Compiladores
Geração de Código

Prof. Dr. Luiz Eduardo G. Martins

(adaptado por Profa Dra Ana Carolina Lorena)

UNIFESP

Geração de Código Intermediário

- A geração de código intermediário é a 1ª etapa da fase de síntese do compilador
- O código intermediário é independente da máquina alvo
- O código intermediário é uma “linearização” da árvore sintática

Geração de Código Intermediário

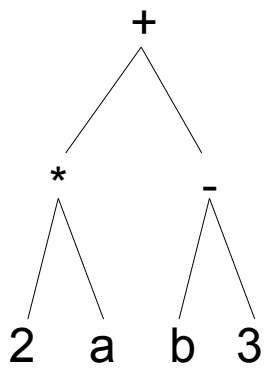
- O código intermediário é preferível, como representação intermediária, do que a árvore sintática
  - Se aproxima mais da representação do código de montagem *(assembly)*
    - Conjunto de instruções do processador (ADD, SUB, CMP, CALL, LOAD, STORE, MOVE, AND, OR, JMP...)
- Representações intermediárias mais conhecidas:
  - Código de três endereços
  - P-código

Geração de Código Intermediário

- Código de três endereços
  - Possibilita especificar instruções com no máximo três operandos (variáveis ou constantes)
    - Base: instrução mais básica é projetada para representar avaliação de operações aritméticas
    - x = y op z
    - Em geral, cada um dos nomes representa um endereço de memória

Geração de Código Intermediário

- Código de três endereços
  - Exemplo: 2*a + (b-3)



Linearizando
da esquerda para a direita

```
t1 = 2 * a
t2 = b – 3
t3 = t1 + t2
```

Os temporários são nós internos da árvore sintática
Em geral são atribuídos a registradores,
mas podem ser também mantidos na RAM

Geração de Código Intermediário

- Código de três endereços
  - Implementação: cada instrução é um registro com campos e monta-se uma lista ligada desses registros

Geração de Código Intermediário

- Código de três endereços
  - Tipos básicos de instruções:
    - Expressão com atribuição
    - Desvio
    - Invocação (e definição) de sub-rotinas
    - Acesso indexado

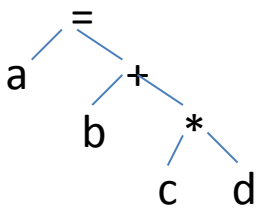
Geração de Código Intermediário

- Instruções de Atribuição
  - São aquelas em que o resultado de uma operação é armazenado na variável especificada à esquerda do operador de atribuição, denotado por =
    - Formas básicas de atribuição:

le = ld	ex: a = b
le = ld1 op ld2	ex: a = b + c
le = op ld	ex: a = -b

Geração de Código Intermediário

- Instruções de Atribuição
  - Expressões de atribuição mais complexas demandam a criação de variáveis temporárias, e a “quebra” em múltiplas instruções
  - Variáveis temporárias devem ter rótulos diferentes dos identificadores do programa fonte
    - Ex: a = b + c * d

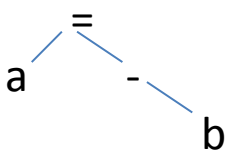


t1 = c * d

t2 = b + t1

a = t2

a = -b



t1 = -b

a = t1

Geração de Código Intermediário

- Instruções de Desvio
  - Podem assumir duas formas básicas:
    - Desvio incondicional, no formato:

      goto L

      onde *L* é um rótulo que identifica uma linha de código
    - Desvio condicional, no formato:

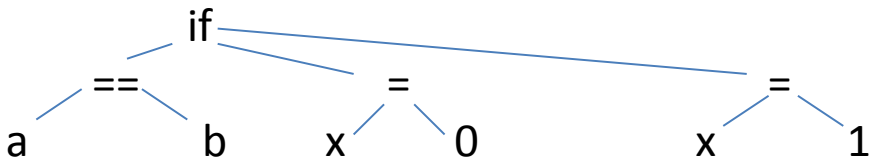
      if x opr y goto L

      Onde *opr* é um operador relacional e *L* é o rótulo da linha que deve ser executada se o resultado da condição for verdadeiro, caso contrário, a linha seguinte é executada

Geração de Código Intermediário

- Instruções de Desvio
  - Exemplo (em C-):

```
if  (a == b) x = 0;
else x = 1;
```

