

Aula 7

Ponteiros

Responsável

Prof. Armando Toda (armando.toda@usp.br)

Agenda

- **Tipos de Dados:**
 - **Alocação Estática de Memória**
 - **Alocação Dinâmica de Memória**
- **Endereços e Ponteiros**
 - **Alocação de Vetores e Matrizes**
 - **Alocação de Vetores de Caracteres (Strings)**
 - **Funções em “C” de alocação de memória:**
Calloc, Malloc

Alocação Estática:

- Espaço de memória é reservado previamente;
- Área de Tamanho FIXO e de Endereço FIXO (estática);
- Declarado e reservado em tempo de compilação

Exemplo: *variáveis globais* - declaradas fora do main/funções

Alocação Dinâmica:

- Espaço de memória é alocado em tempo de execução usando as funções “calloc” e “malloc” (dinâmica);
- Áreas de Tamanho VARIÁVEL e de Endereço VARIÁVEL são criadas (reserva memória) e destruídas (libera memória);
- Acesso usualmente através de Endereços e Ponteiros

Exemplo: *ponteiros+calloc/malloc* ou *variáveis locais*

Operadores de Endereço:

- Realizam operações com endereços de memória.

SINTAXE:

& Obtém o endereço de uma variável. Ponteiros são “tipados”

Exemplo:

```
int num;           /* Variável int */  
int *ptr_num;      /* Ponteiro para uma variável int */  
ptr_num = &num;    /* Obtém o endereço da variável int num */
```

***** Acessa o conteúdo de um endereço especificado.

Exemplo:

```
num = *ptr_num;    /* Atribui o valor contido no endereço  
                   apontado por ptr_num para num */
```

Operadores de Endereço:

- Realizam operações com endereços de memória.

SINTAXE:

& Obtém o endereço de uma variável. Ponteiros são “tipados”

Exemplo:

```
char letra;           /* Variável char */  
char *ptr_char;       /* Ponteiro para uma variável char */  
ptr_char = &letra;    /* Obtém o endereço da variável char letra */
```

***** Acessa o conteúdo de um endereço especificado.

Exemplo:

```
letra = *ptr_letra;    /* Atribui o valor contido no endereço  
                       apontado por ptr_char para letra */
```

Revisão de Ponteiros em "C"

Operadores de Endereço: Ponteiros

- Realizam operações com endereços de memória.

SINTAXE:

& ► Obtém o endereço de uma variável. Exemplo:

```
int dado=51;
```

```
int *x;
```

```
x = &dado;
```

***** ► Escreve/Lê o conteúdo do endereço especificado. Exemplo:

```
dado1 = *x; ou *x = dado1;
```

	100
51	101
	102
	103
	104
	105

Variável Dado End.

Ponteiros são TIPADOS: Apontam para um determinado tipo de dado

Declaração:

```
{ int *ptr_valint;
```

```
double *ptr_valreal; ...
```

Utilização:

```
*ptr_valint = dado_int;
```

```
/* escreve */
```

```
ptr_valreal = &dado_real;
```

```
/* aponta */
```

Revisão de Ponteiros em "C"

Operadores de Endereço: Ponteiros

Exemplos:

main ()

{

int vetor[10]; /* Vetor de Inteiros: vetor == &(vetor[0]) */

int dado;

int *ptr;

ptr = &dado /* Ptr obtém o endereço da variável dado */

ptr = 130; / Armazena o valor 130 no endereço, ou seja, na variável dado! */

ptr = vetor /* Ptr obtém o endereço inicial do vetor de inteiros */

ptr = 130; / Armazena o valor 130 no vetor[0] */

ptr++; /* Aponta para o próximo valor do vetor, o próximo inteiro */

ptr = 131; / Armazena o valor 131 no vetor[1] */

vetor = 1; / Armazena o valor 1 no vetor[0], pois vetor aponta p/ &(vetor[0]) */

(vetor+5) = 2; / Armazena o valor 2 no vetor[5] */

}

	100
dado	51
	102
	103
	104
	105

Variável Dado End.

Exercícios para Entregar

1. Seja o seguinte trecho de programa:

```
int i = 3, j = 5;  
double x = 0;  
int *p, *q;  
p = &i;  
q = &j;
```

Qual é o valor das seguintes expressões ?

