

## Algoritmo e Estrutura de Dados I

### Módulo 2 - Alocação Dinâmica de Memória

Prof<sup>ª</sup>. Elisa de Cássia Silva Rodrigues

Prof. Pedro Henrique Del Bianco Hokama

Prof<sup>ª</sup>. Vanessa Cristina Oliveira de Souza

## Introdução

- Para escrever programas muitas vezes é necessário definir variáveis.
- **Variáveis** são posições de memória que armazenam dados de um determinado tipo.
- Cada posição de memória possui um **endereço**.
- **Ponteiros** são variáveis que armazenam endereços de memória.
- Todo ponteiro também possui um tipo, ou seja, existe um dado de mesmo tipo armazenado naquela posição de memória para a qual o ponteiro aponta.

## Sumário

- Introdução.
- Ponteiros.
- Alocação de Memória.
- Alocação Estática.
- Alocação Dinâmica.
- Exemplos com Alocação Dinâmica.

## Ponteiros

- A linguagem C permite a manipulação de valores de endereços de memória através dos ponteiros.
- Para cada tipo de dado existente, há um *tipo ponteiro* que armazena um endereço de memória que contém aquele tipo de dado.
  - ▶ **Exemplo:**

```
int a; // armazena um dado do tipo inteiro
int *p; // armazena o endereço de memória que contém um inteiro
```
  - ▶ Para atribuir valores ao ponteiro *p*, temos duas formas:
    - ★ Operador unário **&** ("endereço de"): `p = &a;`
    - ★ Operador unário **\*** ("conteúdo de"): `*p = 10;`

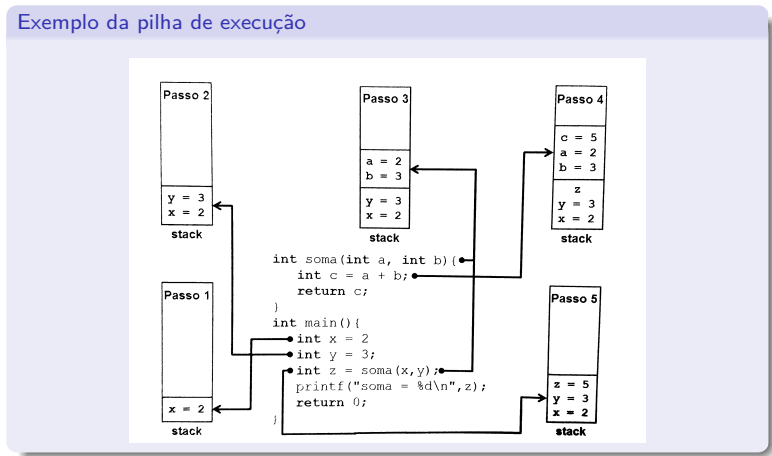
# Alocação de Memória

- **Alocação de memória** é processo de reserva de memória para armazenamento de dados durante a execução de um programa.
- A quantidade de memória pode ser **reservada automaticamente** como acontece quando declaramos uma variável ou um vetor estático:
  - ▶ `int x;` // são reservados 2 bytes de memória para armazenar um inteiro.
  - ▶ `int V[10];` // são reservados (10\*2) bytes de memória para armazenar 10 números inteiros sequencialmente na memória.
- Este processo é chamado de **Alocação Estática de Memória**.

# Alocação Estática

- Cada tipo de variável necessita de uma quantidade de memória que é automaticamente reservada na **Pilha de Execução**.
- **Vantagens:**
  - ▶ Os dados ficam armazenados sequencialmente na memória (vetores).
  - ▶ Programador não precisa se preocupar em gerenciar a memória.
- **Desvantagens:**
  - ▶ Programador não tem controle sob o tempo de vida das variáveis.
  - ▶ Quantidade de memória utilizada pelo programa é definida previamente.
  - ▶ Espaço reservado não pode ser alterado.
  - ▶ Podem haver espaços reservados desnecessariamente.

