Capítulo 6

Tipos de Dados
(continuação)

Tipo Definido pelo Usuário: ordinal

- Não é comum, é mais “teórico”
- Corresponde a um tipo de dados que no qual é possível colocar os valores em correspondência com o conjunto dos inteiros
- Examplos em Java
  - **`integer`**
  - **`char`**
  - **`boolean`**

Tipo: Enumeração

- É um conjunto de nomes simbólicos, associados ou não à valores constantes

- Exemplo em C#

```
enum dias {seg, ter, qua, qui, sex, sáb, dom};
```

- Exemplo em Python:

```
from enum import Enum

class Cores(Enum):
    VERMELHO = 1
    AZUL = 2
    VERDE = 3

print(Cores.AZUL.value)
```

Tipos: Arrays

- Array: é um agregado **homogêneo** de elementos de dados nos quais um elemento individual é identificado por sua posição no agregado

Questões no projeto de Arrays

- O que pode ser um subscrito?
- Os subscritos podem ser checados pela validade da posição?
- Arrays retangulares ou multidimensionais são existentes

Índices em Arrays

- *Indexing* é o mapeamento de índices aos elementos dos vetores
- O índice muda de acordo com a linguagem:
  - Inteiro:
    - FORTRAN, C, JAVA
  - Checagem da amplitude do índice:
    - JAVA, ML, C#
  - Sem checagem da amplitude do índice:
    - C, C++, Perl, FORTRAN

Heterogeneous Arrays

- Um array ***heterogêneo*** é aquele onde os elementos não precisam ser do mesmo tipo
- Existem em: LISP, Perl, Python, JavaScript, eRuby

Inicialização de Arrays

- Exemplos:
  - C, C++, Java, C#:

```
int list [] = {4, 5, 7, 83}
```

  - Strings de caracteres em C, C++

```
char name [] = "renata";
```

  - Arrays de strings em C e C++

```
char *nomes [] = {"João", "José", "Janaína"];
```

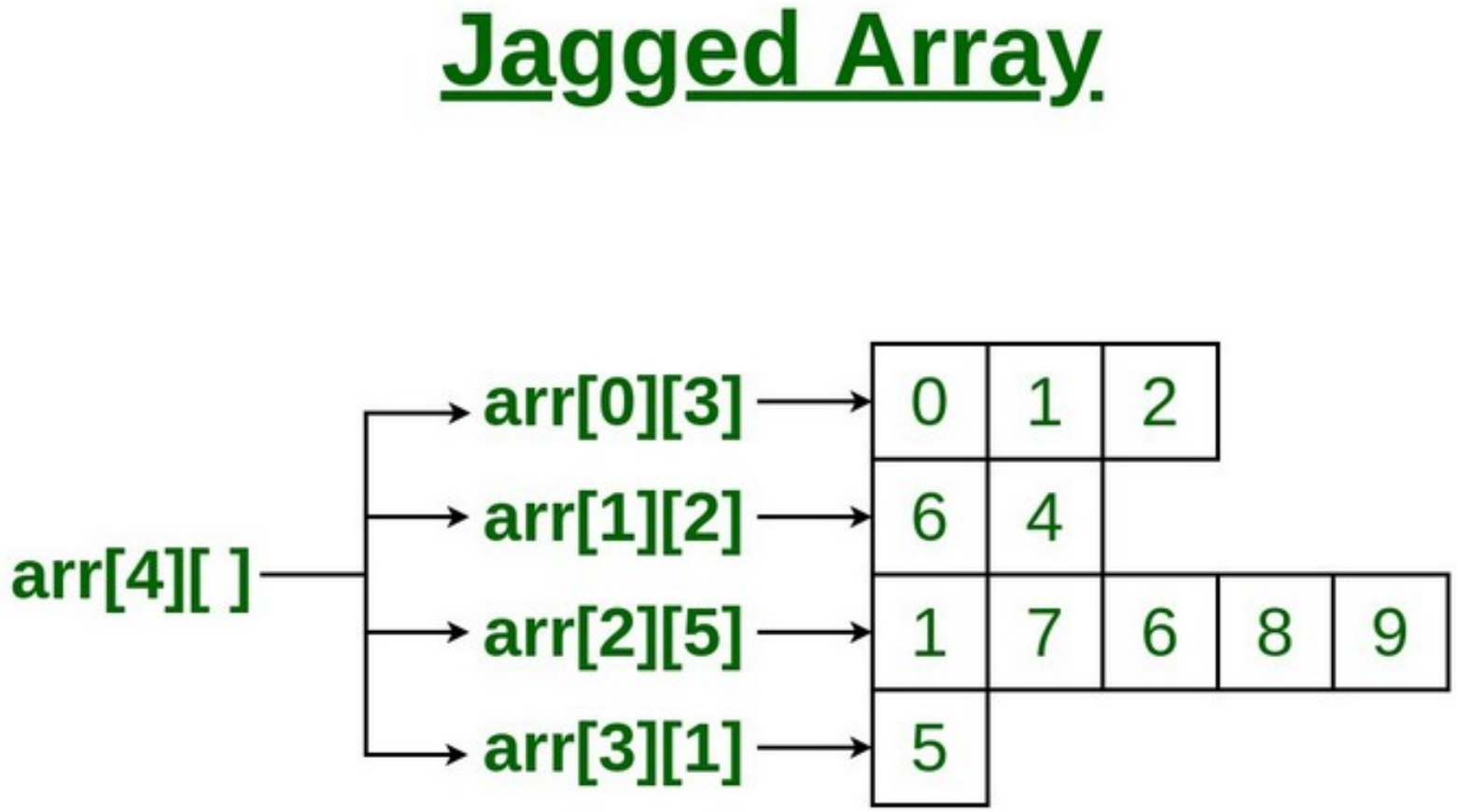
  - Strings em Java (são objetos)

```
String[] nomes = {"João", "José", "Janaína"};
```

