

# **INE5318**

## **CONSTRUÇÃO DE COMPILADORES**

### **AULA 6: GERAÇÃO DE CÓDIGO INTERMEDIÁRIO**

Ricardo Azambuja Silveira  
INE-CTC-UFSC  
E-Mail: [silveira@inf.ufsc.br](mailto:silveira@inf.ufsc.br)  
URL: [www.inf.ufsc.br/~silveira](http://www.inf.ufsc.br/~silveira)

---

---

# Geração de Código Intermediário

- A geração de código intermediário é a transformação da árvore de derivação em um fragmento de código, que pode ou não ser o código objeto final.



# Vantagens

- Vantagens da tradução de código fonte para objeto em mais de um passo:
  - Otimização de código intermediário de modo a obter código objeto mais eficiente.
  - Simplificação da implementação do compilador, resolvendo gradativamente as dificuldades da tradução.
  - Torna possível a tradução de código intermediário para diversas máquinas.



# Desvantagens

- O compilador requer um passo a mais para a tradução, tornando o processo um pouco mais lento.
- A grande diferença entre o código intermediário e o código objeto final é que o intermediário não especifica detalhes da máquina alvo, tais como quais registradores serão usados, quais endereços de memória serão referenciados etc.



# Linguagens Intermediárias

- Os vários tipos de código intermediário fazem parte de uma das seguintes categorias:
  - Representações gráficas: árvores e grafos de sintaxe;
  - Notação pós-fixada ou pré-fixada;
  - Código de três-endereços: quádruplas ou triplas.



# Árvore e Grafos de Sintaxe

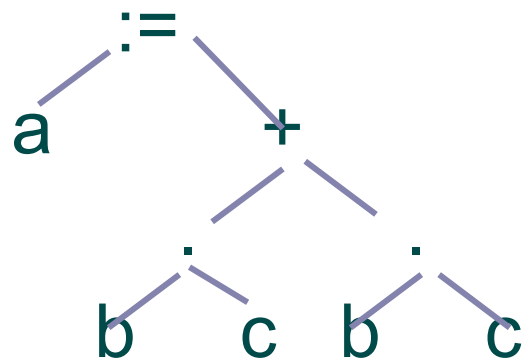
- Uma árvore de sintaxe é uma forma condensada de árvore de derivação na qual somente os operandos da linguagem aparecem como folhas;
- Os operadores constituem nós interiores da árvore.
- Na árvore de sintaxe, as cadeias de produções simples são eliminadas (por ex.: A B e B C fica A C).



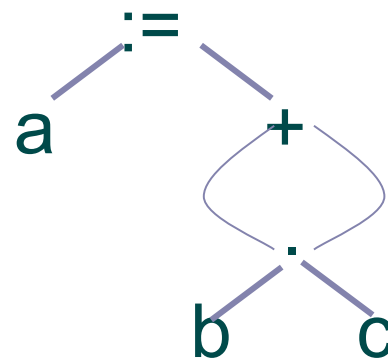
# Exemplo

Seja a equação:  $a := (b * c) + (b * c)$

Árvore de Sintaxe



Grafo de Sintaxe



No grafo de sintaxe, se já existe um nó rotulado este é “re-aproveitado”

# Grafo de Sintaxe

- Antes de construir um novo nó com rótulo *op* e ponteiros *pt1* e *pt2* para subárvores que representam subexpressões, uma função *geranó* verifica se já existe algum nó com rótulo *op* e ponteiros que apontem para árvores idênticas às apontadas por *pt1* e *pt2*.
  - Caso positivo, a função apenas retorna um ponteiro para o nó previamente construído.
  - Uma função *gerafolha* age de forma similar.
- 
-



# Notação Pós-fixa e Pré-fixa

Infixa	Pós-fixa	Pré-fixa
$(a+b)^*c$	$ab+c^*$	$*+abc$
$a^*(b+c)$	$abc+^*$	$*a+bc$
$a+b^*c$	$abc^*+$	$+a^*bc$

# Tradução

- Exemplo de tradução para gerar representações pós-fixadas:

$E \rightarrow E1 + T \{ E.cod := E1.cod || T.cod || "+" \}$

$E \rightarrow T \{ E.cod := T.cod \}$

$T \rightarrow T1 * F \{ T.cod := T1.cod || T.cod || "*" \}$

$T \rightarrow F \{ T.cod := F.cod \}$

$F \rightarrow id \{ F.cod := id.nome \}$



# Código de Três-endereços

- No código intermediário de três-endereços, cada instrução faz referência, no máximo, a três variáveis (endereços de memória).



