

Algoritmos e Programação II

Ponteiros e
Alocação Dinâmica

Profª Yorah Bosse

yorah.bosse@gmail.com

The logo of the Universidade Federal do Mato Grosso do Sul (UFMS) is located in the bottom left corner. It features a stylized graphic of vertical black lines of varying heights on the left, and a circular emblem with horizontal lines on the right. Below these elements, the letters 'UFMS' are displayed in a bold, black, sans-serif font, set against a light blue rectangular background.

UFMS

```
int main()
{
    int a,b;
    a = 10;
    b = 22;
    printf("Valor de a: %d", a);
    printf("\nValor de b: %d", b);

    printf("\nEndereco da variavel a: %d", &a);
    printf("\nEndereco da variavel b: %d", &b);

    return 0;
}
```

```
Valor de a: 10
Valor de b: 22
Endereco da variavel a: 2686788
Endereco da variavel b: 2686784_
```

O que são ponteiros?

O ponteiro nada mais é do que uma variável que guarda o endereço de uma outra variável.

Como declarar um ponteiro?

`int *px;` —————> **px é um ponteiro do tipo int**

Como atribuir um valor para um ponteiro?

```
int x = 5;      // declara uma variável x
int *px;        // declara uma variável do tipo ponteiro que aponta para
                // uma área onde será armazenado um número inteiro
px = &x;        // inicializa o ponteiro px com o endereço de x
*px = 10;       // altera o valor da variável x para 10
```

- O valor inicial de um ponteiro depois que ele é declarado é sempre 0 ou NULL.
- NULL não é um endereço válido.
- Logo, você deve sempre inicializar um ponteiro com um endereço.

```
int main()
{
    int a,b;
    int *pa, *pb;

    a = 10;
    b = 22;
    pa = &a;
    pb = &b;

    printf("\nValor da variavel contida em pa: %d", *pa);
    printf("\nValor da variavel contida em pb: %d", *pb);

    return 0;
}
```

```
Valor da variavel contida em pa: 10
Valor da variavel contida em pb: 22
```

- Explique o funcionamento do código abaixo:

```
int main()
{
    int x=4, y=7;
    int *px, *py;

    printf("\nx eh %d, y eh %d\n", x, y);

    px = &x;
    py = &y;

    *px = *px +10;
    *py = *py + 10;

    printf("\nx eh %d, y eh %d\n", x, y);
    return 0;
}
```

```
x eh 4, y eh 7
x eh 14, y eh 17
```

Deve-se utilizar o operador ‘&’ na chamada da função

```
int main(){  
    int x = 3,y= 7;  
    altera(&x, &y);  
    printf("Valor de x: %d, \nValor de y: %d", x, y);  
    return 0;  
}
```

Utiliza o ‘*’ para declarar o ponteiro na definição da função

```
altera(int *px, int *py)  
{  
    *px = 37;  
    *py = 59;  
}
```

```
int main()
{
    int valor, result;
    printf("\nDigite um numero inteiro: ");
    scanf("%d", &valor);

    modulo(&valor);

    printf("\nResultado: %d", valor);

    getch();
    return 0;
}
```

```
void modulo(int *num)
{
    if(*num < 0)
        *num =(*num) * (-1);

}
```


Passagem por valor:

```
#include <stdio.h>

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

int main(void){
    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;
}
```

Digite dois valores: 5 6
a=5 e b=6
a=5 e b=6

Passagem por referência:

```
#include <stdio.h>

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

int main(void) {
    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;
}
```

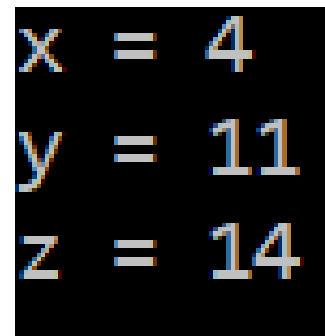
Digite dois valores: 5 6
a=5 e b=6
a=6 e b=5

