Universidade Federal da Grande Dourados

PONTEIROS

Bibliografias:

- Cap 7 – Linguagem C, Silvio do Lago Pereira
- Apostila: Linguagem C: Descomplicada, A.R. Backes

Dourados – MS

Objetivos:

Desenvolver a habilidade de manipular endereços na memória; relação entre ponteiros e matrizes.

Agenda:

- Definição de Ponteiro
- Inicialização de Ponteiro
- Aritmética de Ponteiros
- Alocando e Liberando Memória
- Aplicações de Ponteiros
- Atividade

DEFINIÇÃO DE PONTEIRO

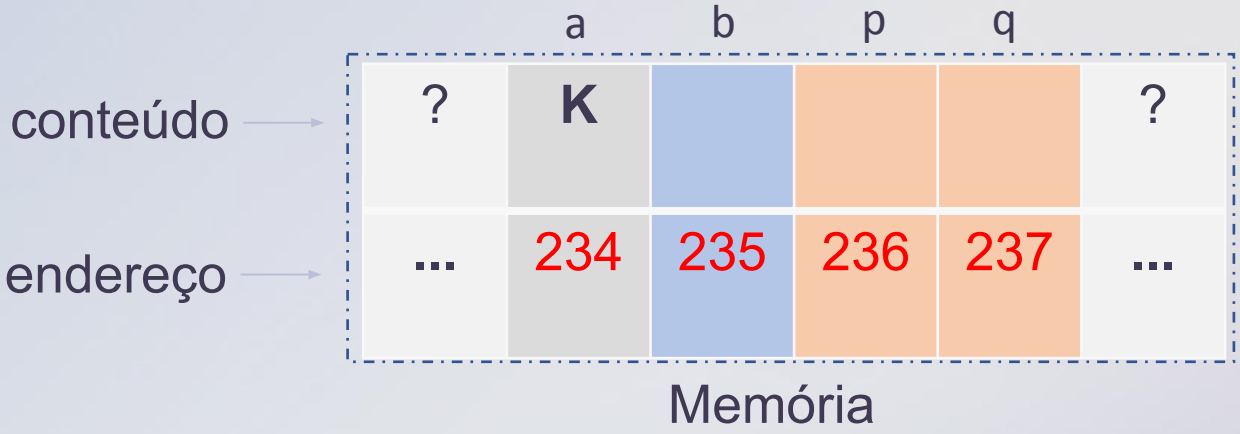
Ponteiro (ou apontador) é um tipo de variável que armazena apenas endereço de memória.

O operador de endereço (&) obtém o endereço de memória da variável.

O operador de redireção ou de indireção (*) acessa o conteúdo da memória apontado pelo ponteiro.

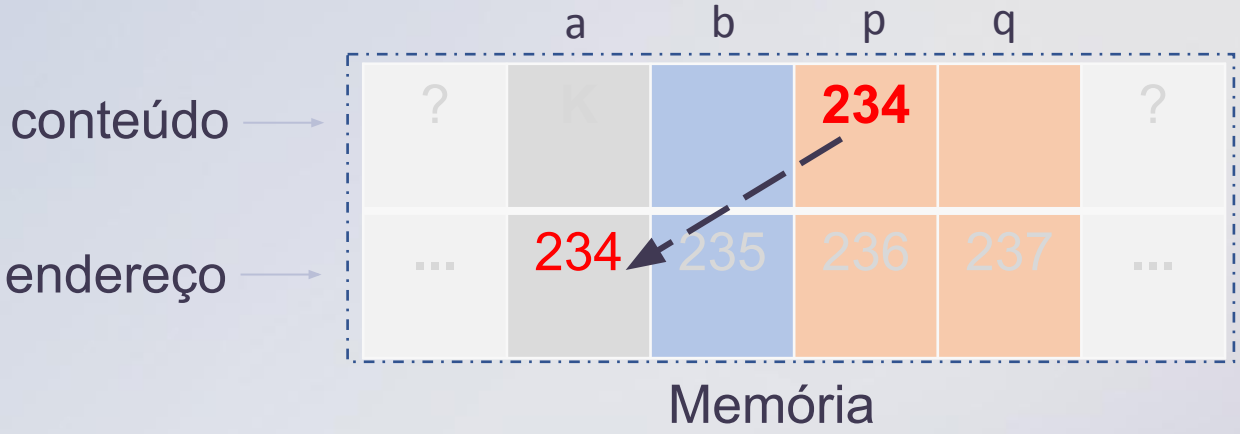
```
char a = 'K', b;
char *p, *q;//ponteiros para caracteres
p = &a; //p recebe o endereço de a
q = p; //p e q apontam para a
b = *p;//b recebe o conteúdo apontado por p
```

