

– INF01147 –
Compiladores

Geração de Código

Arranjos multidimensionais

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



Plano da Aula de Hoje

- ▶ Geração de IR – Arranjos Multidimensionais
 - ▶ Endereçamento
 - ▶ Esquema de Tradução

- ▶ Lançamento da Etapa 5

Considerações Iniciais

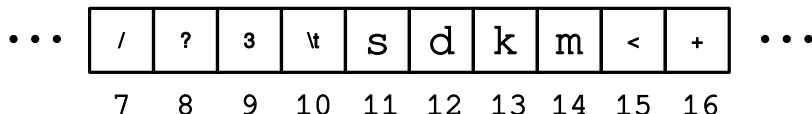
- ▶ Arranjos começam em 0 e terminam em $n - 1$
 - ▶ n é o tamanho do arranjo
- ▶ Qual o TAC para $A[i_1][i_2] = u$?
- ▶ Mais genérico: qual o TAC para $A[i_1][i_2] \dots [i_k] = u$?

Arranjos 1D

$A[i_1]$

Arranjos 1D vetores

- ▶ Memória pode ser vista como um vetor virtual infinito
- ▶ Endereçamento é um **mapeamento** (arranjo \rightarrow memória)
 - ▶ Nome alternativo: serialização



- ▶ Exemplo
 - ▶ Qual o endereço de memória $A[2]$?
 - ▶ Qual o conteúdo dessa posição?

Arranjos 1D vetores

matriz A, tamanho n

$A[i]$

$$e = \text{base} + i$$

- ▶ base: endereço relativo do arranjo na memória
- ▶ i: deslocamento do elemento em relação à base
- ▶ Deve-se levar em conta o tipo de dado
 - ▶ int é 4 bytes
 - ▶ float é 4 bytes
 - ▶ ponteiro é 4 bytes
- ▶ Tamanho w deve aparecer na equação

matriz A, tamanho n

$A[i]$

$$e = \text{base} + i * w$$

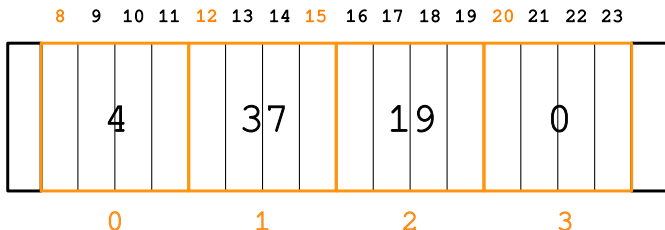
Arranjos 1D – Regra

matriz A, tamanho n

$A[i]$

$$e = \text{base} + i * w$$

- ▶ base: endereço relativo do arranjo na memória
- ▶ i: deslocamento do elemento em relação à base
- ▶ w: tamanho do tipo do dado
- ▶ Exemplo



Arranjos 2D

$A[i_1][i_2]$

Arranjos 2D matrizes, tabelas

- ▶ Matriz A, tamanho $n_1 * n_2$ (3x4)
- ▶ Na figura: qual o endereço de $A[1][2]$?

	$i_2=0$	$i_2=1$	$i_2=2$	$i_2=3$
$i_1=0$				
$i_1=1$				
$i_1=2$				

- ▶ Armazenamento por linhas
 - ▶ C, C++, Java
- ▶ Armazenamento por colunas
 - ▶ Fortran

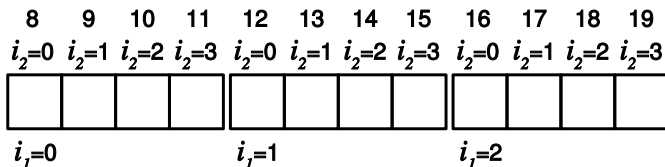
Arranjos 2D – Funcionamento matrizes, tabelas

- ▶ Equação de endereçamento

$$\begin{aligned} &\text{matriz } A, \text{ tamanho } n_1 * n_2 \\ &A[i_1][i_2] \\ &e = \text{base} + (i_1 * n_2 + i_2) * w \end{aligned}$$

- ▶ *base*: endereço relativo do arranjo na memória
- ▶ $i_1 * n_2$ seleciona a linha
- ▶ i_2 seleciona a coluna
- ▶ $(i_1 * n_2 + i_2)$ define o deslocamento
- ▶ w tamanho do dado

- ▶ Qual o endereço de memória de $A[1][2]$ no arranjo $3 * 4$?