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

```
int teste(int x1, int *y1, int z1){  
    z1 = x1 + z1;  
    *y1 = *y1 + x1;  
    x1 *= 2;  
    return *y1 + 3;  
}
```

```
int main(){  
    int x = 4,  
        y = 7,  
        z = 11;  
    z = teste(x, &y, z);  
    printf("x = %d\n",x);  
    printf("y = %d\n",y);  
    printf("z = %d\n",z);  
    return 0;  
}
```

Qual o resultado gerado pelo programa?



```
x = 4  
y = 11  
z = 14
```

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

```
int main(){  
    int x, *px, **ppx;  
    px = &x;  
    *px = 10;  
    ppx = &px;  
    **ppx = 50;  
    printf("%d %d", *px, x);  
    return 0;  
}
```

**Qual o resultado
gerado pelo programa?**



50 50

- Considere um programa que auxilia professores no cálculo da média semestral para uma turma de alunos.
- Qual a limitação do uso de um vetor, no qual cada elemento representa um aluno (matrícula, nota do primeiro bimestre, nota do segundo bimestre), para representar um grupo de alunos se este programa tiver como característica ser flexível o suficiente para lidar com tamanhos de turmas diferentes?

- Limitações da implementação com vetor

1) Superdimensionar um vetor

- Alocar mais espaço do que aquele realmente necessário para armazenar a coleção
- Problema: desperdício de memória

2) Subdimensionar um vetor

- Alocar menos espaço do que aquele realmente necessário para armazenar a coleção.
- Problema: erro fatal

Como solucionar os problemas acima?

- **Definição**

As variáveis de um programa têm alocação dinâmica quando suas áreas de memória – não declaradas no programa – passam a existir durante execução, ou seja, o programa é capaz de criar novas variáveis enquanto executa

Como trabalhar com alocação dinâmica?

Para trabalhar com variáveis alocadas dinamicamente é necessário o uso de apontadores (ponteiros) e o auxílio de funções que permitam reservar (e liberar), em tempo de execução, espaços da memória para serem usados pelo programa

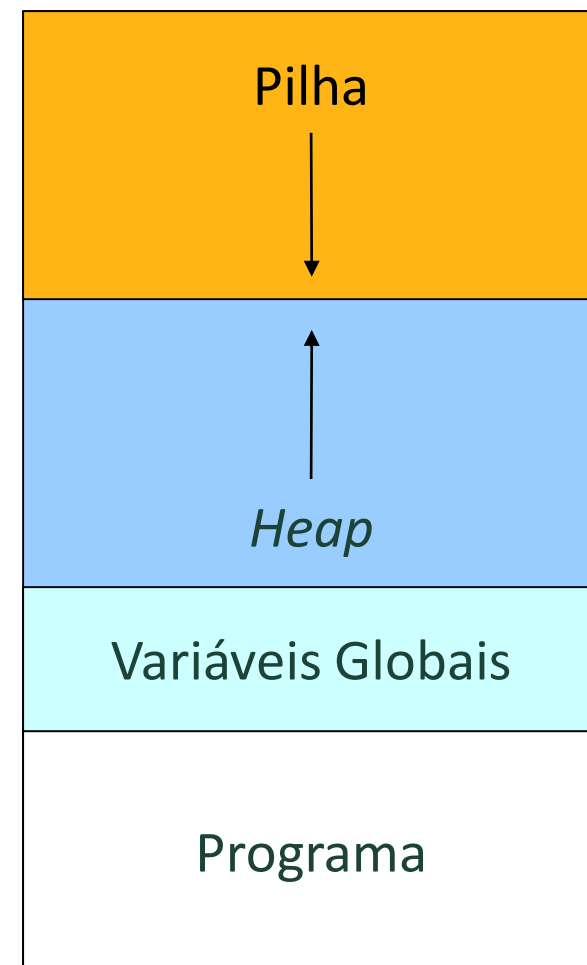
- Até o momento foi usada a memória principal por meio de variáveis locais e globais – incluindo arrays e estruturas
 - Variáveis globais: o armazenamento é fixo durante todo o tempo de execução do programa
 - Variáveis locais: o armazenamento é feito na pilha do computador
- Este tipo de uso da memória exige que o programador saiba, de antemão, a quantidade de armazenamento necessária para todas as situações