Arrays Regulares e Irregulares

- Um **array regular** é um array **multidimensional** no qual todos as linhas têm o mesmo número de elementos e todas as colunas têm o mesmo número de elementos
- Um **array irregular** possui linhas com número variável de elementos:
  - Possível quando arrays multidimensionais aparecem como arrays de arrays
- C, C++ e Java suportam arrays irregulares

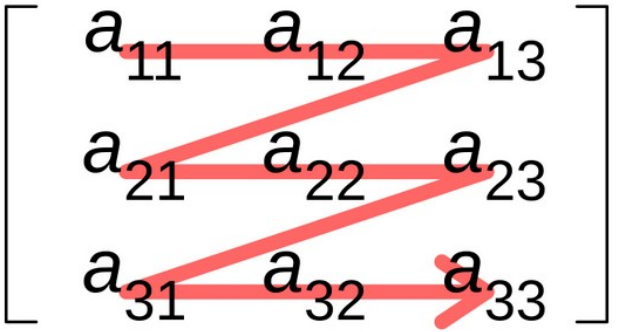
Arrays Regulares e Irregulares



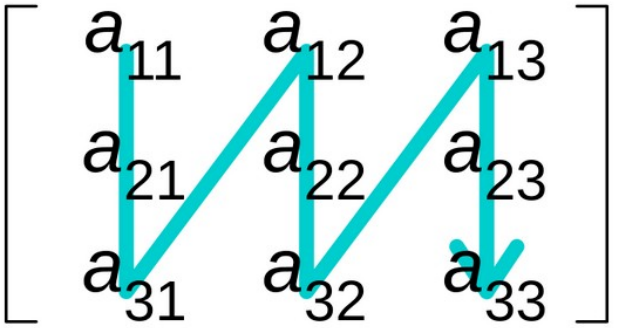
Acesso à Arrays Multidimensionais

- 2 modos comuns:
  - **Row major order** (por linha) – usado na maioria das linguagens
    - **(Largura * x) + y**
      (#colunas * x) + y
  - **Column major order** (por colunas) – usado em Fortran
    - **(Altura * y) + x**
      (#linhas * y) + x

Row-major order



Column-major order



Array Associativo (dicionário)

- É uma coleção não ordenada de dados que é indexada por uma chave, formando um par **chave:valor**

```
notas = {"fulano": 3,
         "ciclano": 8,
         "beltrano": 9.5,
         "zezinho": 6.8}

print(notas["ciclano"])
```

Listas

- São estruturas de armazenamento versáteis, que podem armazenar tipos diferentes de dados!
- Em Python, também serve para armazenar os arrays!
- Em Java, arrays são diferentes de listas (usar a classe List ou ArrayList)
- Podem ser:
  - Imutáveis: LISP, Scheme, ML
  - Mutáveis: Python
- Podem ser acessadas por índices (começa em 0)
- Geralmente implementadas por listas encadeadas simples ou duplas

```
minha_lista = [1, 2, 3, "banana", "maçã"]
print(minha_lista[3])
```

Tuplas

- Semelhantes à listas, mas são imutáveis
- Costumam ser usadas (nas linguagens que suportam) para permitir que as funções retornem mais de um resultado
- Elementos são indexados (começam com 1)

```
def estatisticas(lista):
    soma = 0
    min = lista[0]
    max = lista[0]
    for i in range(len(lista)):
        soma = soma + lista[i]
        if lista[i] >= max:
            max = lista[i]
        if lista[i] <= min:
            min = lista[i]
    media = soma / len(lista)
    return min, max, media
```

```
minha_tupla = 1, 2, "banana", (3, 4, "maçã")
print(minha_tupla)
```

```
min, max, media = estatisticas([1, 2, 3, 4, 5, 6, 7])
```

