Tipos de Dados

George Darmiton da Cunha Cavalcanti
(gdcc@cin.ufpe.br)

Tópicos

- Introdução
- Tipos de Dados Primitivos
- Tipos Cadeia de Caracteres
- Tipos Definidos pelo Usuário
- Tipos Matriz
- Matrizes Associativas
- Tipos Registro
- Tipos União
- Tipos Ponteiro

Tipos Matriz

- Uma **matriz** é um agregado homogêneo de dados cujo elemento individual é identificado por sua posição no agregado em relação ao primeiro.

Questões de Projeto

- Quais tipos são legais para os subscritos?
- As expressões de subscrito nas referências a elementos são verificados quanto à faixa?
- Quando as faixas de subscrito são vinculadas?
- Quando a alocação da matriz se desenvolve?
- Quantos subscritos são permitidos?
- Matrizes podem ser inicializadas quando têm seu armazenamento alocado?
- Quais tipos de fatias são permitidos, se for o caso?

Matrizes e Índices

- *Índices* (ou subscritos) fazem mapeamento para elementos

  `array_name(index_value_list)` → `element`

- Sintaxe do Índice
  - FORTRAN, PL/I e Ada usam parênteses
    - Ada usa parênteses para mostrar uma uniformidade entre matrizes e chamadas de funções, pois ambos são mapeamentos
  - Outras linguagens usam colchetes

Tipos dos Índices

- FORTRAN, C: apenas inteiros
- Pascal: qualquer tipo ordinário
  - inteiro, boolean, char, enumeração
- Ada: inteiro ou enumeração (incluindo Boolean e char)
- Java: apenas inteiros

- C, C++, Perl e Fortran não especificam faixa para checagem
- Java, ML e C# especificam a checagem da faixa

Vinculações de Subscritos e Categorias de Matrizes

- *Matriz Estática*
  - As faixas de subscrito estão estaticamente vinculadas e a alocação de armazenamento é estática (feita antes da execução)
  - Vantagem: eficiência (em alocação dinâmica)

- *Matriz fixa dinâmica na pilha*
  - Faixas de subscrito estão estaticamente vinculadas, mas a alocação é feita no momento da declaração durante a execução
  - Vantagem: eficiência de espaço

Vinculações de Subscritos e Categorias de Matrizes

- *Matriz Dinâmica na Pilha*
  - Faixas de subscritos estão dinamicamente vinculadas e a alocação de armazenamento é dinâmica (feita durante a execução)
  - Vantagem: flexibilidade (o tamanho de uma matriz não precisa ser conhecido antes da sua utilização)

- *Matriz Dinâmica no Monte*
  - A vinculação das faixas dos índices e a alocação são dinâmicas e podem mudar várias vezes
  - Vantagem: flexibilidade (matrizes podem crescer ou encolher durante a execução do programa)

Vinculações de Subscritos e Categorias de Matrizes

- Matrizes C e C++ que incluem `static` são estáticas
- Matrizes C e C++ sem `static` são fixas dinâmicas na pilha
- Matrizes Ada podem ser dinâmicas na pilha
- C e C++ também oferecem matrizes dinâmicas (`malloc` e `free`)
- Perl e JavaScript suportam matrizes dinâmicas

Inicialização de Matrizes

- Algumas linguagens permitem a inicialização no momento em que o armazenamento é alocado
  - Exemplos: C, C++, Java e C#
    - `int list [] = {4, 5, 7, 83}`
  - Cadeias de caracteres em C e C++
    - `char name [] = “freddie”;`
  - Matrizes de strings em C e C++
    - `char *names [] = {“Bob”,“Jake”,“Joe”];`
  - Java inicialização de objetos String
    - `String[] names = {“Bob”, “Jake”, “Joe”};`

