Alocação Dinâmica de Memória

Algoritmos e Estruturas de Dados

Igor Montagner (igorsm1@insper.edu.br)

Alocação estática

```
int imagem[512][512];
```

- tamanho definido em tempo de compilação!
- e se precisar de mais espaço?

Alocação dinâmica (em Java)

```
int N = // lê da entrada ou de algum arquivo
int[][] imagem = new int[N][N];
```

- o valor de `N` muda conforme a entrada do usuário
- o `new` é responsável por "pedir" mais memória

Igor Montagner (igorsm1@insper.edu.br)

malloc

```c
#include <stdlib.h>

void *malloc(size_t size);
```

- **se deu certo** devolve um apontador para uma região de memória com ao menos `size` bytes
- **se deu erro** devolve `NULL`

Igor Montagner (igorsm1@insper.edu.br)

Alocação dinâmica (em C)

```c
int N;
scanf("%d", &N);
int* imagem = malloc(sizeof(int) * N);

// só usar como um array normal :)
```

- queremos `N` itens
- o tipo de cada item é `int`
- o tamanho de cada item é `sizeof(int)`

Igor Montagner (igorsm1@insper.edu.br)

Tipos de alocação dinâmica

- **explícito** (C, C++ e rust)
  - gerenciamento é feito no código
  - funções para pedir **e liberar** memória
- **implícito** (Java, Python, C#)
  - mais alto nível, alocando objetos inteiros
  - gerenciamento é feito pela própria linguagem

free

```c
#include <stdlib.h>

void free(void *p);
```

- Devolve o bloco apontado por `p` para o sistema
- `p` deve ter sido devolvido por uma chamada a `malloc`

Alocação dinâmica (em C)

```c
int N;
scanf("%d", &N);
int* dados = malloc(sizeof(int) * N);

// só usar como um array normal :)

free(dados);
```

- acessar os dados do apontador `dados` resulta em erro após a chamada a `free`
- se não chamar `free` a memória fica reservada para `dados` até o programa terminar

Igor Montagner (igorsm1@insper.edu.br)

Vantagens / Desvantagens (alocação explícita)

- **Vantagens**
  - cresce/diminui consumo de memória conforme necessário
  - controle da quantidade é feita em tempo de execução
  - útil em sistemas com pouca memória
- **Desvantagens**
  - gerenciar uso é trabalhoso e sujeito a erros
  - não liberar espaços sem uso
  - acessar espaços já liberados
  - pedir a quantidade exata de memória necessária

Atividade prática

1. **Exercícios básicos** (em aula): usos simples de malloc e free
2. **APS**: revisão de strings + usos mais realistas de malloc