FaculdadeBarretos
www.faculdadebarretos.com.br

Algoritmos e Estruturas de Dados

(Aula 4 - Estruturas Compostas homogêneas Vetores e Matrizes)

Prof. Me. Diogo Tavares da Silva
contato: *diogotavares@unibarretos.com.br*

FaculdadeBarretos
www.faculdadebarretos.com.br

Relembrando..

- aula anterior:
  - Funções em C/C++
    - Passagem por cópia x passagem por referência
  - Variáveis do tipo **ponteiro** (referência)

Contextualização

- Até o momento trabalhamos apenas com variáveis simples
  - Armazenam apenas um único valor

Contextualização

- obviamente, esta abordagem possui limitações que nos impedem de explorar o poder dos computadores
  - Capacidade massiva de processamento de **coleções** de dados
    - **Estruturas de dados compostas**
      - armazenam conjuntos de informações!

Contextualização

- Na aula de hoje, analisaremos as implementações mais simples de coleções de dados
  - **Estruturas compostas homogêneas**
    - também conhecidas como
      - **VETORES**
      - **MATRIZES**
        - bidimensionais ou n-dimensionais

Estruturas compostas homogêneas

- É importante observar!
  - para todas as estruturas apresentadas neste curso analisaremos:
    - **criação**
    - **percurso**
    - **busca**
    - **inserção**
    - **remoção**
    - **análise de complexidade dos algoritmos utilizados**

Vetores em C++

- “Variáveis compostas homogêneas unidimensionais (ou, simplesmente, vetores) são estruturas capazes de armazenar uma coleção de valores do mesmo tipo.”
  - Cada um desses valores é referenciado pelo mesmo nome de variável (o nome dado ao vetor), sendo diferenciados apenas pelo índice de acesso (posição no vetor)”

Vetores em C++

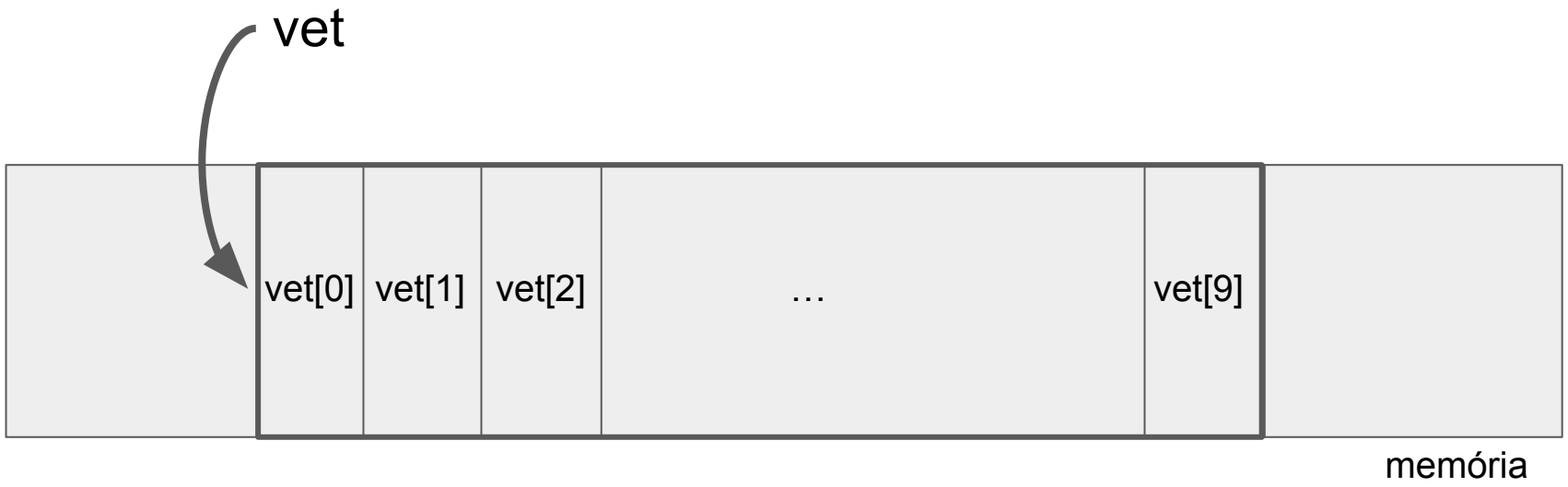
- Características do vetor:
  - Uma única variável que faz **referência** a um conjunto de dados.
    - ponteiro
  - **Estrutura homogênea**
    - armazena dados do mesmo tipo
  - **Unidimensional e sequencial**
    - todos os valores são alocados em posições sequenciais de memória e acessados a partir de um único índice