Operações com Matrizes

- APL permite poderosas operações para matrizes, como também operadores unários
  - Exemplo: inverte os elementos das colunas
- Ada permite atribuição e concatenação de matrizes
- FORTRAN oferece operações **elementares** pois são operações entre pares de elementos de matriz
  - Exemplo: operador + entre duas matrizes resulta em uma matriz que é a soma dos pares de elementos das duas matrizes

Fatias

- Uma **fatia** (slice) de uma matriz é alguma subestrutura desta

- Fatias são úteis em linguagens que possuem operadores sobre matrizes

Fatias: Exemplos

- Fortran 95

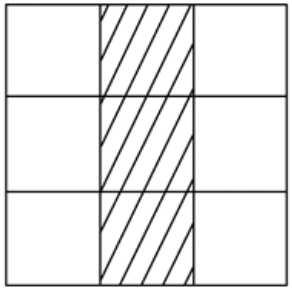
```
Integer, Dimension (10) :: Vector
Integer, Dimension (3,3) :: Mat
Integer, Dimension (3,3,3) :: Cube
```

`Vector (3:6)` é um vetor de 4 elementos

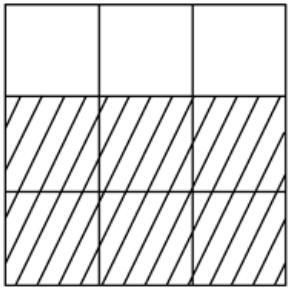
Fatias: Exemplos

Figure 6.4

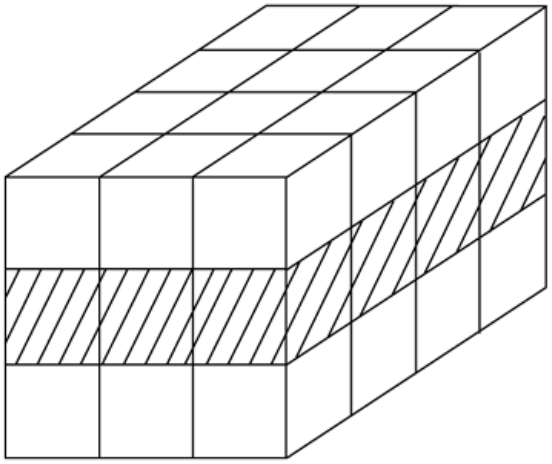
Example slices in Fortran 95



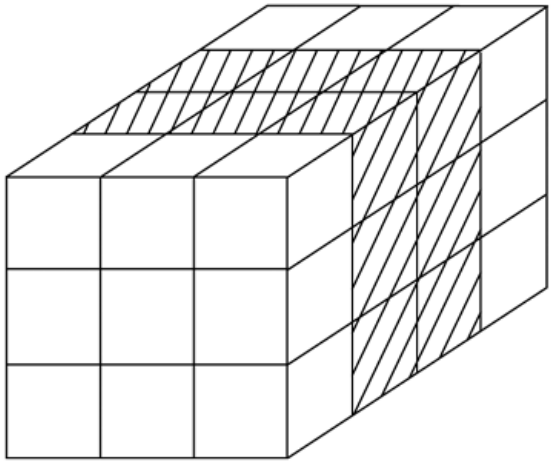
Mat (:, 2)



Mat (2:3, :)



Cube (2, :, :)



Cube (:, :, 2:3)

Implementação do Tipo Matriz

- Funções de aceso mapeiam os índices em endereços na matriz

- Função de acesso a vetores

```
address(list[k]) = address (list[lower_bound])
    + ((k-lower_bound) * element_size)
```

Acessando Matrizes (multidimensionais)

- Duas maneiras comumente usadas
  - Ordem da linha maior (por linhas)
    - Usado na maioria das linguagens
  - Ordem da coluna maior (por colunas)
    - Usando em Fortran

Acessando Matrizes (multidimensionais)

