

ALGORITMOS E PROGRAMAÇÃO II

Prof. Viviane Bonadia dos Santos

SOBRE O QUE NÓS FALAMOS NA AULA
PASSADA?



ALOCACÃO DINÂMICA

Em C podemos escrever código que gerencia a memória de forma dinâmica.

O programa pode **aumentar** ou **diminuir** a quantidade de memória durante a execução do programa.

COMO PODEMOS ALOCAR MEMÓRIA EM TEMPO DE EXECUÇÃO???

1. Precisamos de um ponteiro;

COMO PODEMOS ALOCAR MEMÓRIA EM TEMPO DE EXECUÇÃO???

1. Precisamos de um ponteiro;
2. Precisamos invocar uma função de alocação de memória;

COMO PODEMOS ALOCAR MEMÓRIA EM TEMPO DE EXECUÇÃO???

1. Precisamos de um ponteiro;
2. Precisamos invocar uma função de alocação de memória;
3. Devemos armazenar o endereço da memória alocada em um ponteiro;

COMO PODEMOS ALOCAR MEMÓRIA EM TEMPO DE EXECUÇÃO???

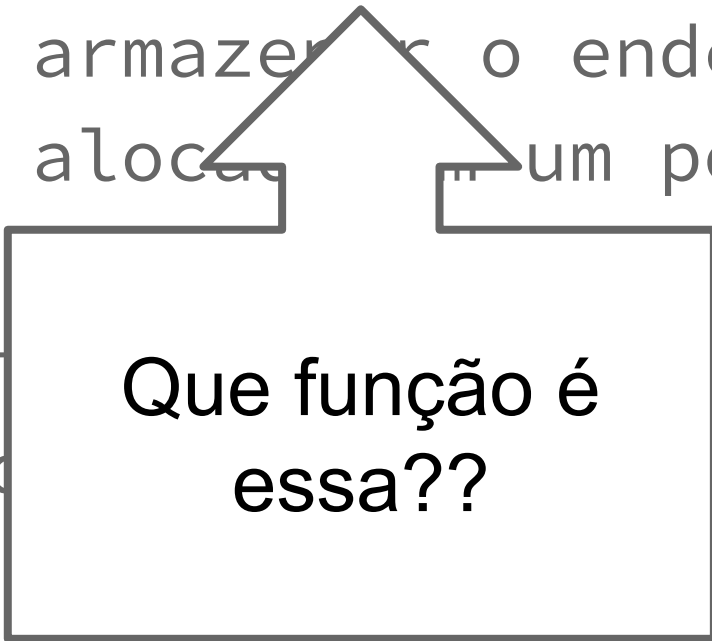
1. Precisamos de um ponteiro;
2. Precisamos invocar uma função de alocação de memória;
3. Devemos armazenar o endereço da memória alocada em um ponteiro;
4. Através do ponteiro podemos utilizar a memória alocada normalmente :)

COMO PODEMOS ALOCAR MEMÓRIA EM TEMPO DE EXECUÇÃO???

1. Precisamos de um ponteiro;
2. Precisamos invocar uma função de alocação de memória;
3. Devemos armazenar o endereço da memória alocada em um ponteiro.
4. Através do ponteiro podemos utilizar a memória alocada normalmente :)
5. Depois de usar a memória alocada devemos liberar ela!

COMO PODEMOS ALOCAR MEMÓRIA EM TEMPO DE EXECUÇÃO???

1. Precisamos de um ponteiro;
2. Precisamos invocar uma função de alocação de memória;
3. Devemos armazenar o endereço da memória alocada em um ponteiro.
4. Através da memória alocada, vamos utilizar a memória livremente :)
5. Depois de utilizarmos a memória alocada, devemos liberá-la.



Que função é essa??

COMO PODEMOS ALOCAR MEMÓRIA EM TEMPO DE EXECUÇÃO???

1. Precisamos de um ponteiro;
2. Precisamos invocar uma função de alocação de memória;
3. Devemos armazenar o endereço da memória alocada em um ponteiro.
4. Através da memória alocada, podemos utilizar a memória livremente :)
5. Depois de não precisarmos mais da memória alocada, devemos liberá-la.



**malloc, calloc ou
realloc**

COMO PODEMOS ALOCAR MEMÓRIA EM TEMPO DE EXECUÇÃO???

