **Obtener NuevoEstado = MATRIZ[Estado, caracter]**
  - **Si el NuevoEstado es ACEPTOR, generar el token, almacenar el lexema y volver al Estado = 0**
  - **Si el NuevoEstado es de ERROR, marcarlo y volver al Estado = 0**
  - **Si el NuevoEstado no es ACEPTOR ni ERROR, anexar el caracter en el lexema y hacer Estado = NuevoEstado**

EJEMPLO en C++

```
#include <iostream>
using namespace std;

// Matriz de transiciones
//              dig   op    (    )  raro  esp    .    $
int MT[5][8] = {{  1, 102, 105, 106,   4,   0,   4, 107},    // edo 0 - edo inicial
                {  1, 100, 100, 100, 100, 100,   2, 100},    // edo 1 - digitos enteros
                {  3, 200, 200, 200,   4, 200,   4, 200},    // edo 2 - primer decimal flotante
                {  3, 101, 101, 101, 101, 101,   4, 101},    // edo 3 - decimales restantes flotante
                {200, 200, 200, 200,   4, 200,   4, 200} };  // edo 4 - edo de error

int filtro (char c)
{
           switch (c)
           {  case '0':   case '1':   case '2':
              case '3':   case '4':   case '5':
              case '6':   case '7':   case '8':
              case '9': return 0; // decimales

              case '+':   case '-':   case '*':
              case '/': return 1; // operadores

              case '(': return 2; // delimitador (
              case ')': return 3; // delimitador )

              case ' ':   case  10:
              case  13: return 5; // blancos
              case '.': return 6; // punto
              case '$': return 7; // fin de entrada
              default:  return 4; // caracter raro (ilegal)
           }
}
```

EJEMPLO en C++

```
int main(void) {
        char c;
        string lexema = "";
        int edo = 0;
        while (true) {
                do {   // mientras el estado no sea ACEPTOR ni ERROR
                        c = getchar();
                        edo = MT[edo][filtro(c)];
                        if (edo < 100 && edo != 0) lexema += c;
                } while (edo < 100);
                switch (edo) {   // token reconocido, error o fin de entrada
                case 100: cout << "Entero " << lexema << endl;
                        ungetc(c, stdin);   // regresa el delimitador
                        break;
                case 101: cout << "Flotante " << lexema << endl;
                        ungetc(c, stdin);   // regresa el delimitador
                        break;
                case 102: cout << "Operador " << c << endl; break;
                case 105: cout << "Delimitador (\n"; break;
                case 106: cout << "Delimitador )\n"; break;
                case 107: return 0;   // termina ejecución
                case 200: cout << "ERROR! palabra ilegal " << lexema << endl;
                        ungetc(c, stdin);   // regresa caracter válido
                        break;
                }
                lexema = "";
                edo = 0; // regresa al inicio
        }
}
```