
# IC-5701
# Compiladores e Intérpretes

Profesor:
Ing. Allan Rodríguez Dávila, MGP

# Agenda

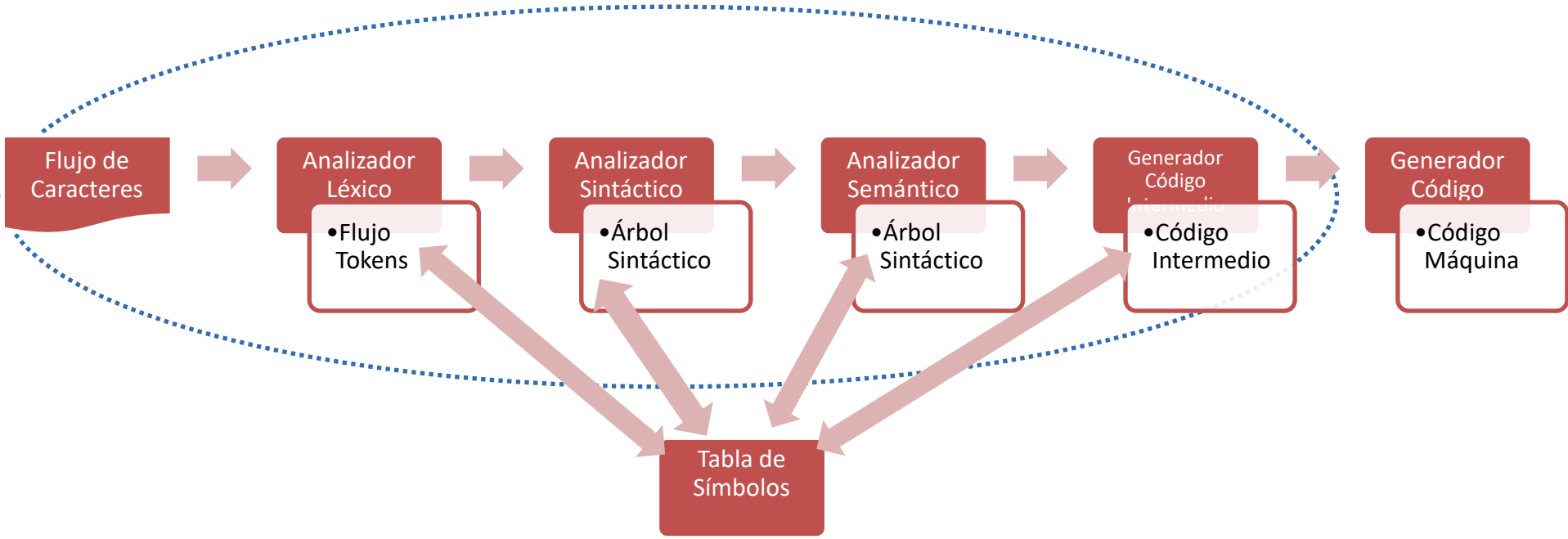
TEC

- Código de Tres Direcciones
- Tipos y Declaraciones
- Traducción de Expresiones
- Comprobación de Tipos
- Flujo de Control
- Parcheo de Retroceso

# Generación Código Intermedio

- El front-end de un compilador analiza el código fuente y crea una representación intermedia.
- Los detalles del código fuente deben ir en el front-end, lo relativo al código máquina debe ir en el back-end.

# Compilación



# Código de tres direcciones

- Secuencia de instrucciones con tres operandos por instrucción.
- Máximo un operador del lado derecho
- No se permiten expresiones acumuladas
- Direcciones e instrucciones

TEC

# Código de tres direcciones

$$x + y * z$$

$$t_1 = y * z$$

$$t_2 = x + t_1$$

TEC

# Instrucciones

- Asignación de operador binario
  - `x = y op z`
- Asignación de operador unario
  - `x = op y`
- Copia
  - `x = y`

TEC

# Instrucciones

- Salto incondicional
  - `goto L`
- Salto condicional
  - `if x goto L e ifFalse x goto L`
- Procedimientos
  - `param x1`
  - `param x2`
  - `call p, n`

TEC

# Instrucciones

- Copia indexadas
  - `x = y[i]`
  - `x[i] = y`
- Asignaciones de direcciones
  - `x = &y`
  - `x = * y`
  - `*x = y`

TEC

# Cuádruplos

```
a = b * - c + b * - c;
```

```
t1 = menos c
t2 = b * t1
t3 = menos c
t4 = b * t3
t5 = t2 + t4
a = t5
```

| | op | arg1 | arg2 | result |
|---|---|---|---|---|
| 0 | menos | c | | t1 |
| 1 | * | b | t1 | t2 |
| 2 | menos | c | | t3 |
| 3 | * | b | t3 | t4 |
| 4 | + | t2 | t4 | t5 |
| 5 | = | t5 | | a |

TEC

# Tripletas

```
a = b * – c + b * - c;
```

```
t1 = menos c
t2 = b * t1
t3 = menos c
t4 = b * t3
t5 = t2 + t4
a = t5
```

| | op | arg1 | arg2 |
|---|---|---|---|
| 0 | menos | c | |
| 1 | * | b | (0) |
| 2 | menos | c | |
| 3 | * | b | (2) |
| 4 | + | (1) | (3) |
| 5 | = | (4) | |

TEC

# Ejercicio

```
while (j < n)
{
      k = k + j * 2;
      m = j * 2;
      j++;
}
```

```
1)  t1 = j
2)  t2 = n
3)  t3 = t1 < t2
4)  if(t3) goto (6)
5)  goto (14)
6)  t4 = k
7)  t5 = t1 * 2
8)  t6 = t4 + t5
9)  k = t6
10) m = t5
11) t7 = t1 + 1
12) j = t7
13) goto (1)
14)
```

TEC

# Ejemplo

```
main(){
      int p; int a, b;
     p = dot_prod(a,b);
}
```

TEC

# Ejemplo

```
main(){
      int p; int a, b;
     p = dot_prod(a,b);
}
```

```
_func_begin_main:
p = 0
a = 0
b = 0
t1 = a
t2 = b
param t1
param t2
t3 = call dot_prod, 2
p = t3
jump back
_func_end:
```

TEC

```
int main(){
       int p; int b;
       read(b);
      if (b >= 10){
              if(p>=10){
                     b=11;
              }
      } else{
              p = b;
      }
      return p;
}
int potencia(int a, int b){
       return b^a;
}
```

# Portafolio #4

- Para cada uno de los siguientes fragmentos de código genere el código de tres direcciones

```
int funExc(char ar1, int sest, int x){
        int prod = 0;
        for x in range(i+10) {
                prod = ar1 + sest * x;
}
        return prod;
}
```

TEC

# Portafolio #4

- Genere el código de tres direcciones

```
switch (id+56*ert) {
      case 10:
            id = 11;
      case 11:
            id = 12;
            break;
      default:
            id = 12;
}
```

TEC

# Ejemplo

```
int fact(int n){
      if (n==0)
            return 1;
      else
            return (n*fact(n-1));
}
```

```
 func begin fact
/*crea espacio en pila
para cargar param[0] en
n*/
     n = param[0]
     t1 = n
     t2 = t1==0
     if t2 goto L1
     t3 = n
     t4 = t1-1
     param t4
     t5 = call fact, 1
     t6 = t4 * t5
     return t6
L1:  return 1
     func end
```

TEC

# Portafolio #4

```
int func(int n, float m, char c){
      print(n);
      print(m);
}
```