# Alocação Estática



# Alocação Estática

E quando a quantidade de memória necessária durante a execução do programa **NÃO** é previamente conhecida?



# Alocação Estática

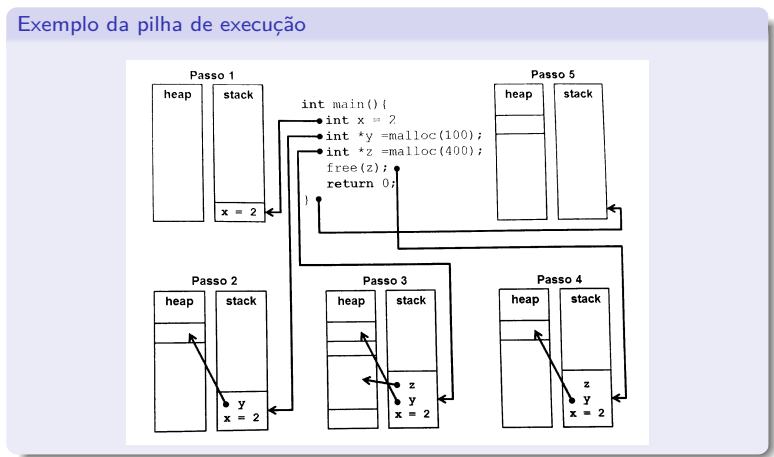
- Considere um problema onde necessita-se cadastrar o valor gasto por cada cliente de uma loja.
  - ▶ Se utilizarmos alocação estática, é preciso definir uma quantidade máxima de clientes já que não sabemos exatamente quantos clientes a loja terá.

```
float V[1000]; // máximo definido como 1000, por exemplo
```

- Problemas dessa solução:
  - ▶ Se precisar cadastrar mais de 1000 clientes o programa não servirá.
  - ▶ Se cadastrar poucos clientes haverá um desperdício de memória.

Solução: Alocação Dinâmica de Memória.

# Alocação Dinâmica



# Alocação Dinâmica

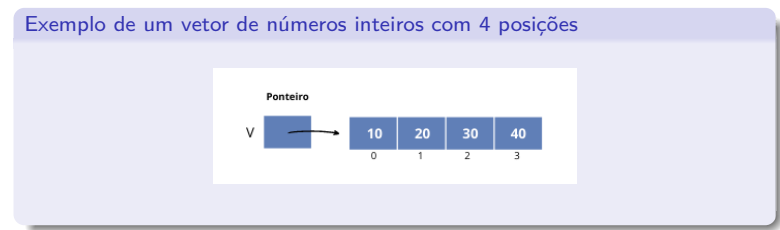
- A memória é **reservada dinamicamente** (em tempo de execução).
- Esta reserva não é feita na **Pilha de Execução**, mas em um outra área da memória (**Heap**).
  - ▶ Na linguagem C, a alocação é feita pela função `malloc()`.
  - ▶ Os dados desta área da memória só podem ser acessados por **ponteiros**.
- **Vantagens:**
  - ▶ Variáveis não dependem do escopo.
  - ▶ Quantidade total de memória não precisa ser previamente conhecida.
  - ▶ Espaço de memória pode ser alterado durante a execução do programa.
  - ▶ Programador controla o tempo de vida das variáveis.
- **Desvantagens:**
  - ▶ Os dados não são necessariamente armazenados de forma sequencial.
  - ▶ A memória utilizada deve ser alocada e liberada manualmente.
    - ★ **Obs:** esquecer de liberar a memória pode gerar falhas.

# Alocação Dinâmica

- **Vetores** (*arrays* unidimensionais):
  - ▶ Sequência de dados de mesmo tipo armazenados automaticamente na memória (alocação estática).


```
int V[4]; // pode-se dizer que o ponteiro V contém o endereço de V[0]
```
  - ▶ A linguagem C também permite definir um ponteiro para acessar um bloco de memória alocado dinamicamente.

```
int *v; // após ter definido o ponteiro, aloca-se memória para 4 inteiros
```
  - ▶ Para isso, utiliza-se as funções da linguagem C apresentadas a seguir.



