

– INF01147 –
Compiladores

Geração de Código Intermediário
Declarações, Escopos e Atribuição

Prof. Lucas M. Schnorr
– Universidade Federal do Rio Grande do Sul –



Plano da Aula de Hoje

- ▶ Revisão
- ▶ Geração de IR
 - ▶ Declaração com Escopo Simples
 - ▶ Declaração com Escopo Aninhado
 - ▶ Comandos de Atribuição

IR Lineares – Código de Três Endereços (TAC)

- Forma da maioria das operações possíveis

$$i \leftarrow j \text{ op } k$$

- Algumas operações não precisam de todos os argumentos
- Exemplo para a expressão $a - 2 * b$

$$t_1 \leftarrow 2$$

$$t_2 \leftarrow b$$

$$t_3 \leftarrow t_1 * t_2$$

$$t_4 \leftarrow a$$

$$t_5 \leftarrow t_4 - t_3$$

- Vantagens
 - Razoavelmente compacto, **sem operações destrutivas**
 - Importância da escolha do espaço de nomes
- Nível de abstração é controlável

Geração – Expressões Esquema de Tradução

$S \rightarrow$	$\text{nome} = E;$	$\{ S.\text{codigo} = E.\text{codigo}; \parallel$ $\text{gera}(\text{nome.local} = E.\text{local}); \}$
$E \rightarrow$	$E_1 + E_2$	$\{ E.\text{local} = \text{temp}();$ $E.\text{codigo} = E_1.\text{codigo} \parallel E_2.\text{codigo} \parallel$ $\text{gera}(E.\text{local} = E_1.\text{local} + E_2.\text{local}); \}$
$E \rightarrow$	$E_1 * E_2$	$\{ E.\text{local} = \text{temp}();$ $E.\text{codigo} = E_1.\text{codigo} \parallel E_2.\text{codigo} \parallel$ $\text{gera}(E.\text{local} = E_1.\text{local} * E_2.\text{local}); \}$
$E \rightarrow$	(E_1)	$\{ E.\text{local} = E_1.\text{local};$ $E.\text{codigo} = E_1.\text{codigo} \}$
$E \rightarrow$	id	$\{ E.\text{local} = \text{id.lexval};$ $E.\text{codigo} = ""; \}$

- ▶ Testar com o exemplo $\text{var} = x + y * z;$
 - ▶ Gerar árvore de derivação ou **AST**
 - ▶ Executar as ações semânticas de tradução

Geração de TAC

Declaração de Escopo Simples

Geração – Declarações (escopo simples)

- ▶ Escopo

- ▶ Variáveis da mesma função pertencem ao mesmo “grupo”
- ▶ Associa **posições de memória a nomes locais**
- ▶ A endereço da variável é um **deslocamento**

- ▶ Funcionamento

- ▶ Quando o analisador vê uma declaração
 - ▶ Atualiza o deslocamento na tabela de símbolos à
 - Base do segmento de dados para globais
 - Base da pilha de dados locais no registro de ativação

- ▶ Peculiaridades da máquina alvo

- ▶ Leva em consideração o tamanho da palavra, alinhamento

Geração – Declarações (escopo simples)

- ▶ Exemplo de código fonte típico em linguagem C

```
foo () {  
    int i, j;  
    float x;  
    double t[10];  
    x = i * 5.2 - (j + x);  
}
```

- ▶ Código TAC de “baixo nível”

```
t0 = (fp+0) * 5.2  
t1 = (fp+4) + (fp*8)  
t2 = t0 - t1  
(fp+8) = t2
```

- ▶ fp contém o endereço base associado ao escopo local
- ▶ Deslocamento segue a ordem de declaração
 - ▶ i (fp+0), j (fp+4), x (fp+8)

Geração – Declarações (escopo simples)

- ▶ Usando a tradução dirigida pela sintaxe
 - ▶ Inserção na tabela de símbolos é feita na análise sintática
 - ▶ Atributos importantes
 - ▶ Nome (lexema associado ao identificador)
 - ▶ Tipo (expressão de tipo)
 - ▶ Tamanho (da representação)
 - ▶ Deslocamento dentro do escopo
- ▶ Função auxiliar
`declara(nome, tipo, tamanho, deslocamento)`

Geração – Declarações (escopo simples)

- ▶ Tamanhos de alguns tipos básicos
 - ▶ int é 4 bytes
 - ▶ float é 4 bytes
 - ▶ vetor é a quantidade de elementos vezes tamanho do tipo
 - ▶ ponteiro é 4 bytes
- ▶ Exemplo de gramática para declarações

