Algoritmo e Estrutura de Dados I

Módulo 2 - Alocação Dinâmica de Memória

Profª. Elisa de Cássia Silva Rodrigues

Prof. Pedro Henrique Del Bianco Hokama

Profª. Vanessa Cristina Oliveira de Souza

Sumário

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.

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

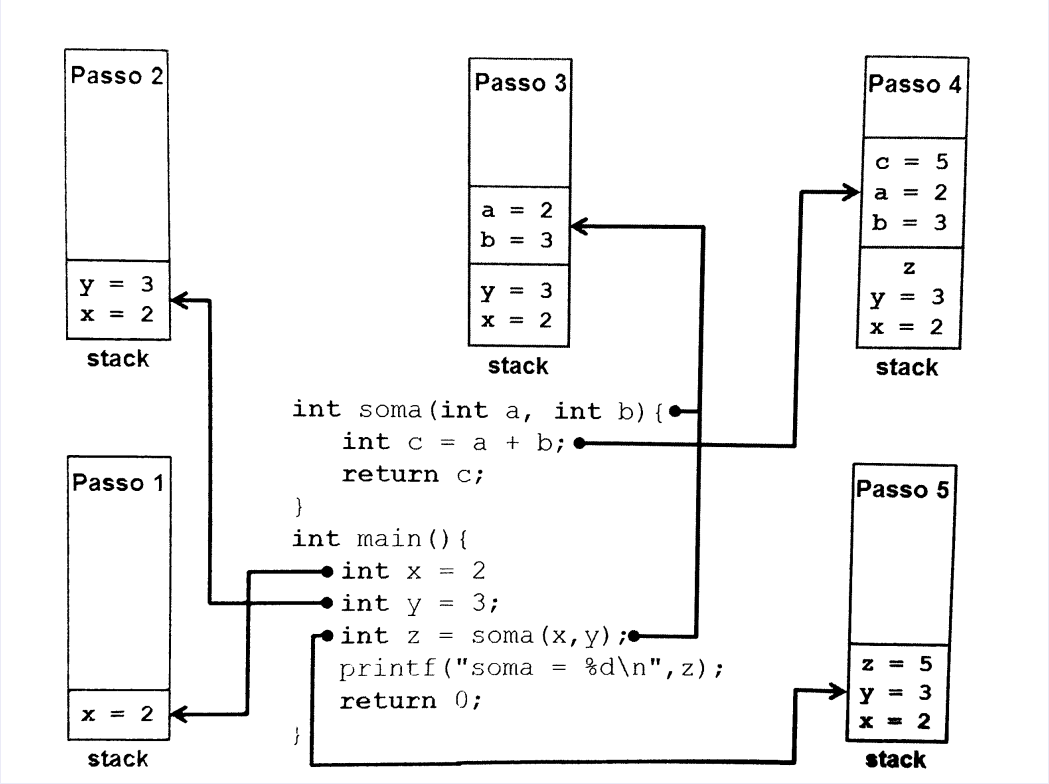
- Alocação de memória é processo de reserva de memória para armazenamento de dados durante a execução de um programa.
- A quantidade de memória pode ser reservada automaticamente como acontece quando declaramos uma variável ou um vetor estático:
  - `int x; // são reservados 2 bytes de memória para armazenar 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

Exemplo da pilha de execução



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-de 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.