Para oferecer um meio de armazenar dados em **tempo de execução**, há um subsistema de **alocação dinâmica**

O armazenamento de forma dinâmica é feito na região de memória livre, chamada de **heap**

A pilha cresce em direção inversa ao *heap*

Conforme o uso de variáveis na pilha e a alocação de recursos no *heap*, a memória poderá se esgotar e um novo pedido de alocação de memória falhar



No núcleo do sistema de alocação dinâmica em C estão as funções:

→ **malloc()**

→ **free()**

Quando **malloc()** é usada, uma porção de memória livre é alocada

Quando **free()** é usada, uma porção de memória alocada é novamente devolvida para o sistema

Os protótipos das funções de alocação dinâmica estão na biblioteca **stdlib.h**

- Função malloc

```
void * malloc(size_t n);
```

Número de bytes alocados



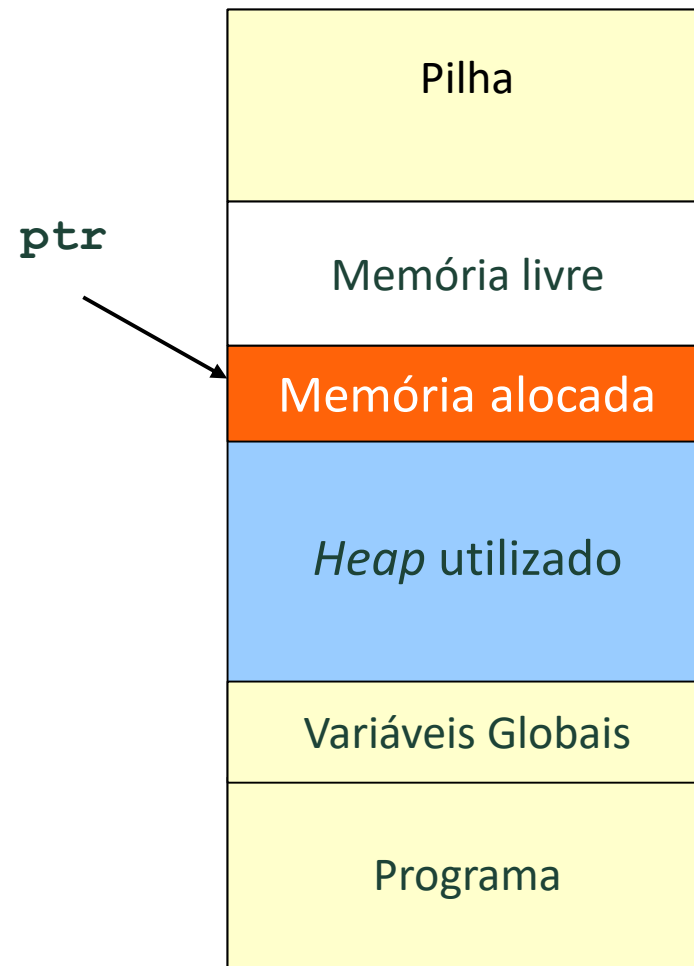
```
/*  
  retorna um ponteiro void para n bytes de memória não  
  iniciados. Se não há memória disponível malloc retorna  
  NULL  
*/
```

- Exemplo de uso:

```
int *pi;  
pi= (int *) malloc (sizeof(int));  
// aloca espaço para um inteiro
```

```
ptr = malloc(size);
```

- A função **malloc()** devolve um ponteiro para o primeiro byte de uma região de memória do tamanho *size*
- Caso não haja memória suficiente, retorna nulo (**NULL**)



```
float *v;  
int n;  
printf("Quantos valores? ");  
scanf("%d", &n);  
v = (float *) malloc(n * sizeof(float) );  
  
if(v!=NULL)  
    // manipula a região alocada
```

- Uma porção de memória capaz de guardar n números reais (float) é reservada, ficando o apontador v a apontar para o endereço inicial dessa porção de memória.
- O cast da função malloc() - (float *) - garante que o apontador retornado é para o tipo especificado na declaração do apontador. Certos compiladores requerem obrigatoriamente o cast.