P → MD

M → ϵ

D → D ; D

D → id : T

T → inteiro

T → flutuante

T → vetor[num] of T_1

T → ponteiro T_1

Geração – Declarações

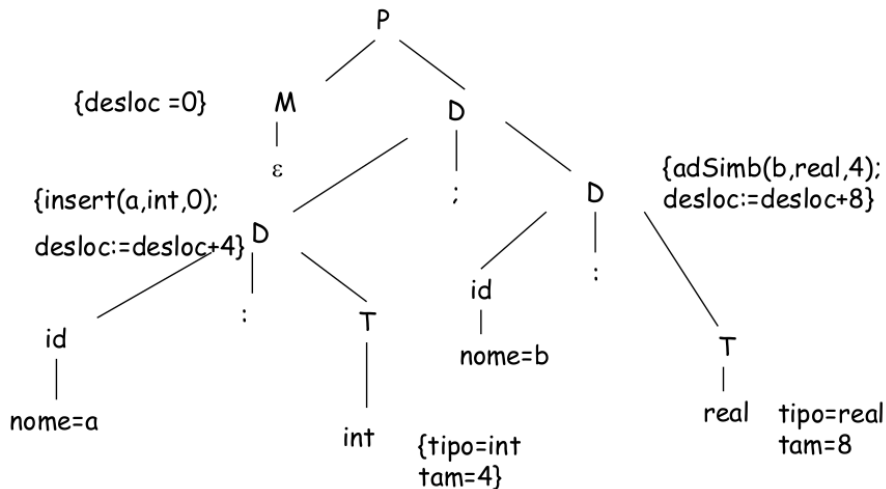
Esquema de Tradução

P	→	MD	
M	→	ε	{ desloc = 0; }
D	→	D ; D	
D	→	id : T	{ declara(id.nome, T.tipo, T.tamanho, desloc); desloc = desloc + T.tamanho; }
T	→	inteiro	{ T.tipo = int; T.tamanho = 4; }
T	→	flutuante	{ T.tipo = float; T.tamanho = 8; }
T	→	v[num] of T ₁	{ T.tipo = vetor(num.val, T ₁ .tipo) T.tamanho = num.val * T ₁ .tamanho }
T	→	ponteiro T ₁	{ T.tipo = ponteiro(T ₁ .tipo); T.tamanho = 4; }

- Testar com exemplo $a : \text{inteiro}; b : \text{flutuante};$

Geração – Declarações

Árvore de Derivação do Exemplo



Geração de TAC

Declaração de Escopo Aninhado

Geração – Declarações (escopo aninhado)

- ▶ Tipos de escopo (Relembrando MLP)
 - ▶ Escopo **Estático**
Vinculações determinadas pela organização do programa
→ Tempo de Compilação
 - ▶ Escopo **Dinâmico**
Vinculações determinadas pela sequência de ativação
→ Tempo de Execução
- ▶ Regras de escopo mais complexas
- ▶ Cada subprograma possui seu escopo
- ▶ Blocos com escopo anônimo
- ▶ **Pergunta:** Como implementar isso em um compilador?
 - ▶ Uma tabela de símbolos por escopo

Geração – Declarações (escopo aninhado)

- ▶ Tabelas de símbolos são organizadas hierarquicamente
- ▶ Adotar uma nova gramática

$P \rightarrow MD$

$M \rightarrow \epsilon$

$D \rightarrow D ; D$

$D \rightarrow id : T$

$D \rightarrow \text{proc } id; N D; S; \text{end}$

$N \rightarrow \epsilon$

Geração – Declarações Estruturas de Dados

- ▶ Árvore de tabelas de símbolos
- ▶ Pilha pDesloc – deslocamento corrente
- ▶ Pilha pTabela – tabela de símbolos corrente

Geração – Declarações

Funções auxiliares

- ▶ Métodos que empilham e desempilham
- ▶ `geraTab(ref_tabela_pai)`
 - ▶ Cria nova tabela de símbolos
- ▶ `declara(tabela, nome, tipo, tamanho, deslocamento)`
- ▶ `declaraFunc(tabela, nome, ptr_tab)`
 - ▶ Declara um novo subprograma
 - ▶ Liga ele a tabela apontada por `ptr_tab`
- ▶ `defTam(tabela, valor)`
 - ▶ Registra o tamanho da área de dados

Geração – Declarações

Esquema de Tradução

$P \rightarrow MD$

$M \rightarrow \epsilon$ { tabela = geraTab(NULL);
 empilha(tabela, pTabela);
 empilha(0, pDesloc); }