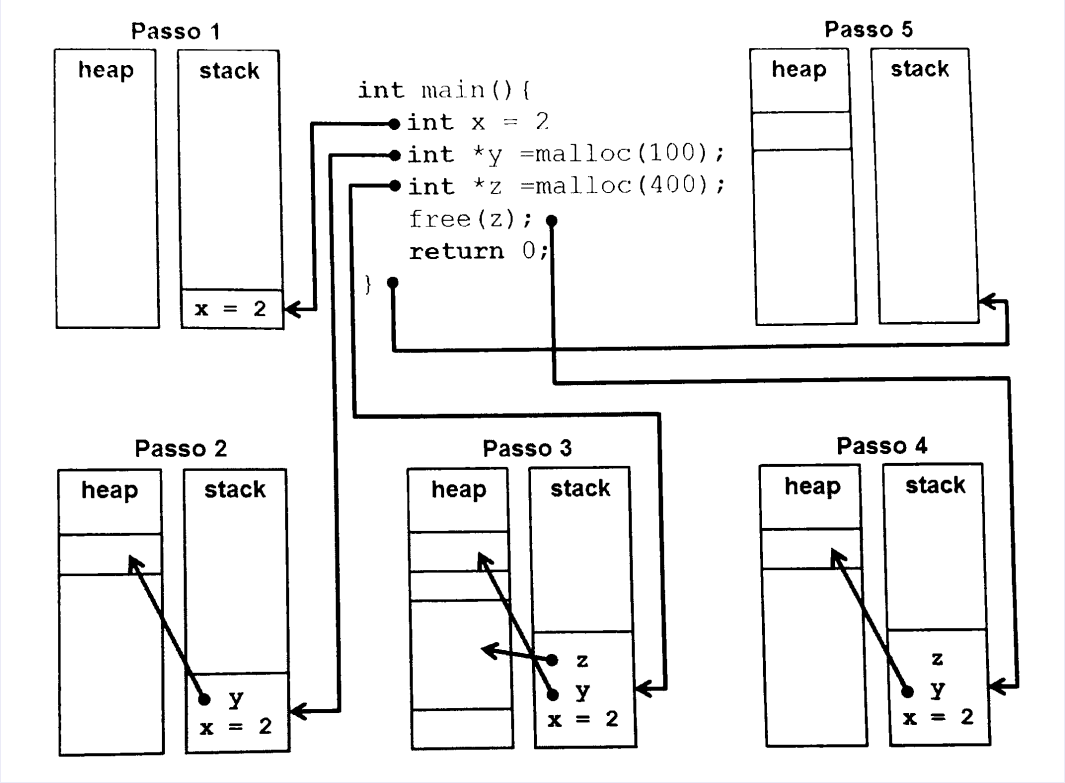
Solução: Alocação Dinâmica de Memória.

Alocação Dinâmica

- 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

Exemplo da pilha de execução



Alocação Dinâmica

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

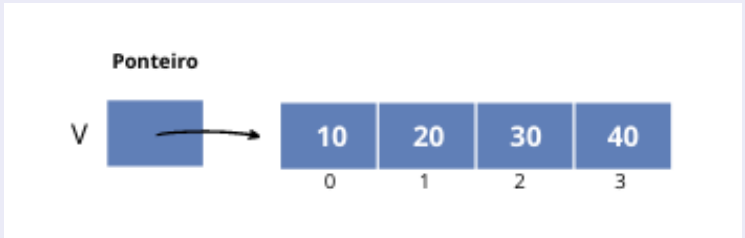
```
int V[4]; // pode-se dizer que o ponteiro V contém o endereço de V[0]
```

  - 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.

Exemplo de um vetor de números inteiros com 4 posições



Alocação Dinâmica

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

[1] https://www.tutorialspoint.com/c_standard_library/stdlib_h.htm

Alocação Dinâmica

- Operador `sizeof` [1]:
  - 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.

[1] https://www.tutorialspoint.com/cprogramming/c_sizeof_operator.htm

Alocação Dinâmica

- Função `malloc` [1]:
  - 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
```

[1] https://www.tutorialspoint.com/c_standard_library/c_function_malloc.htm

Alocação Dinâmica

- Função `free` [1]:
  - 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);
```

[1] https://www.tutorialspoint.com/c_standard_library/c_function_free.htm

Alocação Dinâmica

- Função `calloc` [1]:
  - 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));
```

[1] https://www.tutorialspoint.com/c_standard_library/c_function_calloc.htm

Alocação Dinâmica

- Função `realloc` [1]:
  - 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));
```

[1] https://www.tutorialspoint.com/c_standard_library/c_function_realloc.htm

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);
```

Alocação Dinâmica

- Matrizes (*arrays* multidimensionais):
  - Usa-se o conceito de ponteiro para ponteiro.