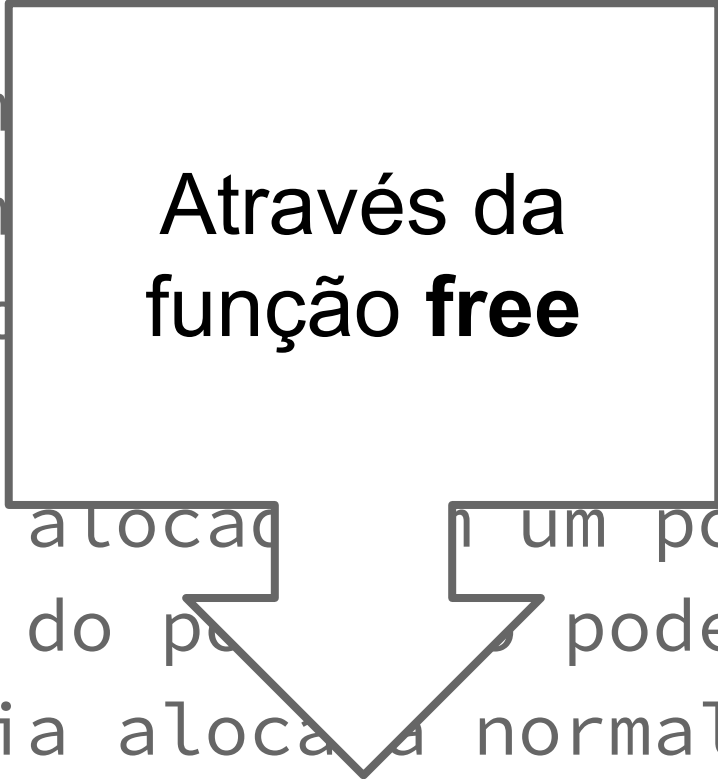


Como liberar memória?

1. Precisamos saber o tamanho da memória que vamos alocar;
2. Precisamos saber qual função de alocação vamos utilizar;
3. Devemos guardar o endereço da memória alocada em um ponteiro.
4. Através do ponteiro podemos utilizar a memória alocada normalmente :)
5. Depois de usar a memória alocada devemos liberar ela!

COMO PODEMOS ALOCAR MEMÓRIA EM TEMPO DE EXECUÇÃO???

1. Precisamos saber o tamanho da memória que vamos alocar;
2. Precisamos saber qual função de alocação vamos utilizar;
3. Devemos saber o endereço da memória alocada e guardar em um ponteiro.
4. Através do ponteiro podemos utilizar a memória alocada normalmente :)
5. Depois de usar a memória alocada devemos liberar ela!



Através da
função **free**

MALLOC

“A função **malloc** recebe como argumento um número inteiro positivo que representa a quantidade de bytes de memória desejada. Ela solicita a memória ao sistema operacional e retorna um ponteiro **void** para o primeiro byte do novo bloco de memória que foi alocado.”

MALLOC

“A função **malloc** recebe um número inteiro que representa a quantidade de bytes de memória desejada. Ela solicita a memória ao sistema operacional e retorna um ponteiro **void** para o primeiro byte do novo bloco de memória que foi alocado.”

Para executar operações de aritmética de ponteiros a linguagem C precisa que o ponteiro tenha um **tipo**. Por isso temos que fazer um **cast** no retorno de malloc para o tipo de ponteiro.

MALLOC

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

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

```
void main(){
```

```
    int *ponteiro;
```

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

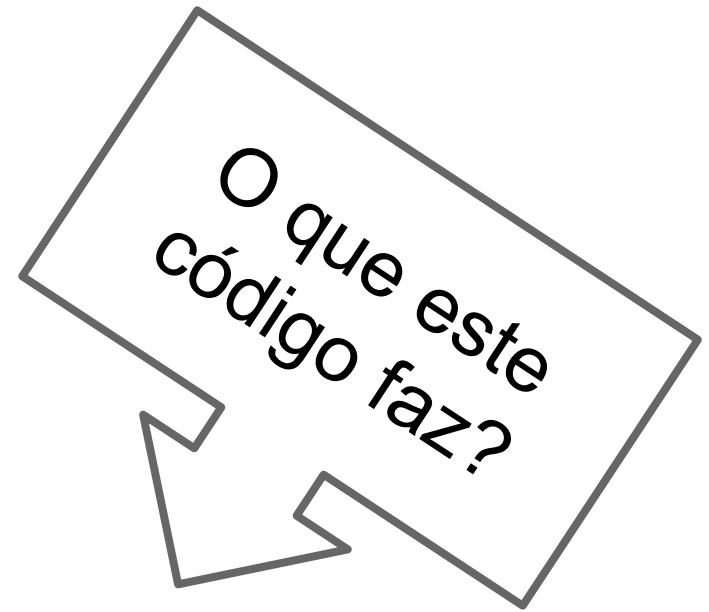
```
    printf("Digite um número: ");
```

```
    scanf("%d", ponteiro);
```

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

```
    free(ponteiro);
```

```
}
```



MALLOC

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

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

```
typedef struct{ int a, b; } Exemplo;
```

```
void main(){
```

```
    int *ponteiro;
```