- a) $p == \&i;$
- b) $*p - *q$
- c) $**\&p$
- d) $p = \&x$
- e) $7 * (*p) / (*q) + 7$

Exercícios para Entregar

2. Qual das instruções abaixo é correta para declarar um ponteiro para inteiro?

- a. `*int pti;`
- b. `*pti;`
- c. `&i;`
- d. `int_pti pti;`
- e. `int *pti;`

3. Na seqüência de instruções abaixo:

```
float f;  
float *pf;  
pf = &f;  
scanf("%f", pf);
```

- a. Efetuamos a leitura de f
- b. Não efetuamos a leitura de f
- c. Temos um erro de sintaxe
- d. Deveríamos estar usando `&pf` no `scanf`
- e. Nenhuma das opções anteriores

Passagem por valor:

```
#include <stdio.h>

void swap(int a,int b) {
    int temp;
    temp = b;
    b = a;
    a = temp;
}

int main() {
    int a,b;
    printf("Entre c/ valores:");
    scanf("%d %d",&a,&b);
    printf("a=%d b=%d \n",a,b);
    swap(a,b);

    printf("a=%d b=%d \n",a,b);
    return 0;
}
```

Passagem por referência:

```
#include <stdio.h>

void swap(int *a,int *b) {
    int temp;
    temp = *b;
    *b = *a;
    *a = temp;
}

int main() {
    int a,b;
    printf("Entre c/ valores:");
    scanf("%d %d",&a,&b);
    printf("a=%d b=%d \n",a,b);

    swap(&a,&b);

    printf("a=%d b=%d \n",a,b);
    return 0;
}
```

Ponteiros na Linguagem “C” :

- Quando usamos a rotina scanf já estamos usando ponteiros...

```
scanf (“%d”, &var_int);    /* Endereço de var_int */
```

```
scanf ("%s", var_string);   /* Vetor de char */
```

- Alocação de memória:

calloc - Aloca memória, zerando os dados

malloc - Aloca memória, sem inicializar os dados

free - Libera um bloco de memória alocada previamente

```
void *calloc ( <quantidade_elementos>, <tamanho_elemento> )
```

Exemplo: `tabela_inteiros = calloc(10, sizeof (int));`

```
void *malloc( <quantidade_elementos>, <tamanho_elemento> )
```

Exemplo: `tabela_inteiros = malloc(10,*sizeof (int));`

```
void free ( void *ponteiro)
```

Exemplo: `free (tabela_inteiros);`

Exemplos:

```
int *ptr_int;  
double *ptr_pf;
```

```
ptr_int = (int *) calloc ( 10 , sizeof (int) );           /* Aloca 10 inteiros em seqüência */  
ptr_pf = (double *) calloc (10, sizeof (double) );      /* Aloca 10 nros. tipo double */  
free (ptr_int);                                         /* Libera a área de memória alocada */
```

Fazendo a criação de um vetor de duas formas equivalentes:

```
int tabela[10];    /* Aloca memória: tabela aponta para o início do vetor */
```

ou

```
int *tabela;  tabela = (int *) calloc (10, sizeof(int));
```

Exemplo: Tabela[10]

```
#include <stdio.h>
#include <conio.h>

main (){
    double Tabela[10];

    int i;

    for (i=0; i < 10; i++)
    {
        printf("Dado %d = ",i);
        scanf ("%lf",&(Tabela[i]));
    }

    printf("\nDados Lidos:\n");
    for (i=0; i < 10; i++)
        printf("Dado %d = %.2lf\n",i, Tabela[i]);
}
```

Exemplo: *Tabela

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h> /* Inclui a biblioteca do "calloc" */

main (){
    double *Tabela; /* Cria somente o Ponteiro */
                    /* Sem alocar memoria */

    int i;

    Tabela=(double *)calloc(10,sizeof(double));

    for (i=0; i < 10; i++)
    {
        printf("Dado %d = ",i);
        scanf ("%lf",&(Tabela[i]));
    }

    printf("\nDados Lidos:\n");
    for (i=0; i < 10; i++)
        printf("Dado %d = %.2lf\n",i, Tabela[i]);

