

Sistemas Hardware-Software

Aula 12 – Tipos abstratos de dados

2021 – Engenharia

Maciel Vidal
Igor Montagner
Fábio Ayres

malloc

```
#include <stdlib.h>  
void *malloc(size_t size)
```

Se bem sucedido: retorna ponteiro para bloco de memória com pelo menos **size** bytes reservados, e com alinhamento de 8 bytes em x86, ou 16 bytes em x86-64. Se **size** for zero, retorna **NULL**.

Se falhou: retorna **NULL** e preenche **errno**

free

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

```
void free(void *p)
```

Devolve o bloco apontado por **p** para o *pool* de memória disponível

Alocação dinâmica

- Vantagens
 - Controle feito em **tempo de execução**
 - **Economia** de memória
 - **Expandir / diminuir / liberar** conforme necessário
- Desvantagens
 - Riscos da gerência
 - **Liberar** espaços não mais necessários
 - Não **acessar espaços já liberados**
 - Acessar **apenas a quantidade requisitada**
 - Etc.

Tipos Abstratos de Dados

Quais tipos de dados estão disponíveis em C?

Tipos Abstratos de Dados



A troca entre postes funciona como uma pilha

Tipos Abstratos de Dados

Operações que podem ser feitas com uma pilha:

```
typedef struct {  
    int capacity;  
    int *data;  
    int size;  
} stack_int;  
  
stack_int *stack_int_new(int capacity);  
void stack_int_delete(stack_int **s);  
int stack_int_empty(stack_int *s);  
int stack_int_full(stack_int *s);  
void stack_int_push(stack_int *s, int value);  
int stack_int_pop(stack_int *s);
```

Tipos Abstratos de Dados

- Conjunto de dados e operações
 - arquivo *.h*
- Criação de algoritmos com essas operações
 - Não depende de detalhes internos

Tipos Abstratos de Dados

- Vantagens:
 - Código mais expressivo
 - Diminui erros por repetição
 - Evita deixar struct em estado inconsistente

Tipos Abstratos de Dados

- Desvantagens:
 - Esconde todos os detalhes
 - Não permite usos mais avançados ou diferentes do original



Atividade prática

Implementação de Point2D (30 minutos)

1. Revisão de malloc
2. Compilação de programas com mais de um arquivo .c

Vetor dinâmico

O tipo de dados **vetor dinâmico** é implementado em diversas linguagens de alto nível.

- Python: `list`
- Java: `ArrayList`
- C++: `std::vector`

Vetor dinâmico

Suas principais operações são

- criação/destruição
- **at(i)** – devolve elemento na posição i
- **remove(i)** – remove o elemento na posição i , deslocando todos os outros para a esquerda
- **insert(i)** – insere um elemento na posição i , deslocando todos os elementos para a direita

Vetor dinâmico

As operações abaixo mudam o tamanho do vetor!

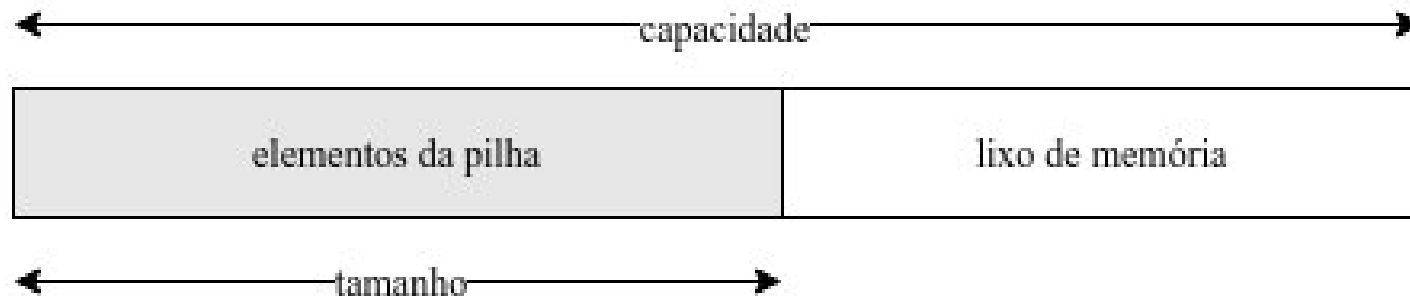
• **remove(i)** – remove o elemento na posição i , deslocando todos os outros para a esquerda