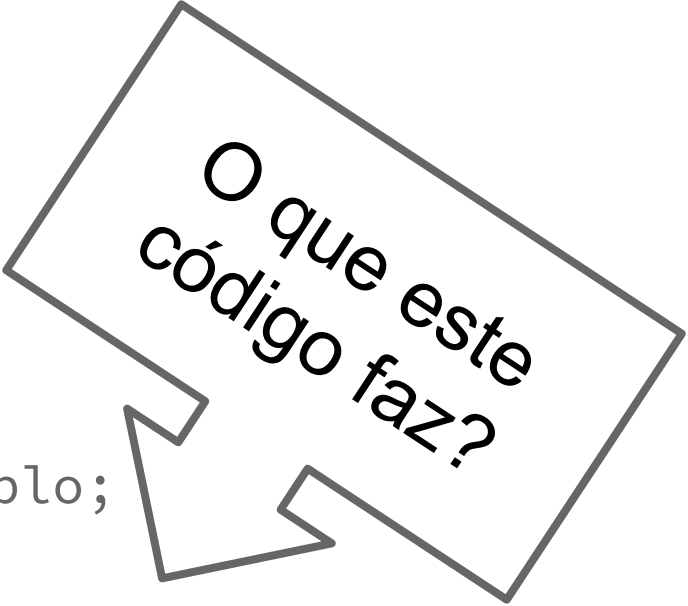
```
    ponteiro = (Exemplo *) malloc(sizeof(Exemplo));
```

```
    printf("Digite dois números: ");
```

```
    scanf("%d%d", &ponteiro->a, &ponteiro->b);
```

```
    free(ponteiro);
```

```
}
```



O que este
código faz?

MALLOC - ACESSANDO MEMBROS DE UM REGISTRO POR MEIO DE PONTEIROS

Você já aprendeu que se o nome de uma estrutura for conhecido, podemos acessar seus membros usando o operador ponto (█).

Será que uma construção análoga usando um ponteiro em vez do nome da variável, poderia ser escrita?



MALLOC - ACESSANDO MEMBROS DE UM REGISTRO POR MEIO DE PONTEIROS

Esta construção está errada, pois **p** não é uma variável estrutura e sim um ponteiro para uma variável estrutura. O operador ponto opera somente sobre o nome da estrutura.

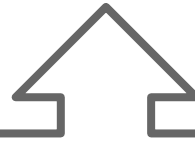
ser escrita?

p.a;



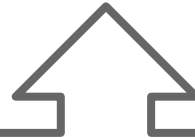
MALLOC - ACESSANDO MEMBROS DE UM REGISTRO POR MEIO DE PONTEIROS

`(*p).a; //OK`



Obtém o nome da variável apontada por p por meio do operador *

`p->a; //Ok. Mais usado.`



Este operador opera sobre o endereço de uma variável estrutura
enão sobre o seu nome.

MALLOC

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

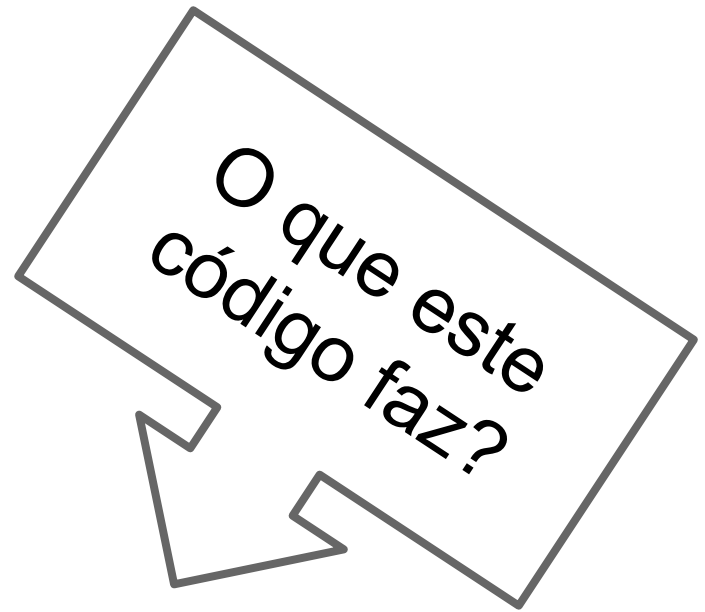
```
#include<stdlib.h>
```

```
void main(){
```

```
    int *v;
```

```
    v = (int *) malloc(30 * sizeof(int));
```

```
}
```



CALLOC

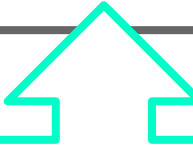
“A função **calloc** recebe dois números inteiros como argumentos: o primeiro indica o número de itens desejados e o segundo, o tamanho de cada item. Então retorna um ponteiro **void** apontando para o primeiro bloco de byte solicitado.”

```
int *v;  
  
v = (int *) calloc(30, sizeof(int));
```

REALLOC

“A função **realloc** serve para realocar o tamanho da memória previamente alocada.”

```
int *p;  
  
p = (int *) malloc(30 * sizeof(int));  
  
p = (int *) realloc(p, 80 * sizeof(int))
```



Um ponteiro para o bloco é devolvido porque **realloc** pode precisar mover o bloco para aumentar seu tamanho. Se isso ocorrer, o conteúdo do bloco antigo é copiado no novo bloco, e nenhuma informação é perdida.

MÉTODOS QUE RETORNAM PONTEIROS

```
int **misterio(int nlin, int ncol) {  
    int **m, i, j;  
    m = (int **) malloc(nlin*sizeof(int *));  
    if(m != NULL){  
        for (i = 0; i < nlin; i++)  
            m[i] = (int *)malloc(ncol*sizeof(int));  
    }  
    return(m);  
}
```