Vetores em C/C++

- Como o vetor é alocado na memória?
  - arranjado sequencialmente na memória
  - variável faz referência à primeira posição do vetor
  - próximas são acessadas com base na primeira posição
    - “aritmética de ponteiros”

Vetores em C/C++

int vet[10];



Vetores em C/C++

- Vetores por ser alocados de duas formas:
  - **Alocação Estática**
    - mais simples
    - O tamanho do vetor é fixo e definido em tempo de compilação
    - não pode ser alterado posteriormente durante a execução

Vetores em C/C++

- Vetores por ser alocados de duas formas:
  - **Alocação Dinâmica**
    - mais complexa
    - O vetor é inicialmente definido com uma variável de ponteiro.
    - O tamanho do vetor é definido em tempo de execução, a quantidade necessária de memória é alocada e o ponteiro passa a referenciar o espaço alocado, criando assim o vetor dinâmico.

Vetores em C/C++

- Vetores por ser alocados de duas formas:
  - **Alocação Estática**
  - **Alocação Dinâmica**

Vetores em C/C++

- **Declarando um vetor:**
  - <tipo_dado> <nome_vetor> [<num_posições>];
  - ex:
    - `int vetor[10];`
    - `float notasSemestrais[200];`
    - `char nome[150];`
    - `#define TAM 3000 //macro para definir uma constante em C`

      `int valores[TAM];`

Vetores em C/C++

- **atribuição:**

`vet[0] = 1;`	atribui o valor 1 à primeira posição do vetor (lembre-se de que o vetor começa na posição zero).
`x[3] = 'b';`	atribui a letra b à quarta posição do vetor (lembre-se de que o vetor começa na posição zero).

Vetores em C/C++

- **Leitura de dados**
  - **preenchendo um vetor pelo teclado:**

```
#define tam 50
int nums[tam];
for (int i =0; i<tam; i++){
   cout << "Digite o valor da posição " << i <<":";
   cin >> nums[i];
}

//sempre temos que controlar o tamanho do vetor “manualmente”
```

Vetores em C/C++

- **Escrita em tela**
  - **imprimindo um vetor:**

```
#define tam 50
//...leitura
//...processamento
for (int i =0; i<tam; i++){
   cout << "Valor da posição " << i <<":";
   cout << nums[i] << endl;
}
```

Vetores em C/C++

- **Processamento um vetor:**
  - **principais desafios**
    - **tamanho do vetor**
      - manter a informação sobre o tamanho do vetor em uma variável e controlar a todo tempo
    - **alocação estática**
      - em tempo de compilação
        - predefinir um tamanho limite e ir preenchendo
    - **organização sequencial**
      - inserir ou deletar informações implica em alterar a posição de todos os elementos do vetor
        - deslocar todos os dados pra frente ou pra trás
          - a menos que seja no fim do mesmo

Vetores em C/C++

- **Vetores e funções:**
- **passagem por referência:**
  - a função recebe como parâmetro uma referência a variável original e a altera diretamente.
  - vetores são passados por referência
  - uso dos conceitos de endereço e de ponteiros

exemplo:

```
int maiorNota = maior(notas[]);

….

int maior(int vet[]){...}
```

Vetores em C/C++

- **Exercícios (Laboratório)**
  a. criar um vetor para receber um número n de elementos definido pelo usuário
  b. ler os n valores
  c. imprimir os n valores
  d. função para retornar menor valor
  e. função para retornar maior valor
  f. função para retornar a soma dos valores
  g. função para retornar a média dos valores