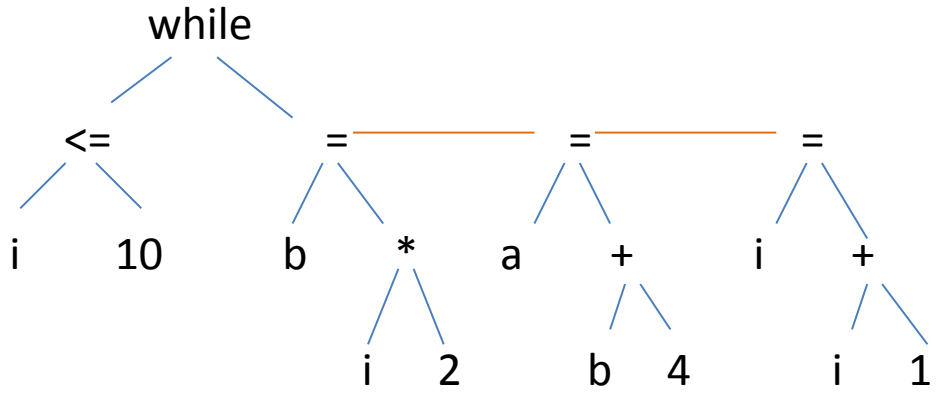



```
t1 = a == b              (avaliação da condição)
if_true t1  goto  L1     (salto para bloco "verdadeiro")
x = 1                    (bloco "falso")
goto  L2                 (salto para próxima instrução)
L1: x = 0                (bloco "verdadeiro")
L2:                      (próxima instrução...)
```

Geração de Código Intermediário

- Instruções de Desvio
  - Exemplo (em C-):

```
while (i <= 10)
{
   b = i * 2;
   a = b + 4;
   i = i + 1;
}
```




```
L1:     t1 = i > 10
        if_true t1 goto  L2
        b = i * 2
        a = b + 4
        i = i + 1
        goto  L1
L2:     (próxima instrução...)
```

Geração de Código Intermediário

- Sub-rotinas
  - É bastante dependente do ambiente de execução
  - Definição/declaração: cria nome, parâmetros e código (mas função não é executada)
  - Ativação/invocação: cria valores para parâmetros e efetua salto para código da sub-rotina, que é executada e retorna um valor

Geração de Código Intermediário

- Definição de sub-rotinas
  - Assume o seguinte formato

    L: (corpo da sub-rotina)

    return *v*

    onde L é o rótulo que identifica a sub-rotina e *v* é o valor de retorno da sub-rotina (se houver tal valor)

Geração de Código Intermediário

- Definição de sub-rotinas
  - Exemplo:

```
int f(int x, int y){
    return x + y + 1;
}
```

f: t1 = x + y

t2 = t1 + 1

return t2

Geração de Código Intermediário

- Invocação de sub-rotinas
  - Assume o seguinte formato

    param x1

    ...

    param xn

    call *f*, *n*

    onde *f* é o nome da função e *n* é a quantidade de parâmetros de entrada da função

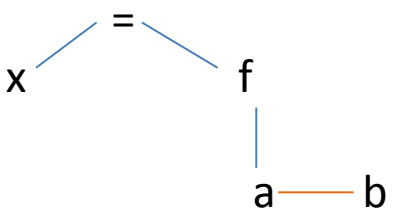
    call é o nome da instrução para ativação da função (código *assembly* normalmente usa esse nome)

    Obs: o parâmetro de retorno da função, se houver, deve ser associado a uma atribuição

Geração de Código Intermediário

- Invocação de sub-rotinas
  - Exemplo (em C-):

    x = f(a, b);



Código de três endereços:

param a

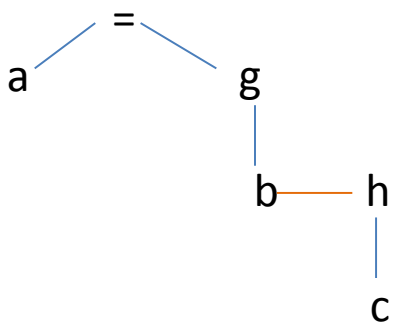
param b

x = call f, 2

Geração de Código Intermediário

- Invocação de sub-rotinas
  - Exemplo (em C-):

    a = g (b, h(c));



    Código de três endereços:

    param b

    param c

    t1 = call h, 1

    param t1

    a = call g, 2

    Obs: os parâmetros anteriores já consumidos, devem ser desconsiderados para a próxima ativação de sub-rotina

