



# INF 1007 – Programação II

## Aula 07 – Vetor de Ponteiros

Edirlei Soares de Lima  
<elima@inf.puc-rio.br>

# Vetor de Cadeia de Caracteres

- Vetores de cadeias de caracteres.
- Exemplo:

```
char alunos[50][81];
```

- Função para imprimir os nomes dos alunos de turmas com 50 alunos, onde o nome dos alunos possui no máximo 80 caracteres:

```
void imprime(int n, char alunos[][81])  
{  
    int i;  
    for (i=0; i<n; i++)  
        printf("%s\n", alunos[i]);  
}
```

# Vetor de Cadeia de Caracteres

- Um vetor de cadeia de caracteres pode ser alocado de duas formas:
  - **Alocação Estática:**
    - alocação como matriz estática de elementos do tipo char;
  - **Alocação Dinâmica:**
    - alocação como vetor de ponteiros, onde cada cadeia de caracteres (elemento do vetor) é alocada dinamicamente;

# Vetor de Cadeia de Caracteres

- **Alocação Estática:**

```
char str[3][15] = {"Bom Dia.", "Boa Tarde.", "Boa Noite"};
```

- A declaração é equivalente a matriz:

$$\begin{bmatrix} B & o & m & & D & i & a & . & \backslash 0 & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ B & o & a & & T & a & r & d & e & . & \backslash 0 & \dots & \dots & \dots & \dots & \dots \\ B & o & a & & N & o & i & t & e & \backslash 0 & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

- É possível alterar um caractere ou imprimir todo um elemento:

```
str[1][4] = 'X';  
printf("%s\n", str[1]);
```

- Quando passamos o vetor de caracteres como parâmetro para uma função, o protótipo da função deve ser:

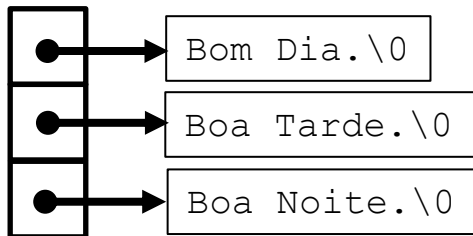
```
void teste(char str[][15], int n);
```

# Vetores de Ponteiros

- **Alocação Dinâmica:**

```
char *str[] = {"Bom Dia.", "Boa Tarde.", "Boa Noite"};
```

- Dessa forma temos um vetor de ponteiros:



A vantagem é que cada elemento aponta para strings de tamanhos diferentes.

- É possível imprimir um elemento:

```
printf("%s\n", str[1]);
```

- Mas nesse caso não podemos alterar os caracteres dos elementos, pois utilizamos constates de strings.

# Vetores de Ponteiros

- **Alocação Dinâmica:**

- Para podermos alterar os caracteres dos elementos, é necessário alocar as cadeias de caracteres dinamicamente:

```
char *str[3];
char *novo_str = "Bom Dia.";
str[0] = (char*)malloc((strlen(novo_str)+1) * sizeof(char));
strcpy(str[0], novo_str);
...
str[0][4] = 'X';
printf("%s", str[0]);
```

- Quando passamos o vetor de ponteiros como parâmetro para uma função, o protótipo da função deve ser:

```
void teste(char **str, int n);
```

# Vetores de Ponteiros

- **Alocação Dinâmica:**


- Também é possível alocar toda a matriz como ponteiros de ponteiro:

```
char *novo_elemento;  
char **str;  
str = (char**)malloc(3 * sizeof(char*));  
  
novo_str = "Bom Dia.";  
str[0] = (char*)malloc((strlen(novo_str)+1) * sizeof(char));  
strcpy(str[0], novo_str);  
  
novo_str = "Boa Tarde.";  
str[1] = (char*)malloc((strlen(novo_str)+1) * sizeof(char));  
strcpy(str[1], novo_str);
```

- O protótipo de funções que recebem a estrutura deve ser:

```
void teste(char **str, int n);
```

# Vetores de Ponteiros para Estruturas

- **Exemplo:** crie um programa que armazene os dados de alunos em um vetor, permitindo a preenchimento, remoção e impressão dos dados.
  - **Dados de cada aluno:**
    - **matrícula:** número inteiro;
    - **nome:** cadeia com até 80 caracteres;
    - **endereço:** cadeia com até 120 caracteres;
    - **telefone:** cadeia com até 20 caracteres.
- 



# Vetores de Ponteiros para Estruturas

- **Solução 1:**

```
#define MAX 100

struct aluno
{
    int mat;
    char nome[81];
    char end[121];
    char tel[21];
};

typedef struct aluno Aluno;
```

a estrutura ocupando pelo menos  $4 + 81 + 121 + 21 = 227$  bytes

**Aluno tab[MAX];**

**vetor de Aluno:** representa um desperdício significativo de memória, se o número de alunos for bem inferior ao máximo estimado

# Vetores de Ponteiros para Estruturas

- **Solução 2:**

```
#define MAX 100

struct aluno
{
    int mat;
    char nome[81];
    char end[121];
    char tel[21];
};

typedef struct aluno Aluno;

Aluno* tab[MAX]; ←
```

## Vetor de ponteiros para Aluno:

- um elemento do vetor ocupa espaço de um ponteiro;
- alocação dos dados de um aluno no vetor:
  - nova cópia da estrutura Aluno é alocada dinamicamente
  - endereço da cópia é armazenada no vetor de ponteiros
- posição vazia do vetor: valor é o ponteiro nulo