- Conselho 1: “Utilize apontadores auxiliares para realizar operações (leitura, escrita) dentro do bloco de memória. Desta forma poderá sempre saber onde começa o bloco de memória dinâmica reservado, sem alterar o valor do apontador que recebeu o retorno da função malloc().”

```
float *v, *paux;  
int n;  
printf("Quantos valores? ");  
scanf("%d", &n);  
v = (float *) malloc(n * sizeof(float));  
if(v != NULL)  
    paux = v;
```

Na prática, não é muito usado.

- Conselho 2: ou utilize-o como um vetor

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

```
int main(){
    int n,*vet,i;
    printf("Digite quantos valores deseja
           armazenar : ");
    scanf("%d", &n);
    vet = (int *) malloc (n * sizeof(int));
    if (vet == NULL)
        exit(1);

    for(i = 0; i < n; i++){
        printf("Digite o %do numero : ",i+1);
        scanf("%d",&vet[i]);
    }
```

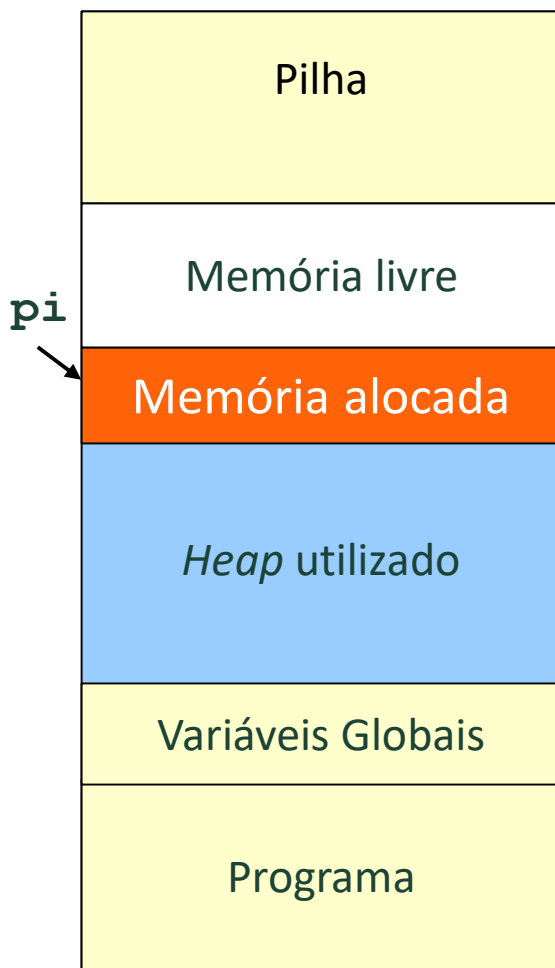
```
system("cls");

for(i = 0; i < n; i++)
    printf("%do numero = %d\n",i+1,vet[i]);

free(vet);

system("pause");
return 0;
}
```

- A função `free` é usada para liberar o armazenamento de uma variável alocada dinamicamente



```
int *pi;  
pi= (int *) malloc (sizeof(int));  
...  
free(pi);
```

- A função **free()** devolve ao *heap* a memória apontada pelo ponteiro **ptr**, tornando a memória livre
- Deve ser chamada apenas com ponteiro previamente alocado



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

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

```
int main ()
```

```
{
```

```
    int *p;
```

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

```
    if ( p == NULL )
```

```
    {
```

```
        printf("Não foi possível alocar memória.\n");
```

```
        exit(1);
```

```
    }
```

```
    *p = 5;
```

```
    printf("%d\n", *p);
```

```
    free(p);
```