- Exemplo de matriz

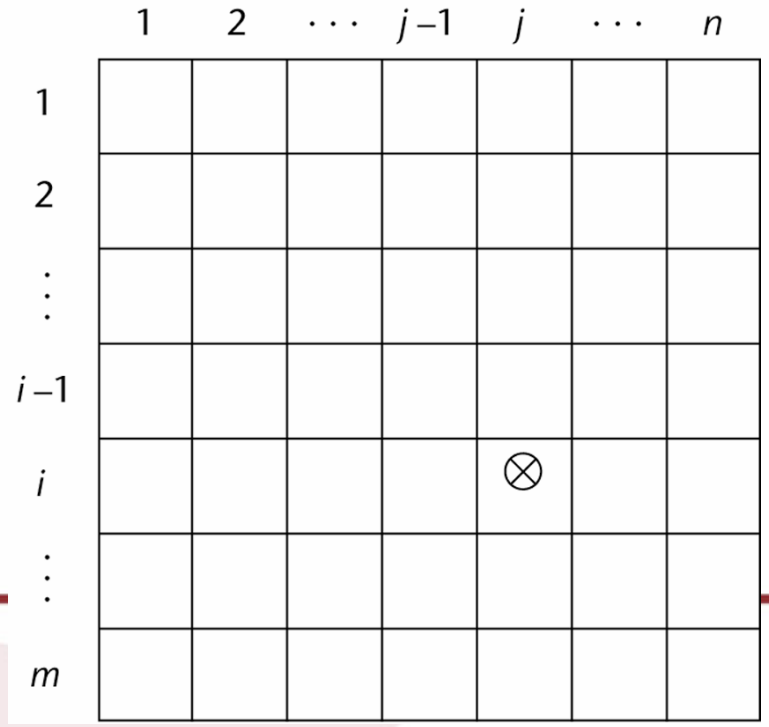
3	4	7
6	2	5
1	3	8

- Ordem da linha maior (por linhas)
  - 3,4,7,6,2,5,1,3,8
- Ordem da coluna maior
  - 3,6,1,4,2,3,7,5,8

Localizando um elemento em uma matriz multidimensional

Formato geral

```
location (a[i,j]) = address of a[row_lb,col_lb] +
(((I - row_lb)*n) + (j - col_lb)) * element_size
```



Matrizes Associativas

- Uma **matriz associativa** é um conjunto não-ordenado de elementos indexados por um número igual de valores chamados chaves
  - Chaves definidas pelos usuários devem ser armazenadas

- Questões de projeto
  - Qual é a forma de referência dos elementos?
  - O tamanho de uma matriz associativa é estático ou dinâmico?

Matrizes associativas em Perl

- Nomes começam com `%;` literais são delimitados por parenteses

```
%hi_temps = ("Mon"=>77, "Tue"=>79,
  “Wed”=>65, …);
```

- Os nomes de variáveis escalares iniciam-se com $

```
$hi_temps{"Wed"} = 83;
```

  – Elementos podem ser removidos

```
delete $hi_temps{"Tue"};
```

Tipos Registro

- Um **registro** é um agregado possivelmente homogêneo de elementos de dados
- Cada elemento individual é identificado por seu nome
- Questões de projeto
  - Qual é a forma sintática das referências a campos?
  - São permitidas referências elípticas?

Definições de Registros

COBOL usa números para aninhar registros; outras linguagens usam definições recursivas

```
01 EMP-REGISTER.
   02 EMP-NAME.
      05 FIRST PIC X(20).
      05 MID   PIC X(10).
      05 LAST  PIC X(20).
   02 HOURLY-RATE PIC 99V99.
```

Referências a Campos do Registro

- A maioria das linguagens usam um ponto na notação
  - `Emp_Rec.Name`

- Referências elípticas (exemplo em Pascal)

```
empregado.nome := 'Bob';
empregado.idade := 42;
empregado.salario := 23750.0;

with empregado do
begin
    nome := 'Bob';
    idade := 42;
    salario := 23750.0;
end;
```