DEFINIÇÃO DE PONTEIRO



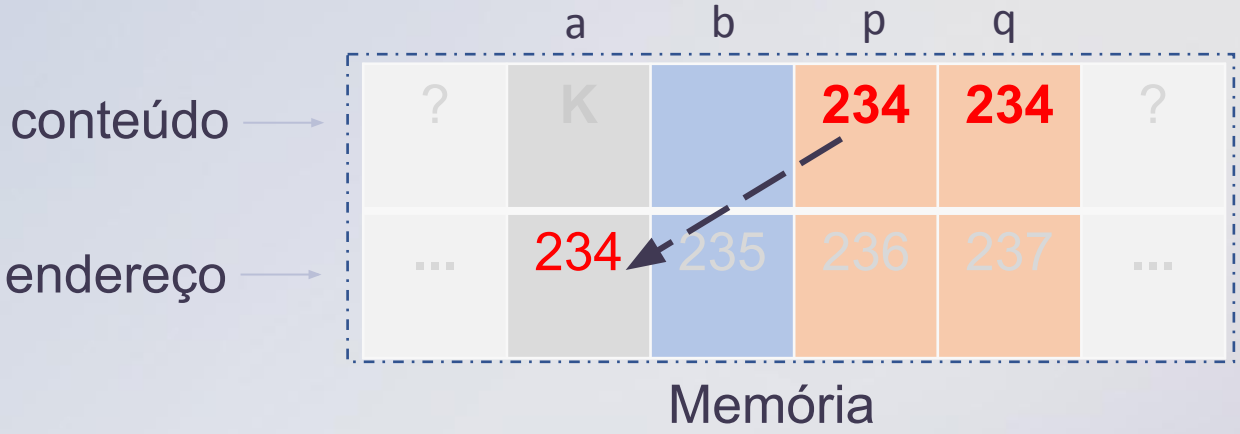

```
char a = 'K', b;
char *p, *q;//ponteiros para caracteres
p = &a; //p recebe o endereço de a
q = p; //p e q apontam para a
b = *p;//b recebe o conteúdo apontado por p
```

DEFINIÇÃO DE PONTEIRO



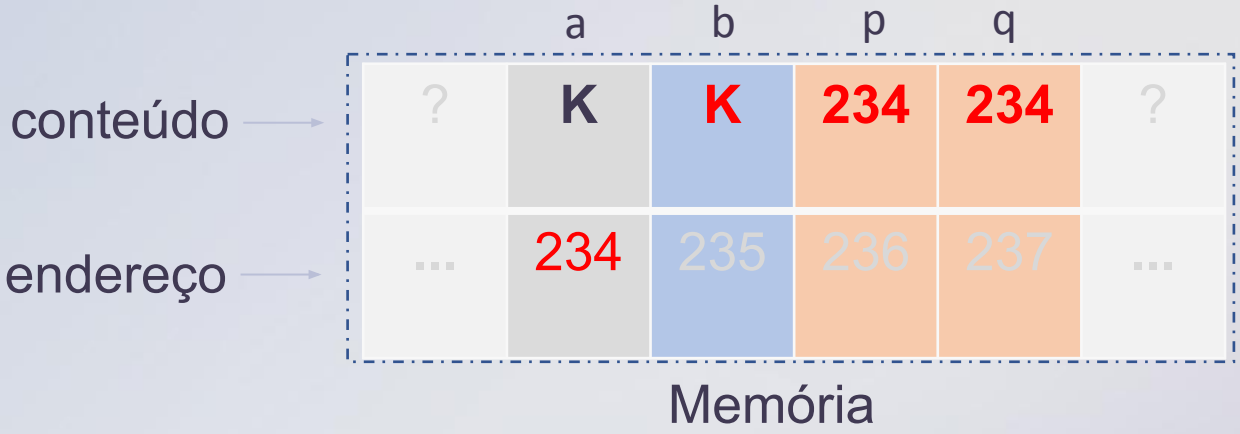

```
char a = 'K', b;
char *p, *q;//ponteiros para caracteres
p = &a; //p recebe o endereço de a
q = p; //p e q apontam para a
b = *p;//b recebe o conteúdo apontado por p
```