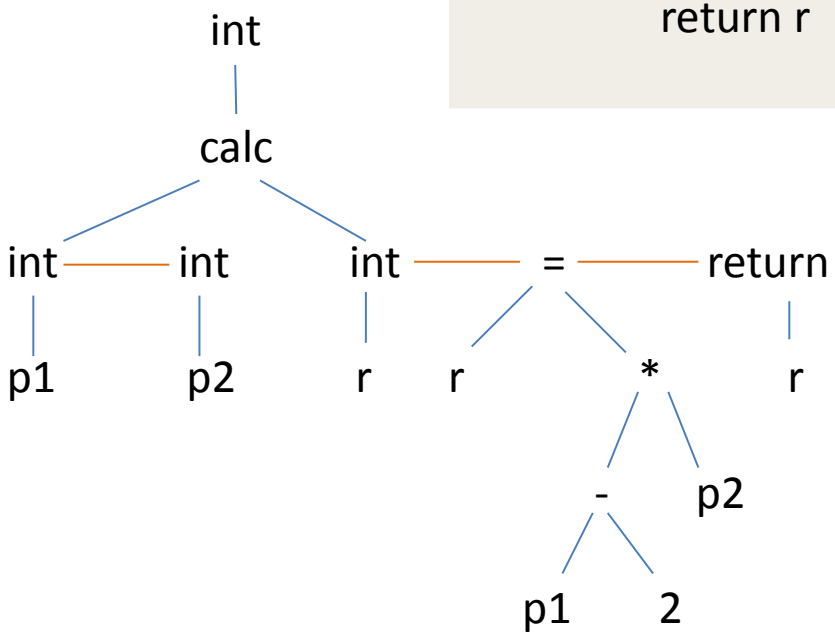
Geração de Código Intermediário

- Definição e invocação de sub-rotinas
  - Exemplo (em C-):

```
x = calc(a, b);              [ativação]
...

int calc(int p1, int p2)     [definição]
{
   int r;
   r = (p1 -2) * p2;
   return r;
}
```

```
param a
param b
x  =  call  calc, 2
...
calc:  t1 = p1 – 2
       r = t1 * p2
            return r
```



Geração de Código Intermediário

- Definição/ativação de sub-rotinas
  - Considere a GLC abaixo:

programa → *decl-lista exp*
decl-lista → *decl-lista decl* | ε
decl → **fn id** (*param-lista*) = *exp*
param-lista → *param-lista*, **id** | **id**
exp → *exp* + *exp* | *ativação* | **num** | **id**
ativação → **id** (*arg-lista*)
arg-lista → *arg-lista*, *exp* | *exp*