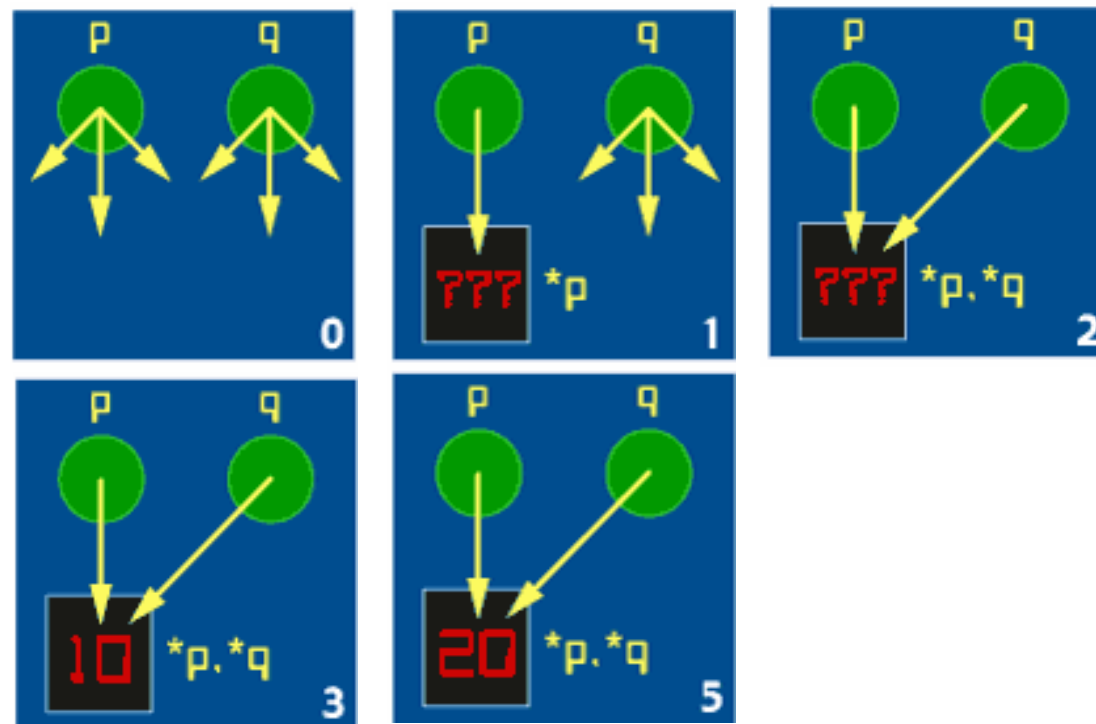
```
    system("pause");
```

```
    return 0;
```

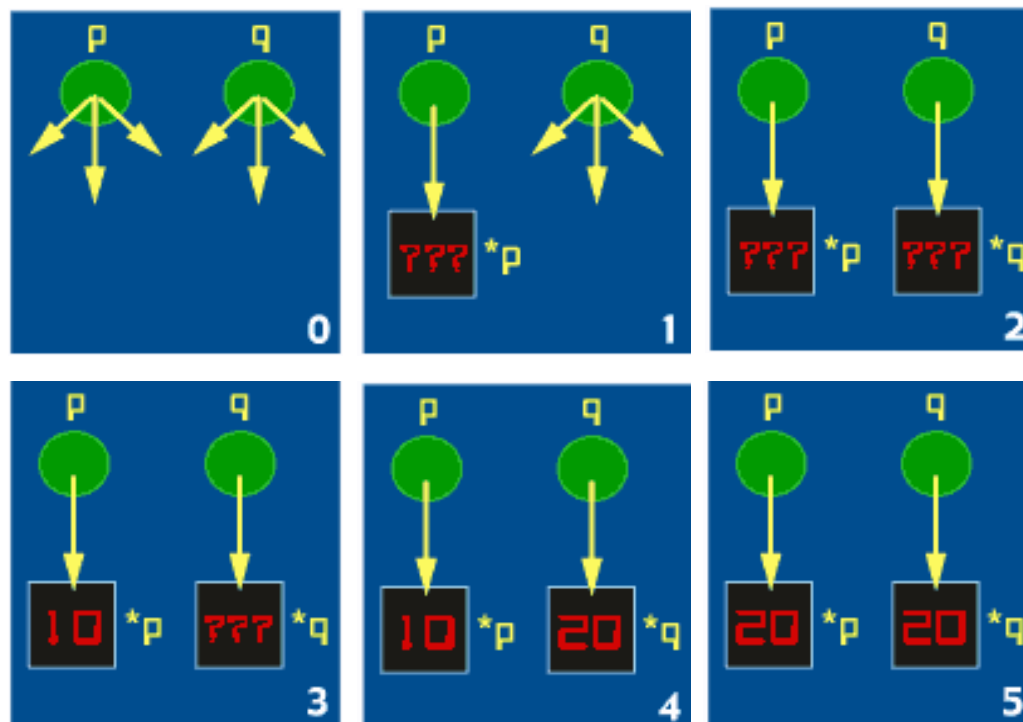
```
}
```



```
int main ()  
{  
    int *p, *q;  
    p = (int *) malloc(sizeof(int));  
    q = p;  
    *p = 10;  
    *q = 20;  
    free(p);  
    q = NULL;  
    return 0;  
}
```



```
int main (void) {
    int *p, *q;
    p = (int *) malloc(sizeof(int));
    q = (int *) malloc(sizeof(int));
    *p = 10;
    *q = 20;
    *p = *q;
    free(p);
    free(q);
    return 0;
}
```