## Alocação Dinâmica

- As funções da linguagem C usadas na alocação dinâmica de memória são encontradas na [biblioteca `stdlib.h`](#)<sup>1</sup>. São elas:
  - ▶ Operador `sizeof`.
  - ▶ Função `malloc`.
  - ▶ Função `free`.
  - ▶ Função `calloc`.
  - ▶ Função `realloc`.

<sup>1</sup>[https://www.tutorialspoint.com/c\\_standard\\_library/stdlib\\_h.htm](https://www.tutorialspoint.com/c_standard_library/stdlib_h.htm) 


COM111 - Estrutura de Dados | Módulo 2 - Alocação Dinâmica de Memória | 9 de setembro de 2021 | 13 / 21

## Alocação Dinâmica

- Função `malloc`<sup>1</sup>:
  - ▶ Usada para alocar memória durante a execução do programa.
  - ▶ Retorna um ponteiro que contém o endereço do início do espaço alocado na memória ou `NULL` em caso de erro (quando não tem memória suficiente para ser alocada).
    - ★ **Protótipo:** `void *malloc(unsigned int num)`.
  - ▶ **Exemplos:**

```
// alocação dinâmica de um vetor com 10 números inteiros
int *v = (int*) malloc(40); // quantidade de memória é 10 * 4 bytes

// mesma alocação usando o operador sizeof
int *v = (int*) malloc(10 * sizeof(int)); // recomendado
```

<sup>1</sup>[https://www.tutorialspoint.com/c\\_standard\\_library/c\\_function\\_malloc.htm](https://www.tutorialspoint.com/c_standard_library/c_function_malloc.htm) 

COM111 - Estrutura de Dados | Módulo 2 - Alocação Dinâmica de Memória | 9 de setembro de 2021 | 15 / 21

## Alocação Dinâmica

- Operador `sizeof`<sup>1</sup>:
  - ▶ Retorna o número de bytes necessários para alocar um único dado de determinado tipo.
  - ▶ **Sintaxe:** `sizeof(nome_do_tipo)`.
  - ▶ **Exemplos:**

```
sizeof(int) = 4 bytes.
sizeof(float) = 4 bytes.
sizeof(double) = 8 bytes.
sizeof(char) = 1 byte.
```

<sup>1</sup>[https://www.tutorialspoint.com/cprogramming/c\\_sizeof\\_operator.htm](https://www.tutorialspoint.com/cprogramming/c_sizeof_operator.htm) 


COM111 - Estrutura de Dados | Módulo 2 - Alocação Dinâmica de Memória | 9 de setembro de 2021 | 14 / 21

## Alocação Dinâmica

- Função `free`<sup>1</sup>:
  - ▶ Usada para liberar a memória alocada dinamicamente.
  - ▶ Se a memória alocada dinamicamente não for liberada corretamente, ela fica reservada e não poderá ser usada por outros programas.
    - ★ **Protótipo:** `void free(void *ptr)`.
  - ▶ **Exemplos:**

```
// alocação dinâmica de um vetor com 10 números inteiros
int *v = (int*) malloc(10 * sizeof(int));

// liberação da memória alocada
free(v);
```

<sup>1</sup>[https://www.tutorialspoint.com/c\\_standard\\_library/c\\_function\\_free.htm](https://www.tutorialspoint.com/c_standard_library/c_function_free.htm) 

COM111 - Estrutura de Dados | Módulo 2 - Alocação Dinâmica de Memória | 9 de setembro de 2021 | 16 / 21

## Alocação Dinâmica

- Função `calloc` <sup>1</sup>:

- ▶ Usada para alocar memória durante a execução do programa.
- ▶ Assim como o `malloc`, retorna um ponteiro que contém o endereço do início do espaço alocado na memória ou `NULL` em caso de erro (quando não tem memória suficiente para ser alocada).