# Vetores de Ponteiros para Estruturas

- Função `inicializa` para inicializar a tabela:
  - recebe um vetor de ponteiros (parâmetro deve ser do tipo “**ponteiro para ponteiro**”);
  - atribui NULL a todos os elementos da tabela;

```
void inicializa(int n, Aluno** tab)
{
    int i;
    for (i=0; i<n; i++)
        tab[i] = NULL;
}
```

# Vetores de Ponteiros para Estruturas

- Função `preenche` para armazenar um novo aluno na tabela:
  - recebe a posição onde os dados serão armazenados
  - se a posição da tabela estiver vazia, função aloca nova estrutura
  - caso contrário, função atualiza a estrutura já apontada pelo ponteiro

```
void preenche(int n, Aluno** tab, int i)
{
    if (i < 0 || i >= n)
    {
        printf("Indice fora do limite do vetor\n");
        exit(1);      /* aborta o programa */
    }
    if (tab[i] == NULL)
        tab[i] = (Aluno*)malloc(sizeof(Aluno));
    printf("Entre com a matricula:");
    scanf("%d", &tab[i]->mat);
    ...
}
```

# Vetores de Ponteiros para Estruturas

- Função `retira` para remover os dados de um aluno:
  - recebe a posição da tabela a ser liberada;
  - libera espaço de memória utilizado para os dados do aluno;

```
void retira(int n, Aluno** tab, int i)
{
    if (i<0 || i>=n)
    {
        printf("Indice fora do limite do vetor\n");
        exit(1);          /* aborta o programa */
    }
    if (tab[i] != NULL)
    {
        free(tab[i]);
        tab[i] = NULL; /* indica que na posição não mais existe dado */
    }
}
```

# Vetores de Ponteiros para Estruturas

- Função `imprime` para imprimir os dados de um aluno:
  - recebe a posição da tabela a ser impressa;

```
void imprime(int n, Aluno** tab, int i)
{
    if (i<0 || i>=n)
    {
        printf("Indice fora do limite do vetor\n");
        exit(1);          /* aborta o programa */
    }
    if (tab[i] != NULL)
    {
        printf("Matrícula: %d\n", tab[i]->mat);
        printf("Nome: %s\n", tab[i]->nome);
        printf("Endereço: %s\n", tab[i]->end);
        printf("Telefone: %s\n", tab[i]->tel);
    }
}
```

# Vetores de Ponteiros para Estruturas

- Função `imprime_tudo` para imprimir todos os dados da tabela:
  - recebe o tamanho da tabela e a própria tabela;

```
void imprime_tudo(int n, Aluno** tab)
{
    int i;
    for (i=0; i<n; i++)
    {
        imprime(n, tab, i);
    }
}
```

# Vetores de Ponteiros para Estruturas

- Função principal:

```
int main (void)
{
    Aluno* tab[10];
    inicializa(10,tab);
    preenche(10,tab,0);
    preenche(10,tab,1);
    preenche(10,tab,2);
    imprime_tudo(10,tab);
    retira(10,tab,0);
    retira(10,tab,1);
    retira(10,tab,2);
    return 0;
}
```




# Vetores de Ponteiros para Estruturas

- Função principal:

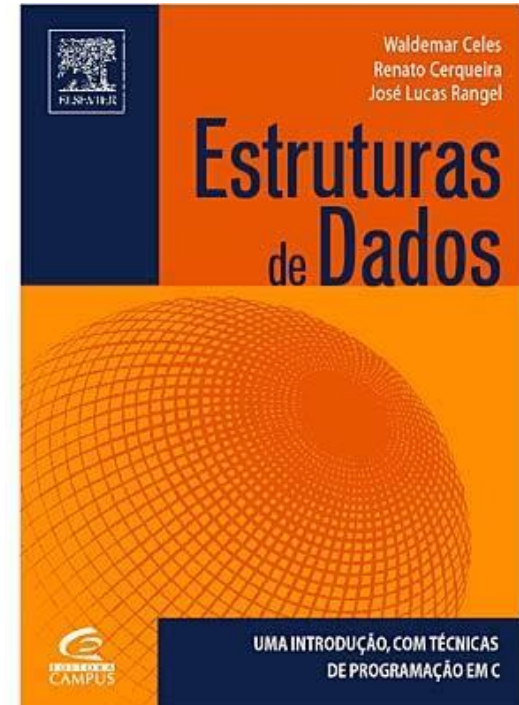
```
int main (void)
{
    Aluno** tab;
    int tam;
    printf("Qual o tamanho do vetor?");
    scanf("%d", &tam);
    tab = (Aluno**)malloc(tam * sizeof(Aluno*));
    inicializa(tam, tab);
    preenche(tam, tab, 0);
    preenche(tam, tab, 1);
    preenche(tam, tab, 2);
    ...
    return 0;
}
```

**Também podemos alocar o vetor dinamicamente!**



# Leitura Complementar

- Waldemar Celes, Renato Cerqueira, José Lucas Rangel, **Introdução a Estruturas de Dados**, Editora Campus (2004).
- **Capítulo 6 – Cadeia de caracteres**
- **Capítulo 8 – Tipos Estruturados**



# Exercícios

## Lista de Exercícios 06 – Vetor de Ponteiros

<http://www.inf.puc-rio.br/~elima/prog2/>