Arranjos 2D – Regra

$$\begin{aligned} &\text{matriz } A, \text{ tamanho } n_1 * n_2 \\ &A[i_1][i_2] \\ e = &base + (i_1 * n_2 + i_2) * w \end{aligned}$$

- ▶ base: endereço relativo do arranjo na memória
- ▶ i_1 : índice da linha
- ▶ i_2 : índice da coluna
- ▶ n_2 : número de colunas
- ▶ w : tamanho do tipo do dado

Arranjos 3D

$A[i_1][i_2][i_3]$

Arranjos 3D cubos

- Matriz A, tamanho $n_1 * n_2 * n_3$ ($3 * 3 * 4$)

	$i_3=0$	$i_3=1$	$i_3=2$	$i_3=3$	$i_3=0$	$i_3=1$	$i_3=2$	$i_3=3$	$i_3=0$	$i_3=1$	$i_3=2$	$i_3=3$
	8	9	10	11	12	13	14	15	16	17	18	19
$i_2=0$												
	20	21	22	23	24	25	26	27	28	29	30	31
$i_2=1$												
	32	33	34	35	36	37	38	39	40	41	42	43
$i_2=2$												
	$i_1=0$				$i_1=1$				$i_1=2$			

- Qual o endereço de $A[i_1][i_2][i_3]$?

$$e = base + ((i_1 * n_2 + i_2) * n_3 + i_3) * w$$

Arranjos kD

$A[i_1][i_2]\dots[i_k]$

Arranjos kD qualquer arranjo

- Equações para as diferentes formas de endereçamento

$$A[i_1] \quad \text{base} + i_1 * w$$

$$A[i_1][i_2] \quad \text{base} + (i_1 * n_2 + i_2) * w$$

$$A[i_1][i_2][i_3] \quad \text{base} + ((i_1 * n_2 + i_2) * n_3 + i_3) * w$$

$$A[i_1][i_2][i_3][i_4] \quad \text{base} + (((i_1 * n_2 + i_2) * n_3 + i_3) * n_4 + i_4) * w$$

- Forma simplificada

$$A[i_1] \quad \text{base} + d_1 * w \quad d_1 = i_1$$

$$A[i_1][i_2] \quad \text{base} + d_2 * w \quad d_2 = (d_1 * n_2 + i_2)$$

$$A[i_1][i_2][i_3] \quad \text{base} + d_3 * w \quad d_3 = (d_2 * n_3 + i_3)$$

$$A[i_1][i_2][i_3][i_4] \quad \text{base} + d_4 * w \quad d_4 = (d_3 * n_4 + i_4)$$

- Forma geral com **recursividade**

$$A[i_1] \dots [i_k] \quad \text{base} + d_k * w \quad d_k = (d_{k-1} * n_k + i_k)$$

Arranjos kD – Regra

$$A[n_1][n_2]\dots[n_k]$$

$$e = base + d_k * w$$

$$d_k = \begin{cases} d_{k-1} * n_k + i_k & \text{se } k \geq 2 \\ i_k & \text{se } k = 1 \end{cases} \quad (1)$$

Arranjos – Observações

- ▶ Nem sempre os limites do arranjo são de 0 até $n-1$
 - ▶ C – 0 até $n-1$
 - ▶ Pascal – 1 até n
 - ▶ Fortran – *low* até *high* (programador decide)

- ▶ Forma geral considerando *low* e *high*
 - ▶ low_k indica o início da dimensão k do arranjo
 - ▶ $high_k$ indica o final da dimensão k do arranjo

Arranjos 1D – low e high

matriz A, tamanho n

$A[i]$

$$e = \text{base} + (i - \text{low}) * w$$

- ▶ base: endereço relativo do arranjo na memória
- ▶ i: deslocamento do elemento em relação à base
- ▶ w: tamanho do tipo do dado
- ▶ low: começo do arranjo

Arranjos kD – low e high

$$A[n_1][n_2]...[n_k]$$

$$e = base + d_k * w$$

$$d_k = \begin{cases} d_{k-1} * n_k + (i_k - low_k) & \text{se } k \geq 2 \\ i_k - low_k & \text{se } k = 1 \end{cases} \quad (2)$$

$$n_k = high_k - low_k \quad (3)$$

Arranjos 2D – low e high – Exemplo

- ▶ Considerando

```
int A[4][2];  
A[3][1] = x;  
A[1][0] = x;
```

- ▶ Para $A[3][1]$

$$base + ((1 - low) * n_2 + (3 - low)) * w$$

- ▶ Para $A[1][0]$

$$base + ((0 - low) * n_2 + (1 - low)) * w$$

- ▶ Problema

- ▶ TAC tem somente três operandos
- ▶ Cálculo do endereço se torna complexo
- ▶ Esquema de tradução complexo

Melhor abordagem para Geração de TAC

- Endereçamento de 1D

matriz A, tamanho n

$A[i]$

$$e = base + i * w$$

- Considerando low e high

$$e = base + (i - low) * w$$

- Podendo esta ser reescrita da seguinte forma

$$e = base + i * w - low * w$$

$$e = i * w + base - low * w$$

$$e = i * w + (base - low * w)$$

- Ou ainda

$$e = i * w + c$$

c podendo ser calculado na declaração do arranjo

Arranjos 2D – low e high – Exemplo

- ▶ Na declaração do arranjo A

```
int A[4][2];
```

- ▶ Calculando C_A para arranjos 2D, temos

$$C_A = base - ((low_1 * n_2) + low_2) * w$$

- ▶ Para $A[i_1][i_2]$

$$e = ((i_1 * n_2) + i_2) * w + c$$

- ▶ Duas partes

- ▶ Variável, que depende do uso do arranjo - $[i_1][i_2]$
- ▶ Estável, pode ser definida na declaração - c