$D \rightarrow D ; D$

$D \rightarrow id : T$

$D \rightarrow \text{proc id; } N \ D; S; \text{end}$

$N \rightarrow \epsilon$

Geração – Declarações

Esquema de Tradução

P	→	MD	{ defTam(topo(pTabela), topo(pDesloc)); desempilha(pTabela); desempilha(pDesloc); }
M	→	ε	{ tabela = geraTab(NULL); empilha(tabela, pTabela); empilha(0, pDesloc); }
D	→	D ; D	
D	→	id : T	
D	→	proc id; N D; S; end	
N	→	ε	

Geração – Declarações Esquema de Tradução

$$\begin{aligned} P &\rightarrow MD \\ M &\rightarrow \epsilon \\ D &\rightarrow D ; D \\ D &\rightarrow id : T \quad f() \\ D &\rightarrow \text{proc id; N D; S; end} \\ N &\rightarrow \epsilon \end{aligned}$$

```
void f() {  
    declara(topo(pTabela),  
            id.nome,  
            T.tipo,  
            T.tamanho,  
            topo(pDesloc));  
    topo(pDesloc) = topo(pDesloc) + T.tamanho;  
}
```

Geração – Declarações Esquema de Tradução

$$\begin{array}{ll} P & \rightarrow MD \\ M & \rightarrow \epsilon \\ D & \rightarrow D ; D \\ D & \rightarrow id : T \\ D & \rightarrow \text{proc } id; N D; S; \text{end} \\ N & \rightarrow \epsilon \end{array} \quad h()$$

```
void h() {  
    nova = geraTab(topo(pTabela));  
    empilha (nova, pTabela);  
    empilha (0, pDesloc);  
}
```

Geração – Declarações Esquema de Tradução

$P \rightarrow MD$

$M \rightarrow \epsilon$

$D \rightarrow D ; D$

$D \rightarrow id : T$

$D \rightarrow \text{proc } id; N D; S; \text{end } g()$

$N \rightarrow \epsilon$

```
void g() {  
    tabela = topo(pTabela);  
    defTam(t, topo(pDesloc));  
    desempilha(pTabela);  
    desempilha(pDesloc);  
  
    declaraFunc(topo(pTabela), id.nome, tabela);  
  
    topo(pDesloc) = topo(pDesloc) + 4;  
}
```

Geração – Declarações

Esquema de Tradução

P \rightarrow MD

M $\rightarrow \epsilon$

D \rightarrow D ; D

D \rightarrow id : T

D \rightarrow proc id; N D; S; end

N $\rightarrow \epsilon$

T \rightarrow inteiro { T.tipo = int; T.tamanho = 4; }

T \rightarrow flutuante { T.tipo = float; T.tamanho = 8; }

T \rightarrow v[num] of T_1 { T.tipo = vetor(num.val, T_1 .tipo)
T.tamanho = num.val * T_1 .tamanho }

T \rightarrow ponteiro T_1 { T.tipo = ponteiro(T_1 .tipo);
T.tamanho = 4; }

Geração – Declarações Exemplo

```
a : flutuante;  
b : inteiro;  
proc p1;  
  c : real;  
end p1;  
proc p2;  
  d : v[5] de inteiro;  
  proc p3;  
    e, f : flutuante;  
  end p3;  
end p2;
```

Geração – Comando de Atribuição

```
S → nome = E;  {{ p = procuraSimbolo(nome.lexval); }  
                  if (p) { S.codigo = E.codigo; ||  
                        gera(nome.local = E.local); }  
                  else erro; }
```

```
E → E1 + E2  { E.local = temp();  
                  E.codigo = E1.codigo || E2.codigo ||  
                  gera(E.local = E1.local + E2.local); }
```

```
E → E1 * E2  { E.local = temp();  
                  E.codigo = E1.codigo || E2.codigo ||  
                  gera(E.local = E1.local * E2.local); }
```

```
E → ( E1 )    { E.local = E1.local;  
                  E.codigo = E1.codigo }
```

```
E → id         { p = procuraSimbolo(id.lexval);  
                  if (p) E.local = p; else erro;  
                  E.codigo = ""; }
```

Conclusão

- ▶ Leituras Recomendadas
 - ▶ Série Didática
 - ▶ Seção 5.2 e 5.3
 - ▶ Livro do Keith
 - ▶ Capítulo 5.5

- ▶ Próxima Aula
 - ▶ Apresentação da Etapa 4
 - ▶ Sala 101 do Prédio 67 (Turma A)
 - ▶ Sala 101 do Prédio 67 (Turma B)