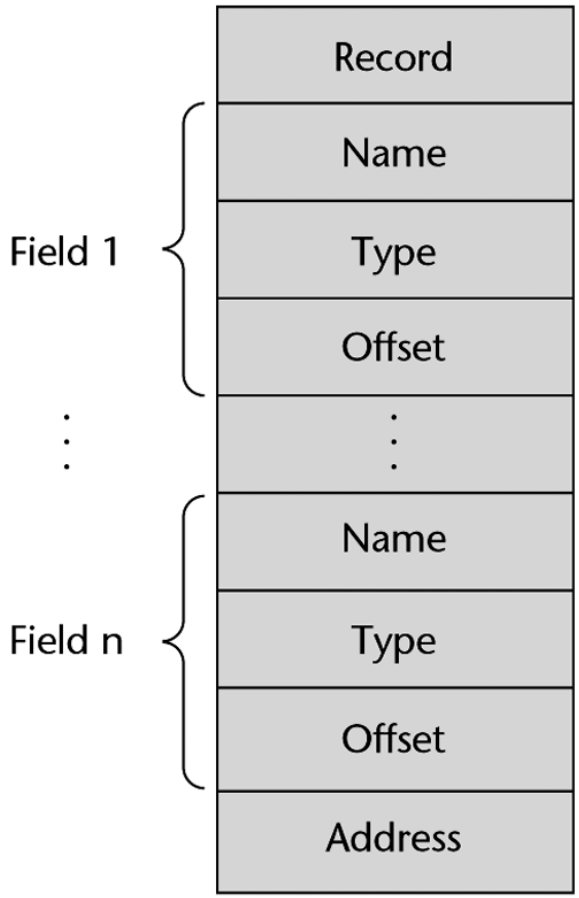
Avaliação e comparação com matrizes

- O projeto de registros é direto e, seu uso, seguro
- Registros são usados quando os dados formam uma coleção heterogênea

Implementação de Tipos Registro

Figure 6.8

A compile-time descriptor for a record



Tipos União

- Uma ***união*** é um tipo que pode armazenar diferentes valores de tipo durante a execução do programa

- Questões de projeto
  - A verificação de tipos deve ser exigida?
    - Note que qualquer verificação de tipos deve ser dinâmica.
  - As uniões devem ser incorporadas aos registros?