Vetores em C/C++

- **exercícios**

  pág. 159 do livro da Ascêncio

Para casa…

- Estudar matrizes (próximos slides)

A Linguagem C/C++

- Livro-base
  - **Fundamentos da Programação de Computadores**
    - Ana Fernanda Gomes Ascencio
    - Edilene Aparecida Veneruchi de Campos

Matrizes em C/C++

- Uma matriz é uma variável composta homogênea multidimensional.
- Ela é formada por uma sequência de variáveis, todas do mesmo tipo, com o mesmo identificador (mesmo nome), e alocadas sequencialmente na memória.
- Uma vez que as variáveis têm o mesmo nome, o que as distingue são índices que referenciam sua localização dentro da estrutura.
- Uma variável do tipo matriz precisa de um índice para cada uma de suas dimensões.

Matrizes em C/C++

- geralmente bi-dimensional
- ex:

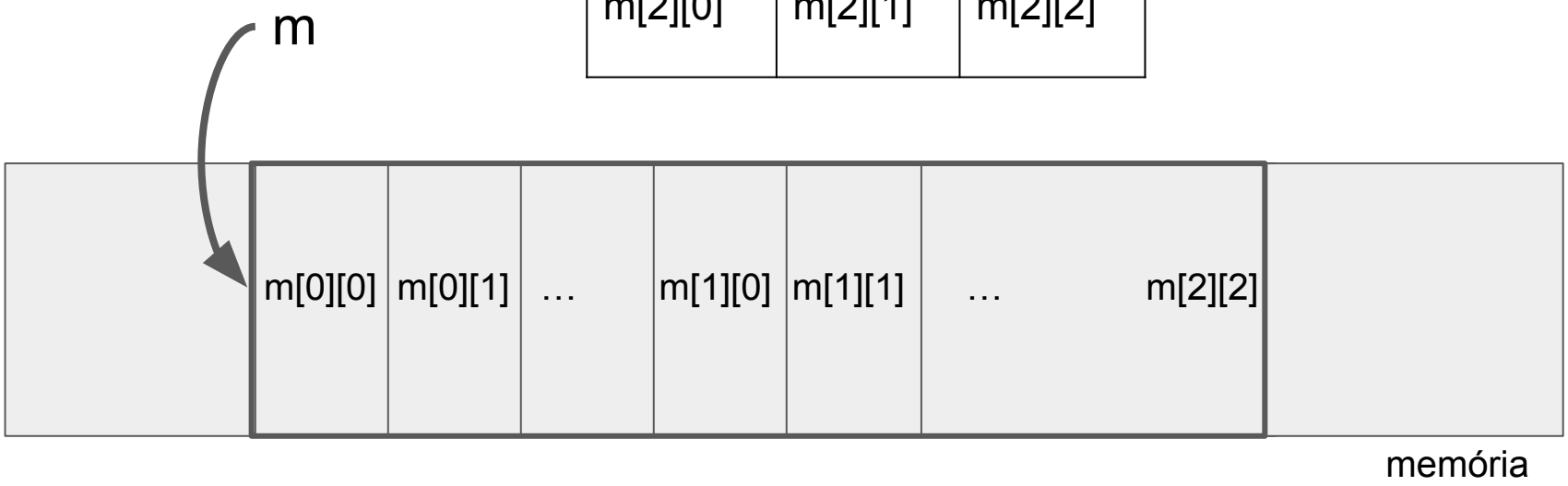
  int m[3][3];

m[0][0]	m[0][1]	m[0][2]
m[1][0]	m[1][1]	m[1][2]
m[2][0]	m[2][1]	m[2][2]