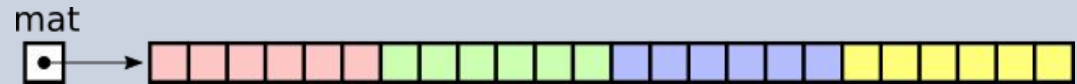
    free(Tabela);
}
```

Matriz estática

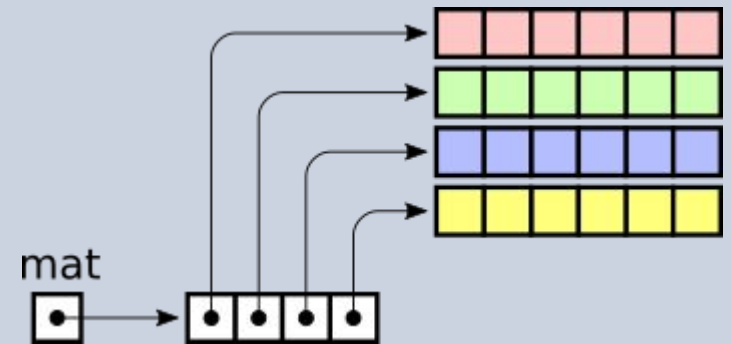
```
int matriz[10][10]
for(int i = 0; i < 10; i++){
    for(int j = 0; j < 10; j++){
        //leitura e manipulação de
        elementos da matriz
    }
}
```

Matriz dinâmica

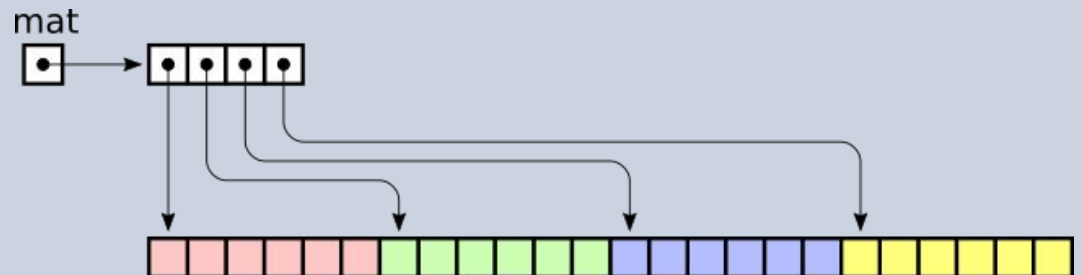
- Alocação Única

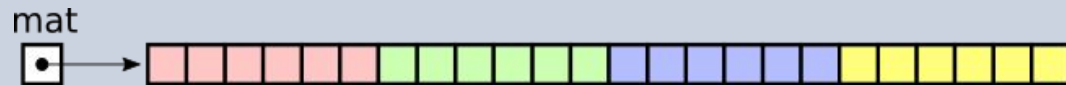


- Vetor de ponteiros



- Vetor de ponteiros de linhas contíguas





- Alocação Única

```
#define LIN 4
#define COL 6
```

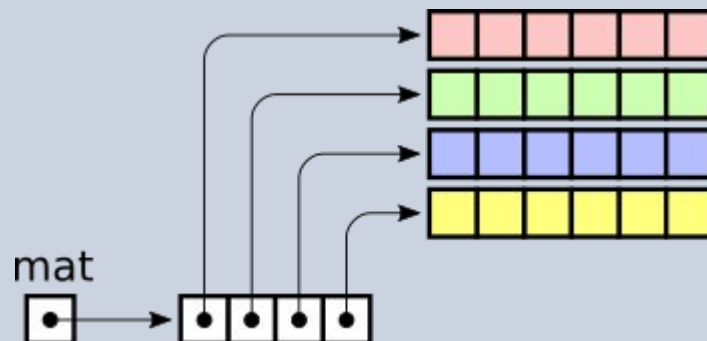
```
int *mat ;
int i, j ;
```

```
// aloca um vetor com todos os elementos da matriz
mat = malloc (LIN * COL * sizeof (int)) ;
```

```
// percorre a matriz
for (i=0; i < LIN; i++)
    for (j=0; j < COL; j++)
        mat[(i*COL) + j] = 0 ; // calcula a posição de cada
elemento
```

```
// libera a memória alocada para a matriz
free (mat) ;
```


- Vetor de ponteiros