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

struct sEndereco {
    char rua[20];
    int numero;
};

int main(){
    struct sEndereco *pend;
    int i;
    pend = (struct sEndereco *)malloc(2 * sizeof(struct sEndereco) );
    if ( pend == NULL ){
        printf("Não foi possível alocar memória.\n");
        exit(1);
    }
```

```
for(i = 0; i < 2; i++){
    printf("Digite o nome da rua : ");
    fflush(stdin);
    scanf("%20[^\n]s",pend[i].rua);
    printf("Digite o número : ");
    scanf("%d", &pend[i].numero);
}

system("cls");
for(i = 0; i < 2; i++)
    printf("Rua %s, %d\n\n",pend[i].rua,pend[i].numero);

free(pend);

system("pause");
return 0;
}
```

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

int main(){
    int *vet1;
    vet1 = (int *) malloc(100 * sizeof(int));
    if (vet1 == NULL ) {
        printf("Não foi possível alocar memória.\n");
        exit(1);
    }
    vet1[99] = 301;
    printf("%d\n", vet1[99]);
    free(vet1);
    system("pause");
    return 0;
}
```

```
int main () {  
    CLIENTE *pc;  
    pc = (CLIENTE *) malloc( 50 * sizeof(CLIENTE) );  
  
    gets( pc[0].nome );  
    scanf("%d", &pc[0].idade );  
  
    printf("%s", pc[0].nome);  
    printf("%d", pc[0].idade);  
  
    free(pc);  
    return 0;  
}
```

```
typedef struct cli  
{  
    char nome[30];  
    int idade;  
} CLIENTE;
```

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

int main(){
    int l, c, **mat, il, ic;
    printf("Digite quantas linhas : ");
    scanf("%d", &l);
    printf("Digite quantas colunas : ");
    scanf("%d", &c);
    mat = (int **) malloc (l * sizeof(int *));
    if (mat == NULL)
        exit(1);
    for (il=0; il<l; il++){
        mat[il] =(int *) malloc (c * sizeof(int));
        if (mat[il]==NULL)
            exit(1);
    }
```

```
for(il = 0; il < l; il++){  
    for(ic = 0; ic < c; ic++){  
        printf("Digite o %do numero da %da linha: ",ic+1,il+1);  
        scanf("%d",&mat[il][ic]);  
    }  
}
```

```
system("cls");
```

```
for(il = 0; il < l; il++){  
    for(ic = 0; ic < c; ic++){  
        printf("%d\t",mat[il][ic]);  
    }  
    printf("\n");  
}
```

```
for (il=0; il<l; il++){  
    free(mat[il]);  
}  
free(mat);  
system("pause");  
return 0;  
}
```


- 1) Faça um programa que leia dois números inteiros X e Y , armazenando-os em memória alocada dinamicamente e depois escreva na tela a soma deles no seguinte formato:
 $X + Y = Z$