- ▶ **Diferença:** inicializa todos os bits de espaço alocado com zero.

★ **Protótipo:** `void *calloc(unsigned int num, unsigned int size).`

- ▶ **Exemplo:**

```
// alocação dinâmica de um vetor com 10 números inteiros
int *v = (int*) calloc(10, sizeof(int));
```

<sup>1</sup>[https://www.tutorialspoint.com/c\\_standard\\_library/c\\_function\\_calloc.htm](https://www.tutorialspoint.com/c_standard_library/c_function_calloc.htm)

COM111 - Estrutura de Dados | Módulo 2 - Alocação Dinâmica de Memória | 9 de setembro de 2021 | 17 / 21

## Alocação Dinâmica

- **Observação:**

- ▶ Se a realocação falhar, a função `realloc` retornará `NULL` e a memória alocada anteriormente será perdida. Para que isso não ocorra, pode-se utilizar um ponteiro auxiliar para a realocação (`aux`).

```
// realocação aumentando o tamanho de v para 100 posições
int *aux = (int*) realloc(v, 100 * sizeof(int));
if(aux != NULL) v = aux; // caso contrário, v continua com tamanho 10
```

- **Outros exemplos:**

```
// realloc usado como equivalente ao malloc anterior
int *v = (int*) realloc(NULL, 10 * sizeof(int));
```

```
// realloc usado como equivalente a função free
v = (int*) realloc(v, 0);
```

COM111 - Estrutura de Dados | Módulo 2 - Alocação Dinâmica de Memória | 9 de setembro de 2021 | 19 / 21

## Alocação Dinâmica

- Função `realloc` <sup>1</sup>:

- ▶ Usada para alocar ou realocar memória durante a execução.
- ▶ Retorna um ponteiro que contém o endereço do início do espaço alocado na memória ou `NULL` em caso de erro (quando não tem memória suficiente para ser alocada).

★ **Protótipo:** `void *realloc(void *ptr, unsigned int num).`

- ▶ **Exemplos:**

```
// alocação dinâmica de um vetor com 10 números inteiros
int *v = (int*) malloc(10 * sizeof(int));

// realocação aumentando o tamanho de v para 100 posições
v = (int*) realloc(v, 100 * sizeof(int));
```

<sup>1</sup>[https://www.tutorialspoint.com/c\\_standard\\_library/c\\_function\\_realloc.htm](https://www.tutorialspoint.com/c_standard_library/c_function_realloc.htm)

COM111 - Estrutura de Dados | Módulo 2 - Alocação Dinâmica de Memória | 9 de setembro de 2021 | 18 / 21

## Exemplos com Alocação Dinâmica

- Alocação dinâmica de vetores:

<https://repl.it/community/classrooms/205600/assignments/5694395>

- Alocação dinâmica de matrizes:

<https://repl.it/community/classrooms/205600/assignments/5694397>

COM111 - Estrutura de Dados | Módulo 2 - Alocação Dinâmica de Memória | 9 de setembro de 2021 | 20 / 21

## Referências Bibliográficas

- 1 BACKES, A. *Linguagem C: Completa e Descomplicada*. 2013.

Vídeo aulas (60ª a 65ª):

<https://programacaodescomplicada.wordpress.com/indice/linguagem-c/>.

- 2 XAVIER, E. C. *Material Didático de MC102 (IC/UNICAMP)*.

Aula 20 (Ponteiros II):

[https://www.ic.unicamp.br/~eduardo/material\\_mc102/aula20.pdf](https://www.ic.unicamp.br/~eduardo/material_mc102/aula20.pdf).

Aula 21 (Ponteiros III):

[https://www.ic.unicamp.br/~eduardo/material\\_mc102/aula21.pdf](https://www.ic.unicamp.br/~eduardo/material_mc102/aula21.pdf).