Matrizes em C/C++

- int m[3][3];
- na memória:
  - na prática é um “vetor de vetor”

m[0][0]	m[0][1]	m[0][2]
m[1][0]	m[1][1]	m[1][2]
m[2][0]	m[2][1]	m[2][2]



Matrizes em C/C++

- declaração de uma matriz

```
tipo_dos_dados nome_variável [dimensão1] [dimensão2] [...] [dimensãoN];
```

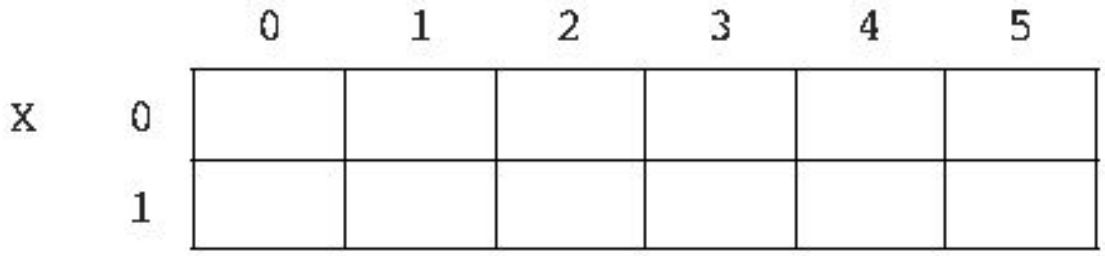
- ex:
  - int mat[2][3];
  - int posPlano[1080][1920];
  - int posEspaco[1000][1000][1000];
  - int rgb[256][256][256];
  - float X[2][6];

Matrizes em C/C++

- declaração de uma matriz

```
tipo_dos_dados nome_variável [dimensão1] [dimensão2] [...] [dimensãoN];
```

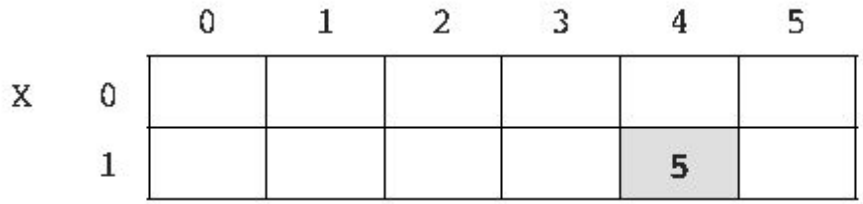
- ex:
  - float X[2][6];



Matrizes em C/C++

- atribuição em uma matriz

```
X[1][4] = 5;
```
→ Atribui o valor 5 à posição identificada pelos índices 1 (2ª linha) e 4 (5ª coluna).



Matrizes em C/C++

- Ler valores de uma matriz pelo teclado

```
#define tamL 50
#define tamC 30
int mat[tamL][tamC];
for (int i =0; i<tamL; i++){
   for (int j = 0; j<tamC; j++){
      cout << "Digite o valor da posição [" << i <<"][" << j << "]:";
      cin >> mat[i][j];
   }
}

//temos que controlar os tamanhos das dimensõe da matriz
“manualmente”
```

Matrizes em C/C++

- imprimir valores de uma matriz:

```
#define tamL 50
#define tamC 30
int mat[tamL][tamC];
for (int i =0; i<tamL; i++){
   for (int j = 0; j<tamC; j++){
      cout << "Valor da posição [" << i <<"][" << j << "]:";
      cout << mat[i][j];
   }
}

//temos que controlar os tamanhos das dimensões da matriz “manualmente”
```

Matrizes em C/C++

Exercícios:

1. ler uma matriz de dimensões m, n escolhidas pelo usuário
2. imprimir essa matriz
3. encontrar o menor elemento da matriz
4. encontrar o maior elemento da matriz
5. soma da matriz
6. média da matriz
7. Encontrar o menor elemento de cada linha
8. encontrar o maior elemento de cada linha
9. encontrar a soma de cada linha
10. encontrar a média de cada linha

Matrizes em C/C++

Exercícios:

pág. 220 do livro