  - A entrada ao lado é válida ?

```
fn f(x)=2+x
fn g(x,y)=f(x)+y
g(3,4)
```

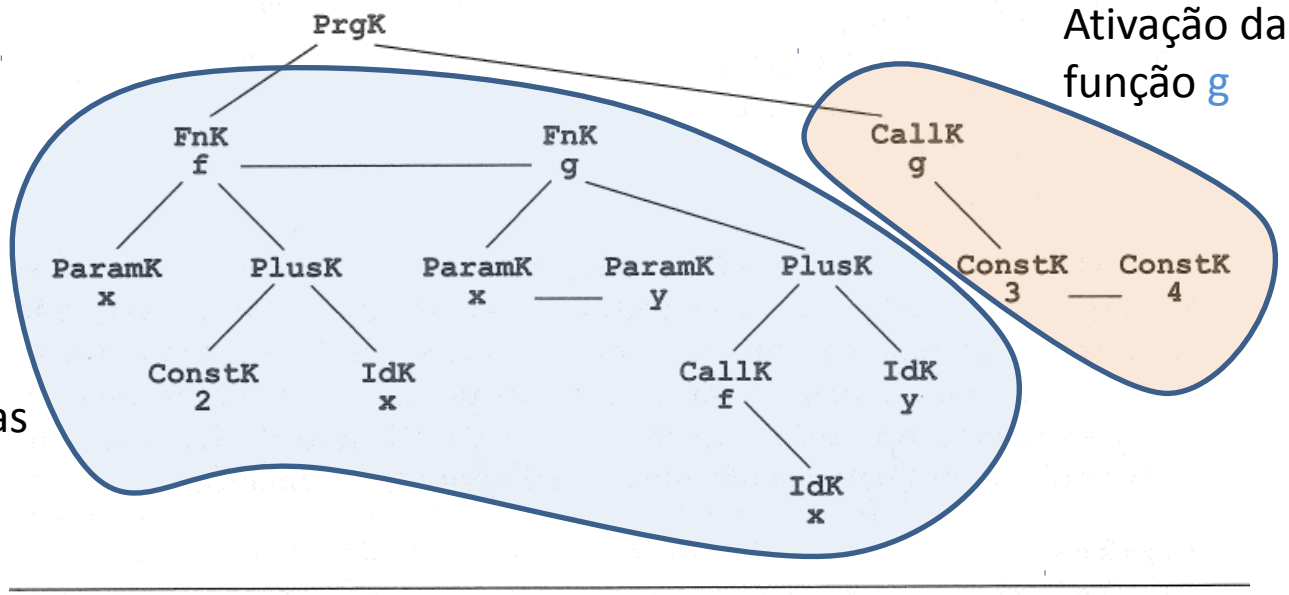
Geração de Código Intermediário

- Definição de sub-rotinas
  - Possível definição para o nó da árvore sintática para a GLC:

```
typedef enum
  {PrgK,FnK,ParamK,PlusK,CallK,ConstK,IdK}
  NodeKind;
typedef struct streenode
        { NodeKind kind;
          struct streenode *lchild,*rchild,
                           *sibling;
          char * name; /* usado com FnK,ParamK,
                          CallK,IdK */
          int val; /* usado com ConstK */
        } STreeNode;
typedef STreeNode *SyntaxTree;
```

Geração de Código Intermediário

- Definição de sub-rotinas
  - Árvore sintática abstrata



Ativação da função g

Definições das funções f e g

Figura 8.13 Árvore sintática abstrata para o exemplo do programa anterior.

```
fn f(x)=2+x
fn g(x,y)=f(x)+y
g(3,4)
```

Geração de Código Intermediário

- Definição de sub-rotinas
  - Código de três endereços

```
f:      t1 = 2 + x
        return t1
g:      param x
        t2 = call f, 1
        return t2 + y
init:   param 3
        param 4
        call g, 2
```

```
fn f(x)=2+x
fn g(x,y)=f(x)+y
g(3,4)
```

Geração de Código Intermediário

- Acesso indexado
  - A posição do item de informação acessado é definida a partir:
    - De um endereço base
    - E de um deslocamento (o índice)
      - O deslocamento depende do tipo de dado
      - Normalmente é calculado em bytes

Geração de Código Intermediário

- Acesso indexado
  - Assume o seguinte formato:

    t = i * n

    v[t]

    - onde

      t é a variável temporária que armazena o deslocamento calculado em bytes

      i é o índice do vetor

      n é o tamanho em bytes ocupado pelo tipo de dado

      v é a variável indexada

Geração de Código Intermediário

- Acesso indexado
  - Exemplo
    - Considere que uma variável do tipo inteiro ocupa 4 bytes

    y = x[i]

    Código de três endereços:

    t1 = i * 4

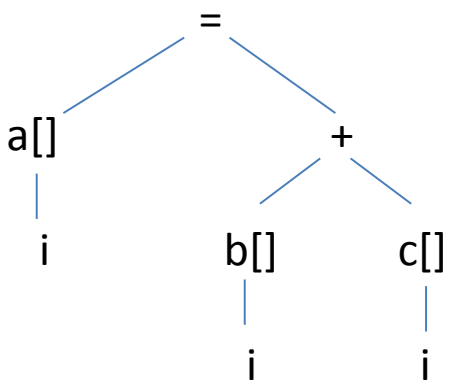
    y = x[t1]

Se o endereço base de x[] é 100, então o conteúdo acessível por x[t1] está no endereço 100 + (i * 4)
Exemplo:
Se i = 5, então o endereço de x[t1] é 120.

Geração de Código Intermediário

- Acesso indexado
  - Exemplo

    a[i] = b[i] + c[i]



Código de três endereços:

t1 = i * 4

t2 = b[t1]

t3 = c[t1]

t4 = t2 + t3

a[t1] = t4

Geração de Código Intermediário

- Orientação geral para implementar algoritmo para geração de código intermediário:
  - Percorrer árvore sintática em pré-ordem
  - Se tipo do nó for *void*, *int ou float* ignorar
  - Se tipo do nó for atribuição, operador aritmético, operador relacional ou ativação de função, e esse nó for raiz de subárvore, percorrer em ordem simétrica

Geração de Código Intermediários

- Bibliografia consultada

RICARTE, I. **Introdução à Compilação.** Rio de Janeiro: Editora Campus/Elsevier, 2008.

LOUDEN, K. C. **Compiladores: princípios e práticas.** São Paulo: Pioneira Thompson Learning, 2004 (cap 8).