Outros tipos (sem detalhes)

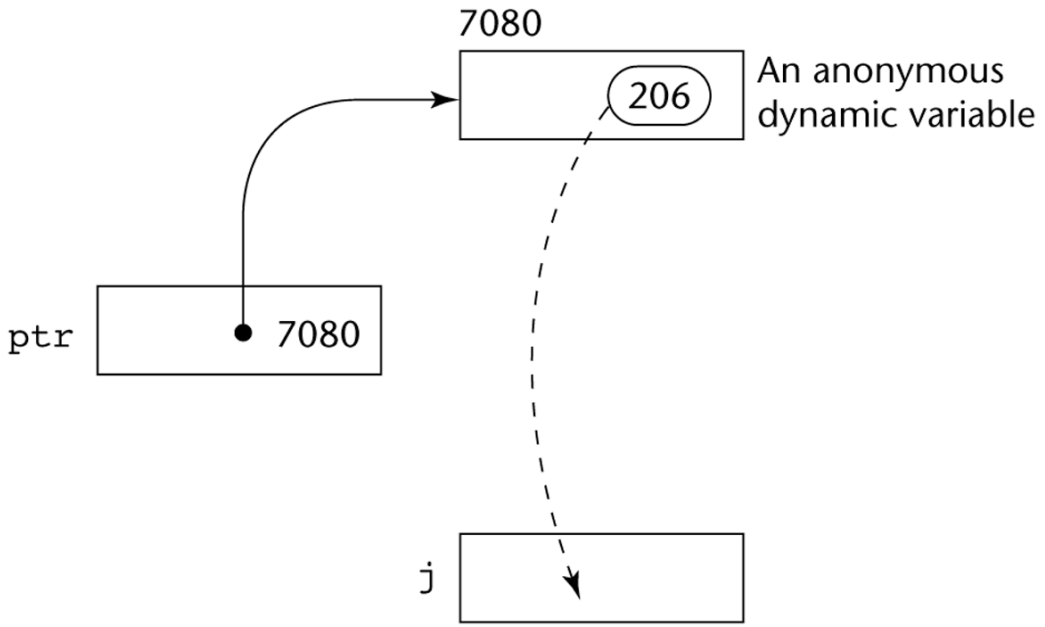
- União
- Registro

Ponteiros

- Um ponteiro é um tipo de dado que armazena endereços de memória (ou nil)
- Fornecem endereçamento indireto
- Pode ser usado para acessar uma localização em uma área de memória dinâmica, como a heap

Exemplo de ponteiro

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    int *ptr = malloc(sizeof(int));
    *ptr = 206;

    int j = 0;
    j = *ptr;

    printf("%d\n", j);

    return 0;
}
```



Problemas com Pointeiros

- **Dangling pointers** (ponteiro pendurado)
  - O ponteiro aponta para uma variável dinâmica na heap que foi desalocada
- **Lost heap-dynamic variable** (lixo)
  - É uma variável dinâmica na heap que não está mais acessível por nenhum ponteiro
  - Memory leak

Problemas com Pointeiros

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    int *ptr = malloc(sizeof(int));
    *ptr = 206;

    int j = 0;
    j = *ptr;

    printf("%d\n", j);

    free(ptr); // Agora tenho dangling pointer

    printf("%d\n", *ptr); // Imprime lixo

    ptr = NULL; // Dangling pointer resolvido

    printf("%d\n", *ptr); // Erro no programam, que bom!

    return 0;
}
```

Problemas com Pointeiros

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    int *ptr = malloc(sizeof(int));
    *ptr = 206;

    int j = 50;
    ptr = &j;  // Provoquei memory leak, pois apontei
               // o ponteiro para outra coisa, e o que
               // estava na heap (206) virou lixo

    printf("%d\n", *ptr);

    return 0;
}
```

Pointeiros em C e C++

- Muito flexíveis, mas deve ser usados com cuidado
- Podem apontar para qualquer variável
- Usados para gerencimento de armazenamento dinâmico de memória
- Pode ser feita aritmética de ponteiros
- Separam os meninos dos homens

Aritmética com ponteiros

```
float teste[100];
float *p;
p = teste;
```

`*(p+5)` é equivalente a `teste[5]` e `p[5]`

`*(p+i)` é equivalente a `teste[i]` e `p[i]`

UFA!

Hora de comemorar!