# Código de Três-endereços

- As instruções dessa linguagem intermediária são as seguintes:

$A := B \text{ op } C$

$A := \text{op } B$

$A := B$

goto L

if A oprel B goto L

Onde A, B e C representam endereços de variáveis, op representa operador (binário ou unário), oprel representa operador relacional e L representa o rótulo de uma instrução intermediária.

# Código de Três-endereços

- Um código de três-endereços pode ser representado por quádruplas ou triplas.
  - As quádruplas são constituídas de quatro campos: um operador, dois operandos e o resultado.
  - As triplas são constituídas de três campos: um operador e dois operandos.
- 
-

# Quádruplas

Exemplo para a equação:  $A := B * (-C + D)$

	Oper	arg1	arg2	Result
(0)	-u	C		T1
(1)	+	T1	D	T2
(2)	*	B	T2	T3
(3)	:=	T3		A

# Triplas

Exemplo para a equação:  $A := B * (-C + D)$

	oper	arg1	arg2
(0)	-u	C	
(1)	+	(0)	D
(2)	*	B	(1)
(3)	:=	A	(2)

Nesse caso não são explicitadas variáveis temporárias.

## Exemplo

- Código de três-endereços para o comando

$A := X + Y * Z$

$T1 := Y * Z$

$T2 := X + T1$

$A := T2$

Onde T1 e T2 são variáveis temporárias.





# Exercícios

1. Dado os comandos:

$$x := (a + b) * c - (a + b) / d$$
$$\text{Area} := (\text{Base} * \text{Altura}) / 2$$

Traduzir para:

1. Árvore de sintaxe
2. Grafo de sintaxe
3. Derivar o código intermediário pós-fixado
4. Derivar o código intermediário de três-endereços.