DEFINIÇÃO DE PONTEIRO



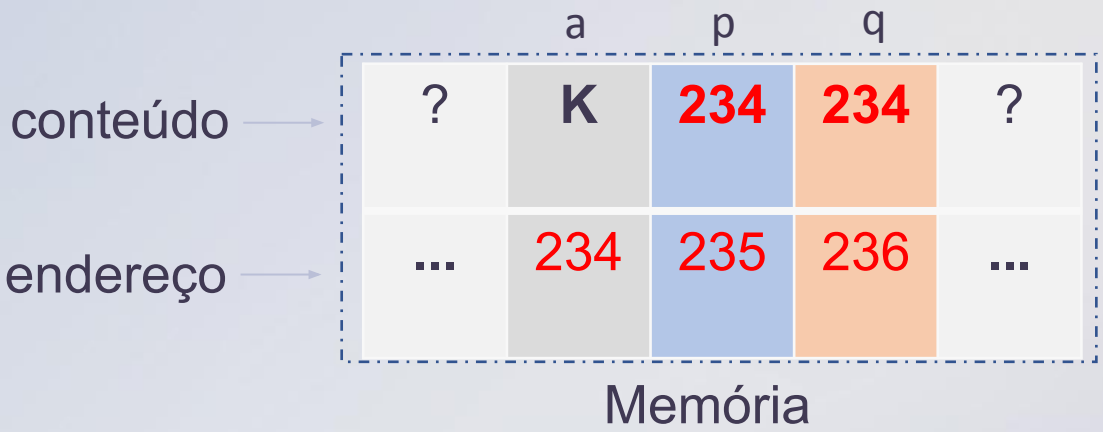

```
char a = 'K', b;
char *p, *q;//ponteiros para caracteres
p = &a; //p recebe o endereço de a
q = p; //p e q apontam para a
b = *p;//b recebe o conteúdo apontado por p
```

DEFINIÇÃO DE PONTEIRO




```
char a = 'K', b;
char *p, *q;//ponteiros para caracteres
p = &a; //p recebe o endereço de a
q = p; //p e q apontam para a
b = *p;//b recebe o conteúdo apontado por p
```

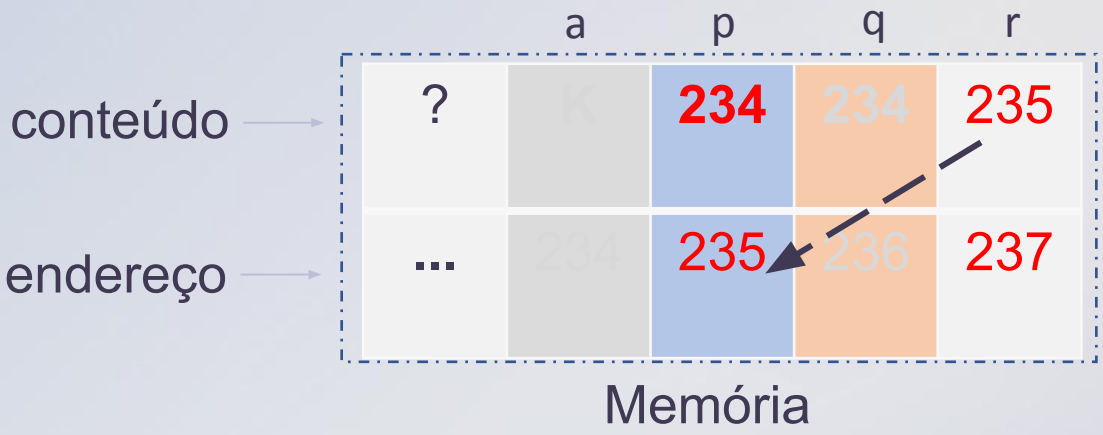
DEFINIÇÃO DE PONTEIRO




```
char a = 'K';
char *p, *q;
p = &a;
q = p;
```

Como fazer para um ponteiro apontar para outro ponteiro?