Discriminantes vs. Uniões Livres

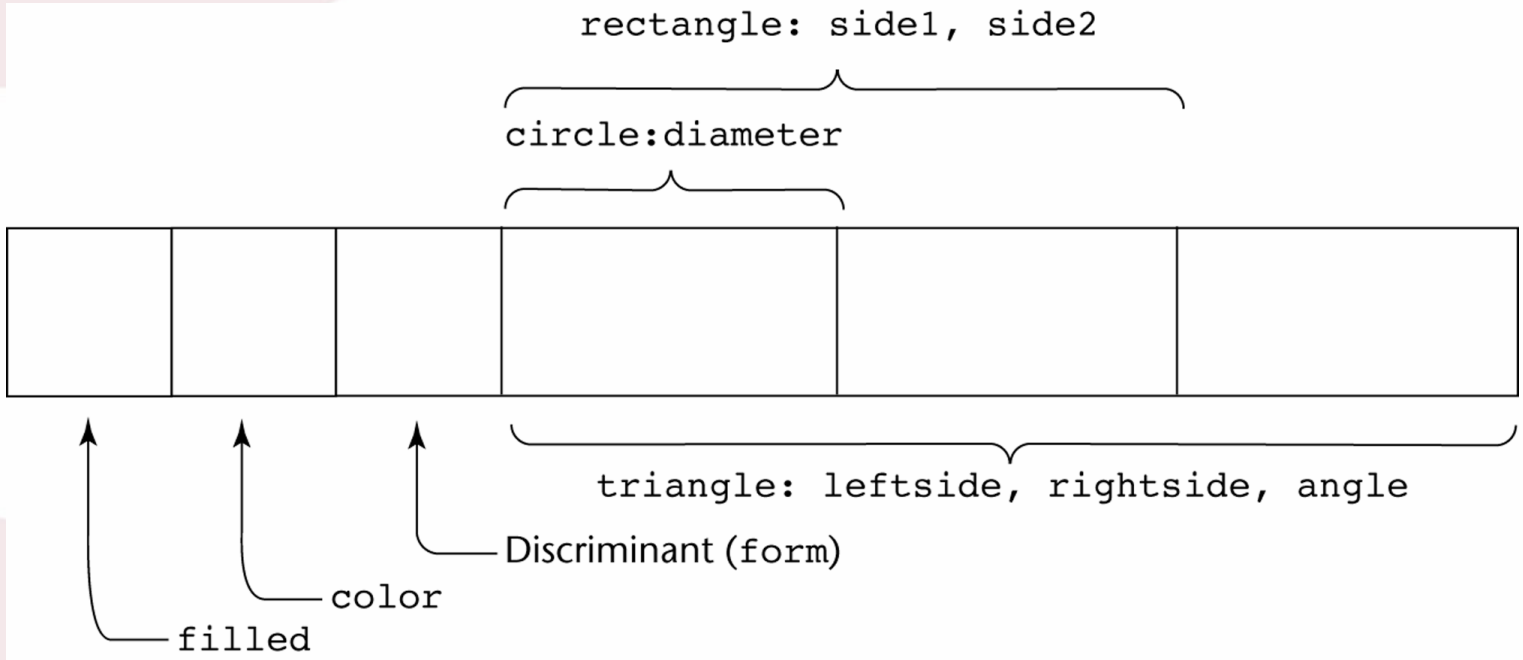
- Fortran, C e C++ oferecem construções de união que não há nenhum suporte na linguagem para verificação de tipos
  - A união nessas linguagens são chamadas de **uniões livres**

- A verificação de tipos em uniões exige que cada construtor de união inclua um indicador de tipo, chamado **discriminante**
  - Suportado por Pascal e por Ada

Uniões em Ada


```
type Shape is (Circle, Triangle, Rectangle);
type Colors is (Red, Green, Blue);
type Figure (Form: Shape) is record
   Filled: Boolean;
   Color: Colors;
   case Form is
       when Circle => Diameter: Float;
       when Triangle =>
             Leftside, Rightside: Integer;
             Angle: Float;
       when Rectangle => Side1, Side2: Integer;
   end case;
end record;
```

Estrutura de um registro variante



Uma união discriminada de três variáveis de forma

Avaliação de Uniões

- Construções potencialmente inseguras
  - Não permitem verificação de tipos das referências a uniões
  - Um dos motivos pelos quais FORTRAN, Pascal, C e C++ não são fortemente tipificadas
- Java e C# não suportam uniões
  - Reflexo da crescente motivação por linguagens de programação mais seguras

Tipos Ponteiro

- Um **tipo ponteiro** é aquele em que as variáveis têm uma faixa de valores que consistem em endereços de memória e um valor especial, *nil*

- Oferece o poder de endereçamento indireto

- Oferece uma alternativa para gerenciar endereçamento dinamicamente

Questões de projeto de ponteiros

- Quais são o escopo e o tempo de vida de uma variável de ponteiro?
- Qual é o tempo de vida de uma variável dinâmica no monte?
- Os ponteiros são restritos quanto ao tipo de valor para o qual eles apontam?