Arranjos kD – low e high – Generalizando

- ▶ Declaração de um arranjo de k dimensões
type $A[i_1][i_2] \dots [i_k];$
- ▶ Pode-se definir o valor estável C_A
→ inserindo-o na tabela de símbolos juntamente com A

$$C_A = base - ((\dots((low_1 * n_2 + low_2) * n_3 + low_3) \dots) * n_k + low_k) * w \quad (4)$$

$$A[n_1][n_2] \dots [n_k]$$
$$e = d_k * w + C_A$$

$$d_k = \begin{cases} d_{k-1} * n_k + i_k & \text{se } k \geq 2 \\ i_k & \text{se } k = 1 \end{cases} \quad (5)$$

Geração de TAC para Arranjos kD

TAC para kD – Motivação

- ▶ Considerando a declaração em Pascal

A : array (10, 20) of Integer

- ▶ Matriz de 10x20 inteiros, 1 a 10 linhas, 1 a 20 colunas
- ▶ $w = 4$ bytes

- ▶ Equações

$$c = base - ((low_1 * n_2) + low_2) * w$$

$$e = ((i_1 * n_2) + i_2) * w + c$$

- ▶ TAC para o código $x := A[i, j]$

t1 := i * 20

t1 := t1 + j //deslocamento

t2 := c

t3 := 4 * t1 //considerando w

t4 := t2 (t3) //aplicando deslocamento na base

x := t4

TAC para kD – Exemplo

- Supondo a declaração $V[3..10]$
- TAC (em lLOC) para acesso a um posição $i - V[i]$
 - loadI @V $\Rightarrow r_{@V}$ //Carrega endereço de V
 - subI $r_i, 3 \Rightarrow r_1$ //(offset - low)
 - multI $r_1, 4 \Rightarrow r_2$ //multiplica por w
 - add $r_{@V}, r_2 \Rightarrow r_3$ //endereço de $V[i]$
 - load $r_3 \Rightarrow r_V$ //valor de $V[i]$

TAC para kD – Gramáticas

- ▶ Gramática para referências a elementos de arranjos

$$\begin{aligned} L &\rightarrow id[EList] \mid id \\ EList &\rightarrow EList, E \mid E \end{aligned}$$

- ▶ Se torna um esquema L-Atribuído
 - ▶ Atributos sintetizados e herdados
 - ▶ Dificulta a análise
- ▶ Gramática menos natural, mas S-Atribuída

$$\begin{aligned} L &\rightarrow EList \mid id \\ EList &\rightarrow EList, E \mid id[E] \end{aligned}$$

TAC para kD – Gramática completa

S	→	Var := E
E	→	E + E
E	→	(E)
E	→	Var
Var	→	id
Var	→	EList]
EList	→	EList, E
EList	→	id[E

TAC para kD

- ▶ Elist.ndim - contém o número de dimensões da matriz
- ▶ limit(array, j) - função que retorna o número de elementos na dimensão j
- ▶ Elist.nome - temporário que contém o valor computador a partir da expressão Elist
- ▶ Var.desloc - temporário contendo o valor da expressão de endereçamento computada em tempo de execução; quando igual a null, indica variável simples

TAC para kD

S	→	Var := E	if (Var.desloc == NULL) geracod(Var.nome = E.nome); else geracod(Var.nome "[" Var.desloc "]" = E.nome);
E	→	E ₁ + E ₂	E.nome = geratemp(); geracod(E.nome = E ₁ .nome + E ₂ .nome);
E	→	(E ₁)	E.nome = E ₁ .nome;
E	→	Var	if (Var.desloc == NULL) { * Var não é vetor * E.nome = Var.nome; } else { E.nome = geratemp(); geracod(E.nome = Var.nome "[" Var.desloc "]"); }

TAC para kD

Var	→	id	Var.nome = id.nome; Var.desloc = NULL;
Var	→	EList]	Var.nome = geratemp(); Var.desloc = geratemp(); geracod (Var.nome = w * Elist.nome)
EList	→	EList ₁ ,E	t = geratemp(); m = Elist.ndim + 1; geracod (t = Elist ₁ .nome * limit(Elist ₁ .array, m)); geracod (t = t + E.nome); Elist.array = Elist ₁ .array; Elist.nome = t; Elist.ndim = m;
EList	→	id[E	Elist.array = id.nome; Elist.nome = E.nome; Elist.ndim = 1;

Exercício

- ▶ Seja M uma matriz 5×10 , $w = 8$
- ▶ Calcular a árvore para o comando $X = M[i, j + k]$

- ▶ Código TAC resultante

T1 = J + K

T2 = I * 10

T2 = T2 + T1

T3 = endM - 88

T4 = 8 * T2

T5 = T3[T4]

X = T5

Conclusão

- ▶ Leituras Recomendadas
 - ▶ Livro do Dragão
 - ▶ Seções 6.4.3 e 6.4.4,
 - ▶ Série Didática
 - ▶ Seção 5.3.2

- ▶ Próxima Aula
Geração de Código