DEFINIÇÃO DE PONTEIRO




```
char a = 'K';
char *p, *q;
p = &a;
q = p;
```

```
char **r;
r = &p;
```

A variável `r` é um ponteiro para ponteiro para caracteres

(Um exemplo de aplicação é mostrado adiante, em alocação de memória)

INICIALIZAÇÃO DE PONTEIRO

Todo ponteiro deve ser inicializado, ou com o endereço de uma variável, ou com NULL

```
int a = 4, *p, *q;//p e q são ponteiros para inteiros
p = &a;
q = NULL;
```

Abra o arquivo **a_ponteiros_comente.c** e descreva o que ocorre em cada linha do programa onde está o //

ARITMÉTICA DE PONTEIROS

```
int num = 5, *p = &num;
```

p

endereço	num
471	5
472	
473	
474	
475	3
476	
477	
478	
...	?

Um `int` ocupa 4Bytes

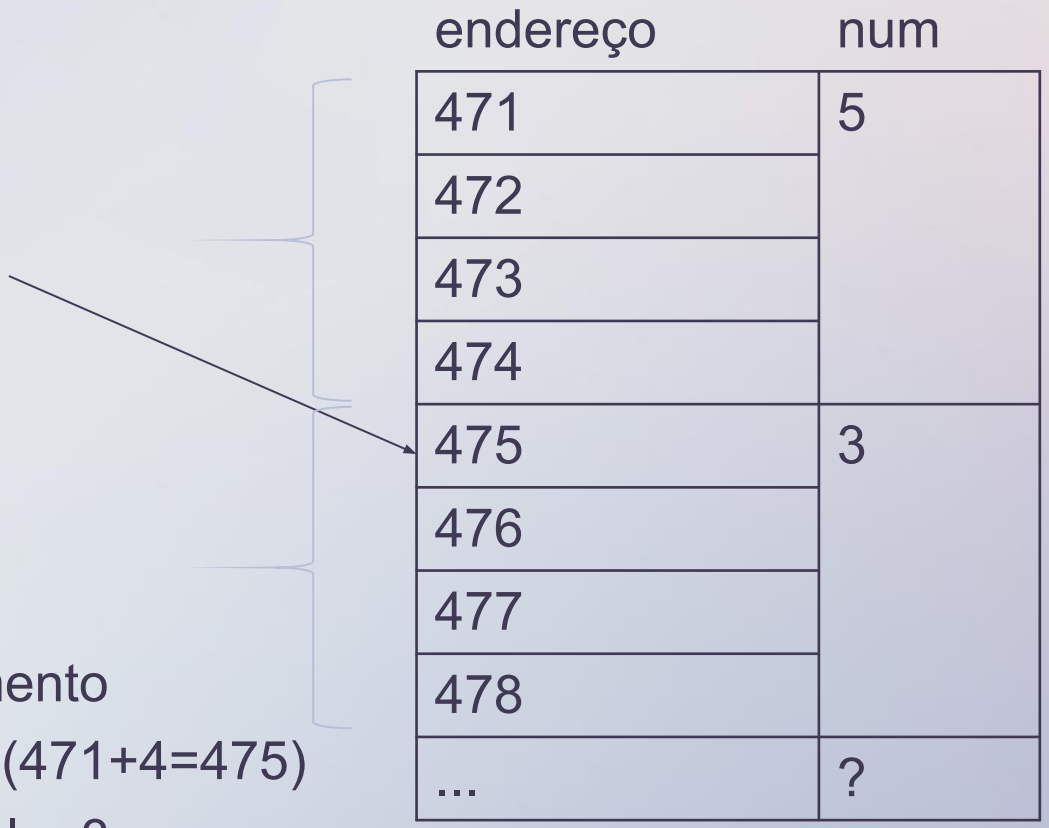