```
#define LIN 4
```

```
#define COL 6
```

```
int **mat, i, j ;
```

```
// aloca um vetor de LIN ponteiros para linhas
```

```
mat = malloc (LIN * sizeof (int*)) ;
```

```
// aloca cada uma das linhas (vetores de COL inteiros)
```

```
for (i=0; i < LIN; i++)
```

```
    mat[i] = malloc (COL * sizeof (int)) ;
```

```
// percorre a matriz
```

```
for (i=0; i < LIN; i++)
```

```
    for (j=0; j < COL; j++)
```

```
        mat[i][j] = 0 ; // acesso com sintaxe mais simples
```

```
// libera a memória da matriz
```

```
for (i=0; i < LIN; i++)
```

```
    free (mat[i]) ;
```

```
free (mat) ;
```

- Vetor de ponteiros de linhas contíguas

```

#define LIN 4
#define COL 6
int **mat, i, j ;

// aloca um vetor de LIN ponteiros para linhas
mat = malloc (LIN * sizeof (int*)) ;
// aloca um vetor com todos os elementos da matriz
mat[0] = malloc (LIN * COL * sizeof (int)) ;
// ajusta os demais ponteiros de linhas (i > 0)
for (i=1; i < LIN; i++)
    mat[i] = mat[0] + i * COL ;
// percorre a matriz
for (i=0; i < LIN; i++)
    for (j=0; j < COL; j++)
        mat[i][j] = 0 ;
// libera a memória da matriz
free (mat[0]) ;
free (mat) ;

```

The diagram shows a variable `mat` pointing to a vector of 4 pointers. The first pointer points to the start of a 4x6 matrix. The subsequent 3 pointers point to the start of each row, which are contiguous in memory. The matrix is represented as a horizontal row of 24 colored boxes: 4 red, 6 green, 6 blue, and 8 yellow.

Exemplo: Matriz[10][10]

```
#include <stdio.h>
#include <conio.h>

main (){
    int Matriz[10][10];
    int i,j;

    for (i=0; i < 10; i++)
    {
        for (j=0; j < 10; j++)
        {
            printf("Dado %d %d = ",i, j);
            scanf ("%d",&(Matriz[i][j]));
        }
    }

    printf("\nDados Lidos:\n");
    for (i=0; i < 10; i++){
        for (j=0; j < 10;j++){
            printf("Dado %d %d = %.d \n",i, j,
                Matriz[i][j]);
        }
    }
```

Exemplo: *Matriz

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h> /* Inclui a biblioteca do "calloc" */

main (){
    int **Matriz; /* Cria somente o Ponteiro */
                /* Sem alocar memoria */

    int i,j;

    Matriz=malloc(10*sizeof(int));
        for (i=0; i < 10; i++){
            Matriz[i] = malloc(10 * sizeof(int));
        }

    for (i=0; i < 10; i++)
    {
        for (j=0; j < 10; j++)
        {
            printf("Dado %d %d = ",i, j);
            scanf ("%d",&(Matriz[i][j]));
        }
    }

    free(Tabela);
```

Exercício

```
int i, *p, *q;  
q = &i;  
*q = 15;  
p = malloc(1*sizeof(int));  
*p = 10;  
printf("%d %d %d\n", *p, *q, i);  
*q = *p;  
printf("%d %d %d\n", *p, *q, i);  
free(p);  
printf("%d %d %d\n", *p, *q, i);  
q = p;  
free(p);  
printf("%d %d %d\n", *p, *q, i);
```

Exercícios

1. Faça um programa para armazenar em memória um vetor de dados contendo 1500 valores do tipo *int*, usando a função de alocação dinâmica de memória CALLOC.
 - 1a) Faça um loop e verifique se o vetor contém realmente os 1500 valores inicializados com zero (conte os 1500 zeros do vetor).
 - 1b) Atribua para cada elemento do vetor o valor do seu índice junto a este vetor. Exibir na tela os 10 últimos elementos do vetor.
2. Faça um programa que pergunte ao usuário quantos valores ele deseja armazenar em um vetor de *doubles*, depois use a função MALLOC para reservar (alocar) o espaço de memória de acordo com o especificado pelo usuário. Use este vetor dinâmico como um vetor comum, atribuindo aos 10 primeiros elementos do vetor valores aleatórios (rand) entre 0 e 100. Exiba na tela os valores armazenados nos 10 primeiros elementos do vetor.