• **insert(i)** – insere um elemento na posição i , deslocando todos os elementos para a direita

Não é preciso declarar tamanho para o vetor dinâmico

Vetor dinâmico - capacidade

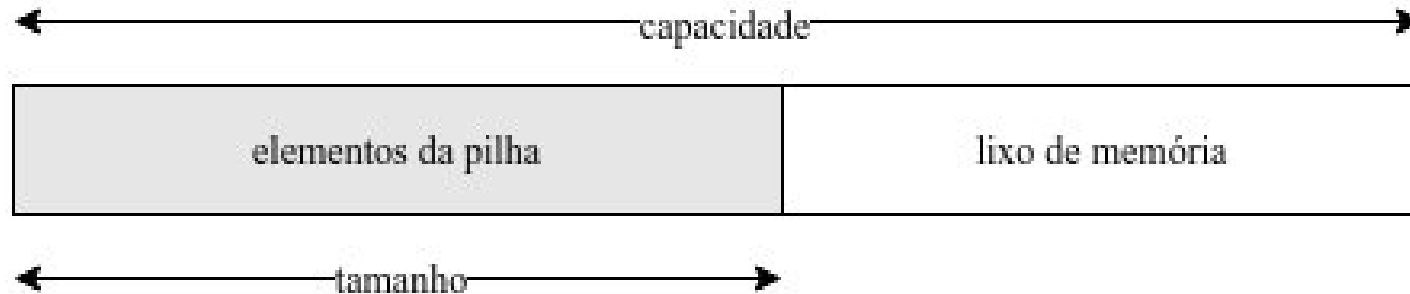
Relembrando *Desafios*



Supondo que soubéssemos o tamanho máximo que o vetor dinâmico assumiria, podemos aplicar esta técnica

Vetor dinâmico - capacidade

E se `tamanho == capacidade`?



Bom, nesse caso precisamos de um espaço de memória maior para nosso vetor!

realloc

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

```
void *realloc(void *ptr, size_t new_size)
```

Se bem sucedido: aloca um novo bloco de tamanho `new_size`, copia o conteúdo apontado por `ptr` para o novo bloco e retorna seu endereço. Antes de retornar chama `free(ptr)`.

Se falhou: retorna **NULL** e preenche **errno**

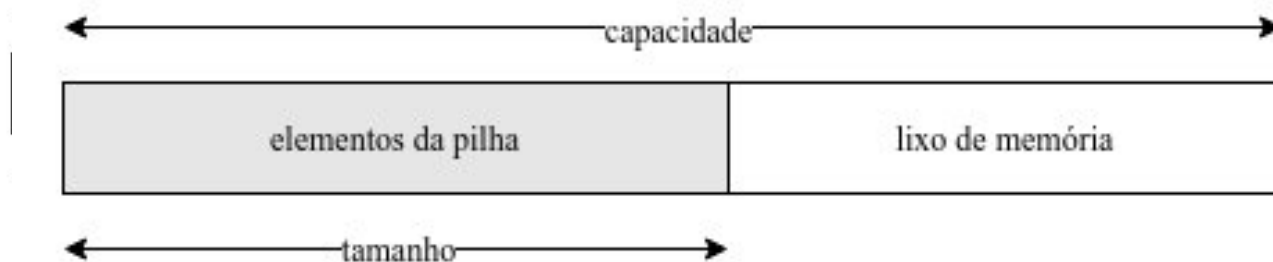
Vetor dinâmico – redimensionamento

- **Quando encher:** dobrar capacidade
- Quando ficar com **menos de um quarto** da capacidade: **diminuir** a capacidade pela **metade**

Vetor dinâmico - capacidade

E se tamanho == capacidade?

- 1) Criamos um novo espaço de memória e copiamos o conteúdo para lá com realloc
- 2) Atualizamos a nova capacidade
- 3) Atualizamos o ponteiro para os novos dados



Atividade prática

Implementação de Vetor dinâmico (Entrega)

1. Revisão de malloc
2. Compilação de programas com mais de um arquivo .c
3. Entender uso de um TAD a partir de exemplos de uso

Insper

www.insper.edu.br