A variável `p` aponta para `num`

`*p` acessa o valor 5

ARITMÉTICA DE PONTEIROS

```
int num = 5, *p = &num;
```

p++

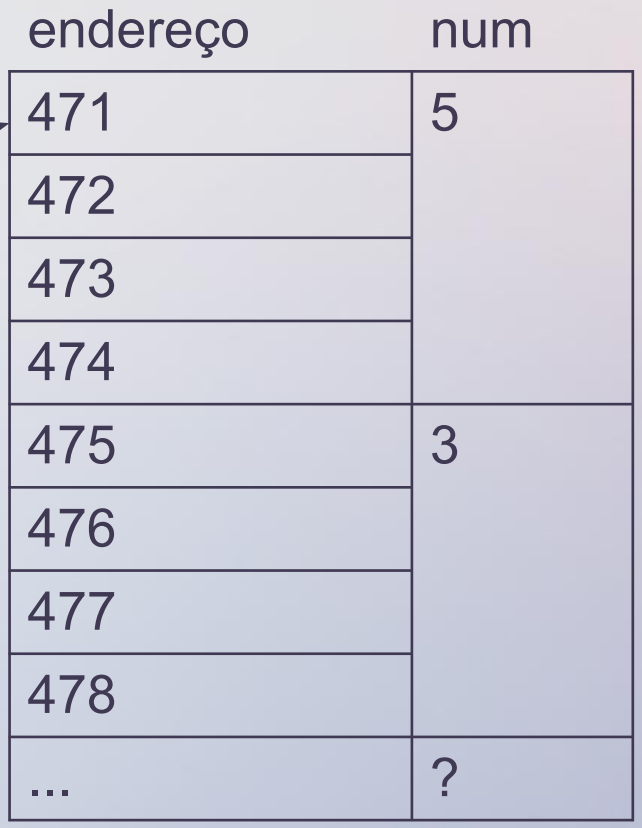


A operação de incremento
faz `p` avançar 4Bytes (471+4=475)
Agora `*p` acessa o valor 3

ARITMÉTICA DE PONTEIROS

```
int num = 5, *p = &num;
```

p--



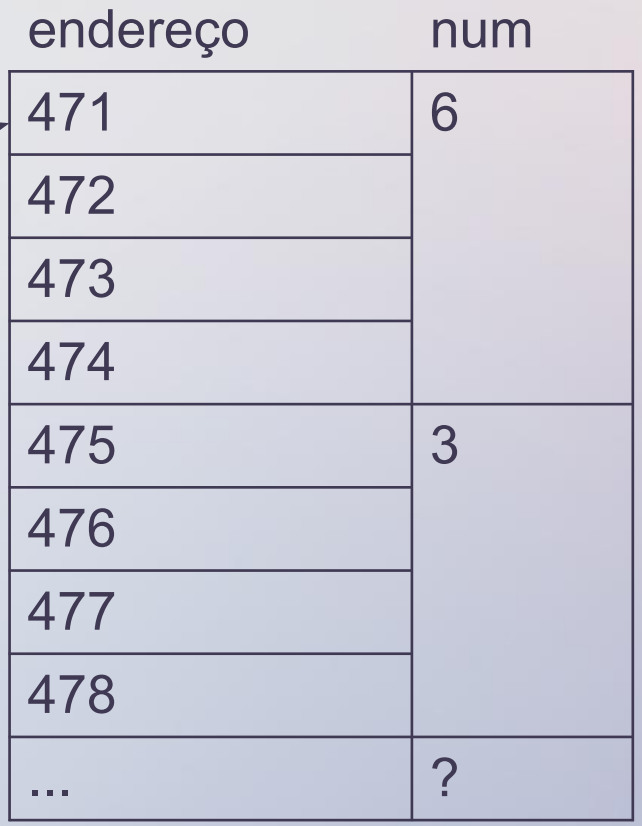
A operação de decremento faz `p` retroceder 4Bytes

Agora `*p` acessa o valor 5

ARITMÉTICA DE PONTEIROS

```
int num = 5, *p = &num;
```

```
(*p)++
```



A operação incrementa
o conteúdo apontado por `p`
Agora `*p` acessa o valor 6

ARITMÉTICA DE PONTEIROS

```
int num = 5, *p = &num;
```

*(p+1)

endereço	num
471	6
472	
473	
474	
475	3
476	
477	
478	
...	?