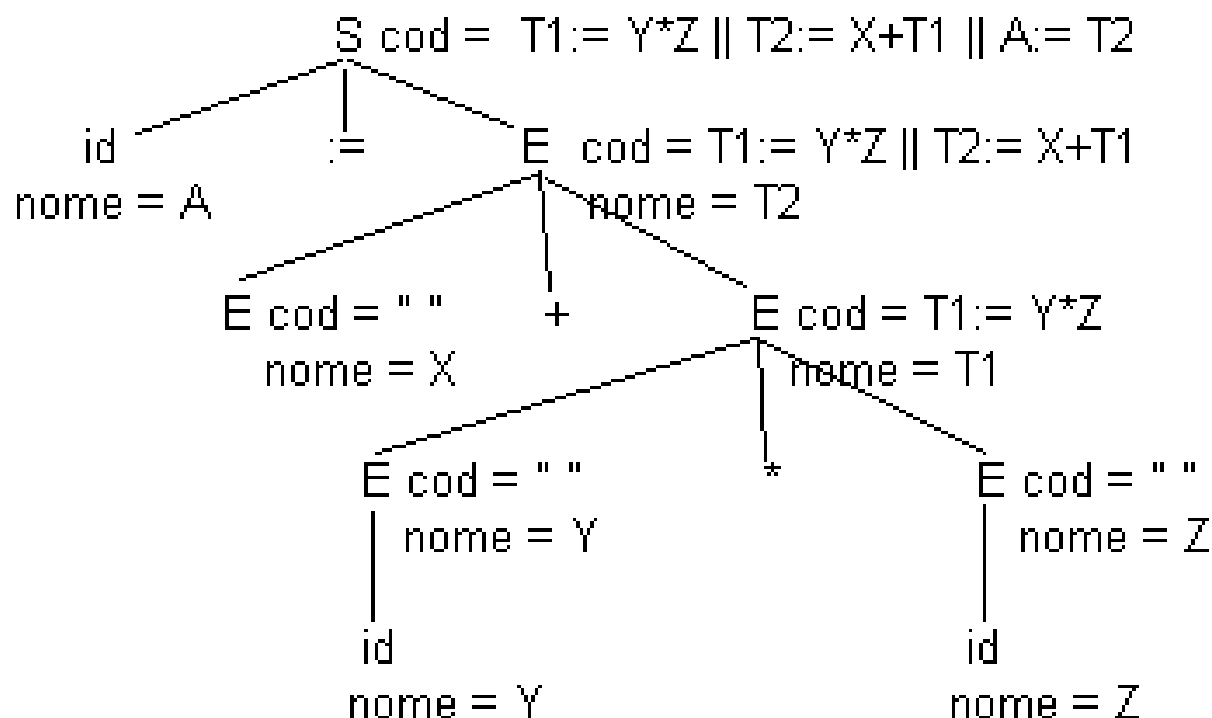
# Exemplo

- Esquema de tradução para gerar código de 3 endereços:
  - A função *geratemp()*, gera um nome de variável temporária
  - A estrutura *E.nome*, armazena o nome de uma variável ou temporária
  - A função *geracod()*, gera uma string (texto) correspondente a uma instrução de código intermediário a partir dos argumentos entre parênteses.
  - O atributo *cod* *S* contém o código final gerado para o comando.
  - O atributo *cod* *E* contém o código para a expressão correspondente.

```
S --> id := E    {S.cod=E.cod || geracod(id.nome ":=" E.nome)}
E --> E1 + E2    {E.nome = geratemp; E.cod = E1.cod || E2.cod||
                  geracod(E.nome ":=" E1.nome "+" E2.nome)}
E --> E1 * E2    {E.nome = geratemp; E.cod = E1.cod || E2.cod||
                  geracod(E.nome ":=" E1.nome "*" E2.nome)}
E --> (E1)       {E.nome = E1.nome; E.cod = E1.cod}
E --> id         {E.nome = id.nome; E.cod = " "}
```

# Exemplo

- Árvore de derivação para o comando:  $A := X + Y * Z$



# Reutilização de Temporárias

- Após uma variável temporária ser referenciada (aparecer no lado direito de uma atribuição), ela pode ser descartada.
  - Para reutilização de temporárias, usa-se um contador C, com valor inicial 1.
  - Sempre que uma nova temporária é gerada, usa-se TC no código e incrementa-se C de 1.
  - Sempre que um temporário é usado como operando decrementa-se C de 1.
- 
-

# Exemplo

- Utilização de temporárias para o comando de atribuição:
  - $X := A * B + C * D - E * F$
  - $T1 := A * B$  (incrementa C de 1)
  - $T2 := C * D$  (incrementa C de 1)
  - $T1 := T1 + T2$  (decrementa C de 1, 2 vezes)
  - $T2 := E * F$  (incrementa C de 1)
  - $T1 := T1 - T2$  (decrementa C de 1, 2 vezes)
  - $X := T1$  (Fim)

# Exercício

- Desenhar a árvore de derivação para as expressões:
  - $Y = 3x^2 + 2x + 5$
  - $\text{Saldo} = \text{Credito} - (0,5 * \text{Debito})$
- Descrever a utilização de temporárias para as equações do exercício anterior

# Expressões Lógicas

- Representação Numérica
  - Na representação numérica de expressões lógicas deve-se codificar numericamente as constantes *true* e *false* (por ex.: 1 e 0) e avaliar as expressões lógicas de forma numérica, ficando o resultado da avaliação numa variável temporária.



# Exemplo

- Código para avaliar expressões lógicas de forma numérica:
  - Supondo que o código gerado seja armazenado a partir de uma quádrupla número 100, o comando de atribuição  $X := A \text{ or } B \text{ and not } C$  (onde A, B e C são variáveis lógicas) seria traduzido para:
    - 1  $T1 := \text{not } C$
    - 2  $T2 := A \text{ and } T1$
    - 3  $T3 := A \text{ or } T2$
    - 4  $X := T3$



# Exemplo

- A expressão  $A < B$  onde  $A$  e  $B$  são variáveis numéricas, pode ser traduzida como:

100 if  $A < B$  goto 103

101  $T1 := 0$

102 goto 104

103  $T1 := 1$

104

Nesse caso o valor da expressão fica na última temporária gerada no processo de avaliação.