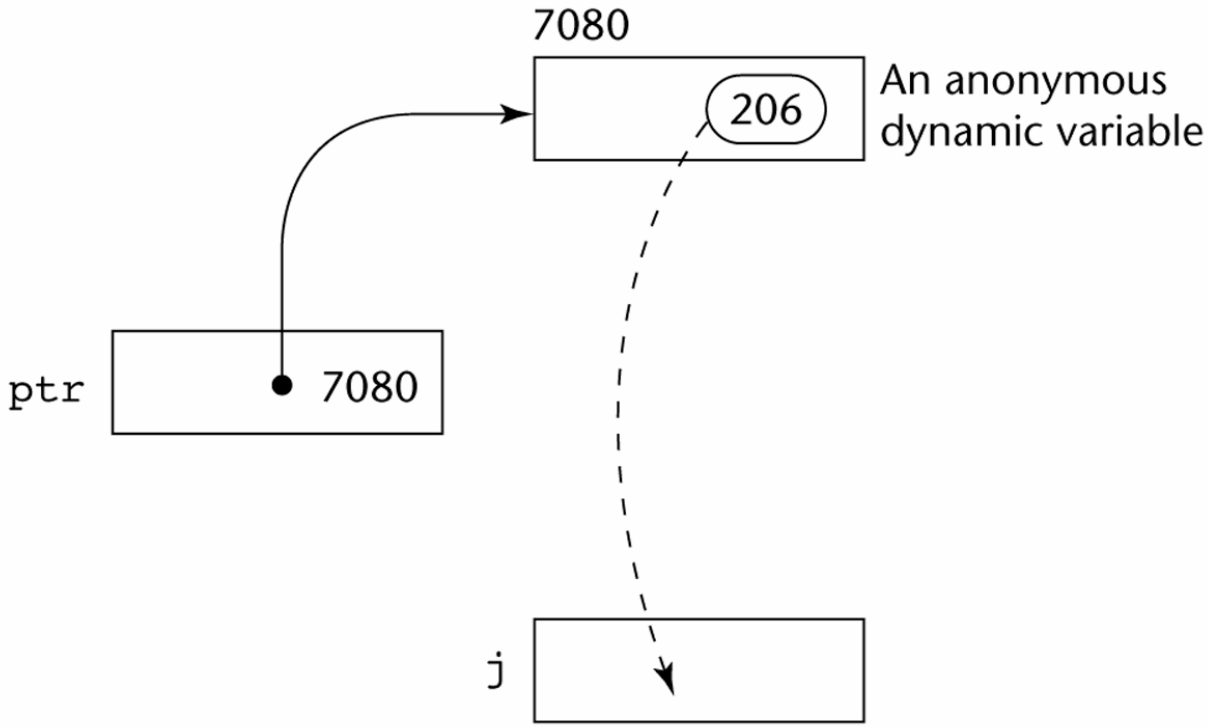
Operações com Ponteiros

- Duas operações fundamentais
  - Atribuição e desreferenciamento
- Atribuição é usada para fixar o valor de uma variável de ponteiro em um endereço útil
- Desreferenciamento referencia o valor da célula de memória (não apenas o endereço)
  - Dereferenciamento pode ser implícito ou explícito
  - C++ usa uma operação explícita ia *

    `j = *ptr`

    fixará j ao valor ao valor alocado em `ptr`

Ilustração de atribuição de ponteiro



A operação de atribuição j = *ptr

Problemas com Ponteiros

- Ponteiros Pendurados (*Dangling pointers*)
  - Um ponteiro que contém o endereço de uma variável dinâmica no monte desalocada

- Variáveis dinâmicas no monte perdidas
  - Uma variável dinâmica no monte alocada não mais acessível ao programa usuário (geralmente chamada de lixo)
    - Ponteiro `p1` é ajustado para apontar para uma variável dinâmica no monte recém-criada
    - Mais tarde, ponteiro `p1` é ajustado para outra variável dinâmica no monte recém-criada

    - Vazamento de memória

Sumário

- Os tipos de dados de uma linguagem são uma grande parte daquilo que determina o seu estilo e seu uso
- Os tipos de dados primitivos da maioria das linguagens imperativas incluem os tipos numérico, caractere e booleano
- Os tipos enumeração e a subfaixa definidos pelo usuário são convenientes e aumentam a legibilidade e a confiabilidade dos programas
- Matrizes e registros estão presentes na maioria das linguagens de programação
- Ponteiros são usados para dar flexibilidade de endereçamento e para controlar o gerenciamento de armazenamento dinâmica