A operação acessa o conteúdo da próxima posição apontada por `p`

Mas `p` não foi incrementado

ARITMÉTICA DE PONTEIROS

- Relação entre Ponteiros e Matrizes

Em Linguagem C, a relação entre ponteiro e matriz é intrínseca

Uma matriz (unidimensional ou não) é uma coleção de elementos de mesmo tipo, alocados contiguamente na memória. Assim, ao declarar essa estrutura, o sistema reserva a quantidade de memória contígua, determinada pelo programa, e um ponteiro para o início dessa sequência de bytes é retornado para o nome dessa estrutura

Logo, operações envolvendo matrizes podem ser realizadas utilizando ponteiros e aritmética de ponteiros

ARITMÉTICA DE PONTEIROS

Veja os arquivos **b_aritmetica_ponteiro_vetor.c** e **c_aritmetica_ponteiro_matriz.c**

Abra o **d_aritmetica_ponteiro_comente.c** e descreva o que ocorre em cada linha do programa onde está o //

Obs.

- Operadores relacionais podem ser usados, desde que apontem para elementos do mesmo tipo
- Não é permitido somar, subtrair, multiplicar ou dividir ponteiros entre si

ALOCANDO E LIBERANDO MEMÓRIA

A alocação de mais de uma posição é sempre contígua.
Memória pode ser alocada de forma estática ou dinâmica.

Ex. Alocação estática:
int a; char v[]={‘a’, ‘b’, ‘c’};

Alocação estática define um tamanho fixo na memória, em tempo de compilação

Quando não se sabe o tamanho exato da quantidade de memória a ser utilizada, esta deve ser definida em tempo de execução, ou seja, dinamicamente, o que evita desperdício de memória, ou então a quantidade necessária pode ser superior àquela alocada previamente

ALOCANDO E LIBERANDO MEMÓRIA

A biblioteca stdlib deve ser usada para alocação de memória em tempo de execução

A função usada para alocar memória é a `malloc` (*memory alloc*):

```
void *malloc(unsigned int); //protótipo
```

Recebe como entrada a quantidade de bytes para alocar

Retorna um ponteiro genérico, que pode ser o ponteiro NULL, caso não encontre memória, ou o ponteiro para a primeira posição da memória alocada

ALOCANDO E LIBERANDO MEMÓRIA

Exemplo:

```
int *p = (int*)malloc(sizeof(int)*10);
```

São alocadas, contiguamente, 10 posições de memória do tipo `int` (isto é, de tamanho 4Bytes, cada posição), ou seja, 4Bytes*10 = 40Bytes alocados sequencialmente

O ponteiro `p` recebe a primeira posição da memória alocada. Para desalocar a memória, usar a função: `free(<ponteiro>);`

Lembrando: um ponteiro pode ser tratado como um vetor!

ALOCANDO E LIBERANDO MEMÓRIA

- Relação entre Ponteiros e Matrizes

Veja os arquivos a seguir para alocação dinâmica de memória, mostrando a relação entre ponteiros, matrizes e aritmética de ponteiros

- ✔ **e_malloc_vetor.c**
- ✔ **f_malloc_matriz_ponteiro_para_ponteiro.c**

APLICAÇÕES DE PONTEIROS

- ✔ Matrizes, com aritmética de ponteiros
- ✔ Alocação de memória em tempo de execução
- ✔ Passagem de parâmetro por referência
- ✔ Listas encadeadas, Árvores

(estruturas de dados dinâmicas, de modo geral)

Como você implementaria uma solução para uma pilha dinâmica, isto é, a quantidade de posições podendo aumentar/diminuir a cada operação de empilhar/desempilhar? Ficará mais claro quando forem abordados os conceito de Estrutura e de Lista Encadeada

ATIVIDADE

Faça uma solução, usando modularidade, para realizar as seguintes operações:

1. Inserir 10 valores em um vetor de inteiros
2. Duplicar os elementos do vetor
3. Imprimir os valores do vetor

Use a estrutura switch-case para a escolha das opções acima. O usuário decidirá quando encerrar o programa.

*Para o vetor de inteiros, alocar memória com a função malloc()

Quando escolher a opção para encerrar o programa, libere memória usando a função free()