

Sistemas Hardware-Software

Aula 12 - Tipos abstratos de dados

Engenharia

Fabio Lubacheski

Maciel C. Vidal

Igor Montagner

Fábio Ayres

Tipos-de-dados

Um **tipo-de-dados** (*=data type*) é um conjunto de **valores** munido de um conjunto de **operações**.

Quais tipos de dados estão disponíveis em C?

Tipos-de-dados

Um **tipo-de-dados** (*=data type*) é um conjunto de **valores** munido de um conjunto de **operações**.

Quais tipos de dados estão disponíveis em C?

Exemplo o tipo **int** representam **valores inteiros** e é possível realizar **operações** de comparação (<, <=, ==, >, >=) e as aritméticas (+, -, *, /).

Tipos Abstratos de Dados

Um tipo abstrato de dados (= *abstract data type*) é um **tipo-de-dados** definido em termos do seu **comportamento** e não em termos de sua **representação**.

As operações são especificadas em uma **interface** que **diz o que cada operação faz, sem dizer como faz o que faz**.

Exemplo o TAD – Pilha (=Stack)

No TAD Pilha temos as seguintes **operações**: **inserção** de um item (**Push**) e **remoção** de um item mais recente (**Pop**).

Os **itens** podem ter tipos-de-dados (int,char) e outros TADs

Comportamento: o primeiro item a entrar é o primeiro a sair.

Exemplos de uso de pilha:

- Eliminação de recursividade em funções;
- Navegação entre páginas Web (botão voltar); e
- cálculo do valor de uma expressão aritmética.

TAD – Pilha (=Stack) - Interface

Os programas que usam o TAD só tem acesso a **interface**. Na linguagem C a interface é representada por um arquivo-interface (=header file).

```
1  // stack.h
2  typedef struct {
3      int capacity;
4      int *data;
5      int size;
6  }stack_int;
7
8  stack_int * stack_int_new(int capacity);
9  void stack_int_delete(stack_int *_s);
10 int stack_int_empty(stack_int *s);
11 int stack_int_full(stack_int *s);
12 void stack_int_push(stack_int *s, int value);
13 int stack_int_pop(stack_int *s);
```

TAD – Pilha (=Stack) – Implementação

A implementamos das funções do arquivo .h ficam em um arquivo .c, por convenção de mesmo nome.

```
1 //stack.c
2 #include <stdlib.h>
3 #include <stdio.h>
4 #include "stack.h"
5 stack_int *stack_int_new(int capacity) {
6     stack_int *s = (stack_int *)malloc(sizeof(stack_int));
7     if (s == NULL) {
8         printf("Erro de alocação de memoria para estrutura!\n");
9         exit(EXIT_FAILURE);
10    }
11    s->capacity = capacity;
12    s->size = 0;
13    s->data = (int *)malloc(capacity * sizeof(int));
14    if (s->data == NULL) {
15        free(s);
16        printf("Erro de alocação de memoria para os dados!\n");
17        exit(EXIT_FAILURE);
18    }
19    return s;
20 }
```