# Esquema de Tradução

- O esquema de tradução a seguir gera código para expressões lógicas, supondo que as instruções geradas são armazenadas num vetor de quádruplas.
- A função *geracod()* utiliza a variável *proxq* para indicar o índice da próxima quádrupla disponível
- Após gravar uma quádrupla, a função *geracod* incrementa *proxq*.

E --> E1 or E2	{E.nome = geratemp; geracod(E.nome ":=" E1.nome "or" E2.nome)}
E --> E1 and E2	{E.nome = geratemp; geracod(E.nome ":=" E1.nome "and" E2.nome)}
E --> not E1	{E.nome = geratemp; E.cod = E1.cod    geracod (E.nome ":=" E1.nome "and" E2.nome)}
E --> ( E1 )	{E.nome = E1.nome}
E --> id1 oprel id2	{E.nome = geratemp; geracod("if" id1.nome oprel.simb id2.nome "goto" proxq+3) geracod(E.nome ":= 0"); geracod ("goto" proxq+2); geracod (E.nome ":= 1")}
E --> true	{E.nome = geratemp; geracod (E.nome ":= 1")}
E --> false	{E.nome = geratemp; geracod (E.nome ":= 0")}

# Exemplo

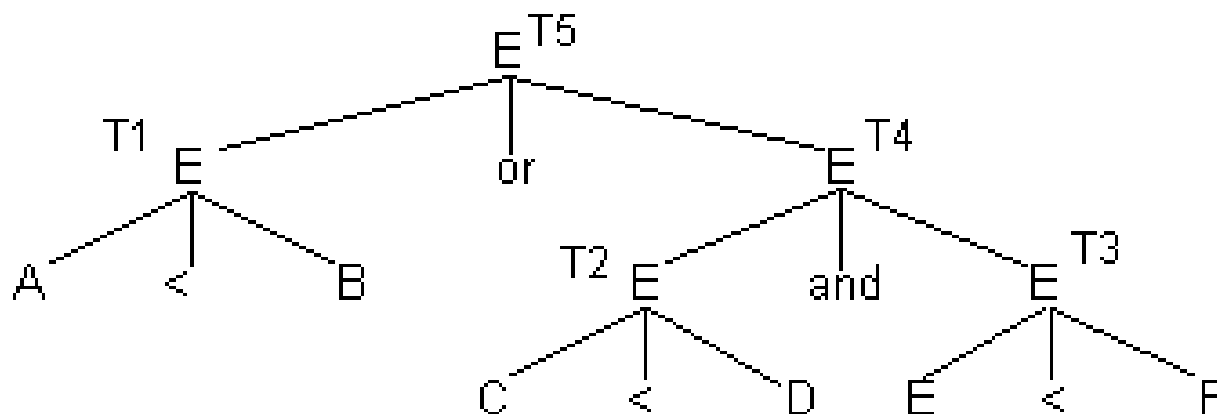


Código gerado para a expressão:  $A < B$  or  $C < D$  and  $E < F$

```

100 if A<B goto 103
101 T1:= 0
102 goto 104
103 T1:= 1
104 if C<D goto 107
105 T2:= 0
106 goto 108
107 T2:= 1
108 if E<F goto 111
109 T3 := 0
110 goto 112
111 T3:= 1
112 T4:= T2 and T3
113 T5:= T1 or T4

```



# Exemplo

- Geração de código para o comando *while*:

```
S  while E do S1  { S.inicio = gerarotulo; S.prox= gerarotulo;  
                      S.cod = geracod (S.inicio ":") || E.cod ||  
                      geracod ("if" E.nome "=0 goto" S.prox) ||  
                      geracod ("goto" S.inicio)||  
                      geracod (S.prox ":") }
```

Nesse caso o código gerado é uma string de instruções e rótulos que fica armazenado no atributo *cod* de *S*. Os atributos *inicio* e *prox* identificam, respectivamente o início da iteração e o início do comando seguinte ao *while*.

---

---