```
int *v; // matriz com 1 dimensão (vetor)
int **p; // matriz com 2 dimensões
int ***d; // matriz com 3 dimensões
```

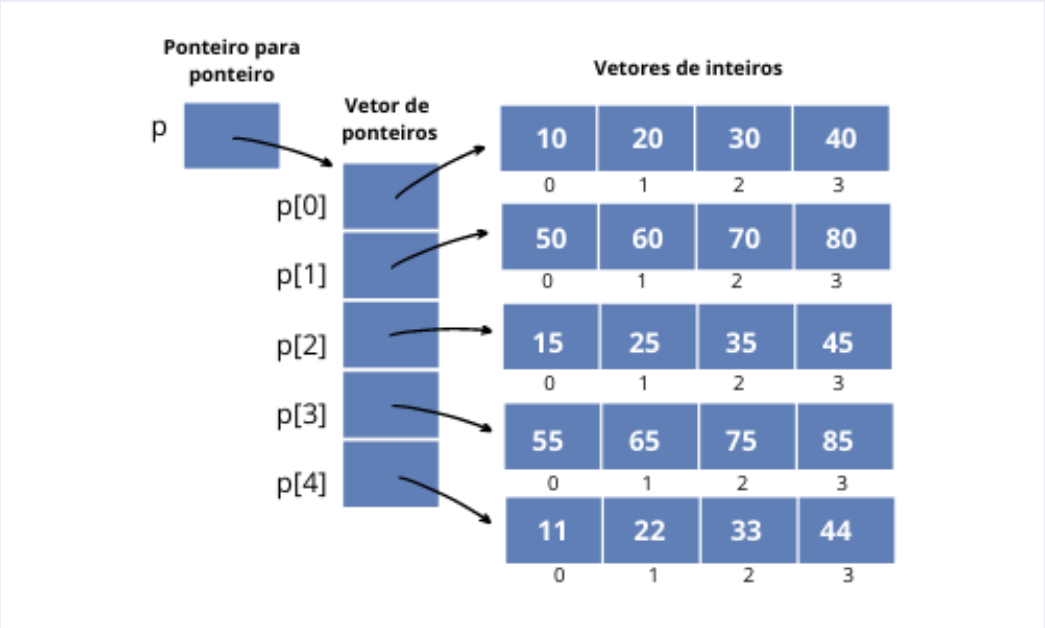
  - **Exemplo:**

```
// alocação dinâmica de uma matriz (m x n) de inteiros
int **p = (int**) malloc(m * sizeof(int*)); // vetor de ponteiros

// alocação dos vetores referentes a cada linha da matriz
for(int i=0; i<m; i++)
    p[i] = (int*) malloc(n * sizeof(int)); // vetor de inteiros
```

Alocação Dinâmica

Exemplo de uma matriz bidimensional de números inteiros com 5 linhas e 4 colunas



Alocação Dinâmica

- Matrizes (*arrays* multidimensionais):
  - Note que primeiro foi criado um vetor de ponteiros `p` (ponteiro para ponteiro) que representa a matriz bidimensional.
    - ★ Cada ponteiro `p[i]` desse vetor aponta para a primeira posição de um vetor de inteiros (alocado posteriormente) que representa a linha `i` da matriz.
  - Para liberação da memória, essa ordem deve ser inversa (antes de liberar `p`, deve-se liberar os ponteiros `p[i]`).
  - **Exemplo:**

```
// liberação de memória dos vetores de inteiros
for(int i=0; i<m; i++)
    free(p[i]);

// liberação de memória da matriz (vetor de ponteiros)
free(p);
```

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

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.

   Aula 21 (Ponteiros III):

   https://www.ic.unicamp.br/~eduardo/material_mc102/aula21.pdf.