TAD – Pilha (=Stack) – Testando

Para testar a implementação podemos criar um programa com a função main() que utiliza o TAD.

```
1 // test_stack.c
2 // gcc -Og -Wall -g test_stack.c stack.o -o teste_stack
3 #include <stdio.h>
4 #include "stack.h"
5
6 int main() {
7     // Cria um stack com capacidade para 5 elementos
8     stack_int *s = stack_int_new(5);
9
10    stack_int_push(s, 10);
11    stack_int_push(s, 20);
12    stack_int_push(s, 30);
13    // Remove o topo: 30
14    printf("Valor removido: %d\n", stack_int_pop(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!

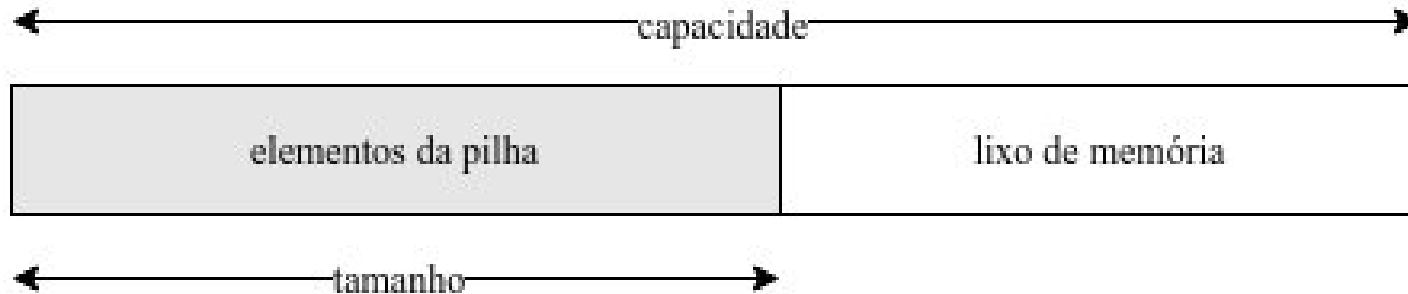
- **remove(i)** – remove o elemento na posição i , deslocando todos os outros para a esquerda
- **insert(i)** – insere um elemento na posição i , deslocando todos os elementos para a direita

Não é preciso declarar tamanho para o vetor dinâmico

Vetor dinâmico

Capacidade

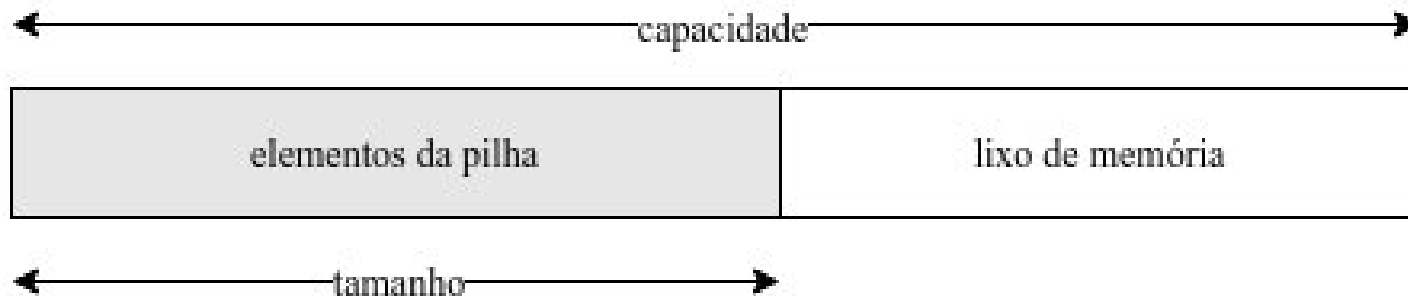
Relembrando *Desafios*



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

Vetor dinâmico

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**

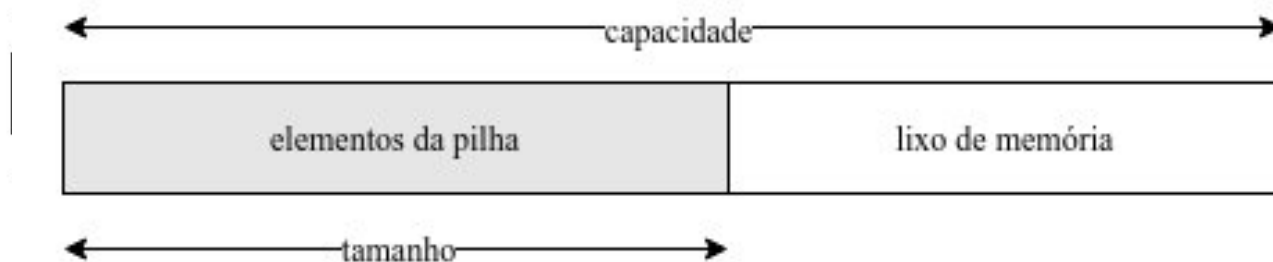
Vetor dinâmico

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

Vetor dinâmico

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 para entrega

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