UMA MINI-ZINE SOBRE

PILHA

(STACK)

@andrekishimoto

- 2023 -

PILHA (STACK)

A **pilha** (***stack***) é uma estrutura de dados que define como os dados são acessados, aplicando uma regra que segue o princípio **LIFO**: *Last-In, First-Out* (o último a entrar é o primeiro a sair). Há pessoas que falam **FILO**: *First-In, Last-Out* (o primeiro a entrar é o último a sair). Também serve.

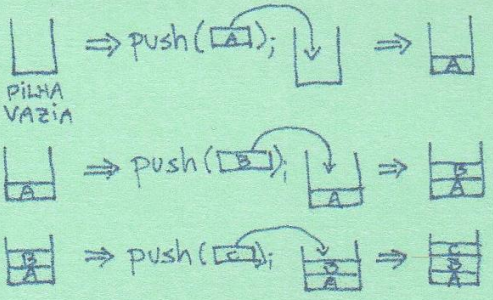
Dado que o nome "pilha" vem do sentido de empilhar (colocar uma coisa em cima de outra), o acesso aos elementos/dados armazenados em uma pilha é feito a partir do topo da pilha, que contém o elemento mais recente.

↳ o último elemento inserido na pilha.

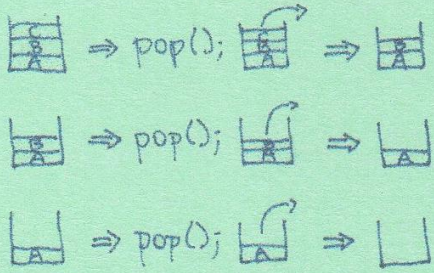
Seguindo o princípio LIFO, precisamos ter em mente que 1) um elemento inserido na pilha sempre é colocado no topo e 2) só é possível acessar/remover o elemento que está no topo da pilha. Ou seja, para acessar ou retirar o penúltimo elemento inserido na pilha, antes precisamos remover o último elemento inserido na pilha.

Considerando a possibilidade de inserir, remover e consultar um elemento na pilha, podemos assumir que existem três operações fundamentais para uma pilha: `push()`, `pop()` e `top()`.

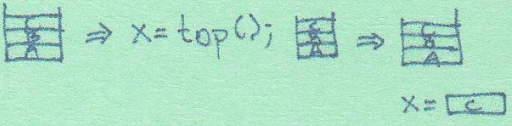
» **push( )**: insere um elemento no topo da pilha.



» **pop( )**: remove e retorna o elemento do topo da pilha.



» **top( )**: retorna o elemento do topo da pilha, sem removê-lo.



A operação **top( )** também é conhecida como **peek( )** ("dar uma olhadinha" no topo da pilha), e alguns materiais não a consideram fundamental, pois podemos descobrir quem está no topo da pilha com a remoção via **pop( )** e depois colocamos o elemento de volta ao topo com **push( )**. → não é lá muito eficiente, né?

Outras operações comuns em uma pilha, que facilitam seu uso:
» **create( )**: cria e retorna uma pilha vazia.
» **size( )**: retorna a capacidade da pilha.
» **count( )**: retorna a quantidade de elementos na pilha.
» **isEmpty( )**: retorna se a pilha está vazia ou não.
» **isFull( )**: retorna se a pilha está cheia ou não.
» **clear( )**: esvazia a pilha (remove todos os elementos da pilha).

PSEUDOCÓDIGO*

** Observação: há diversas maneiras de implementar uma pilha, sendo que o conteúdo desta zine apresenta apenas uma possibilidade de implementação.*

Podemos criar pilhas estáticas (sequenciais) e dinâmicas (encadeadas). A seguir, veremos pseudocódigos apenas da versão sequencial, cuja implementação usa um *array* para armazenar os dados inseridos em uma pilha.

O básico que precisamos para representar uma pilha sequencial em código é:

» Um **contador** de quantos elementos estão na pilha (`int count`). Esse contador começa em zero e também é usado para saber o índice correto de um novo elemento que será inserido no topo da pilha, assumindo que o índice do primeiro elemento de um *array* é zero.

» Um ***array*** do tipo de dado que queremos armazenar na pilha - vamos chamar de `<type> data[]` (sendo que `<type>` deve ser substituído pelo tipo da pilha - por ex., `int`, `float`, `String`, etc.).

O *array* usado para armazenar os dados na pilha estática tem um tamanho fixo (a capacidade da pilha) que pode ser definido em tempo de compilação (alocação estática) ou em tempo de execução (alocação dinâmica).

Quando o tamanho é definido em tempo de compilação, é recomendável declarar uma constante inteira ao invés de usar "números mágicos" no código. Por exemplo:

Em C, poderíamos declarar uma
```
const int STACK_CAPACITY = 128;
```

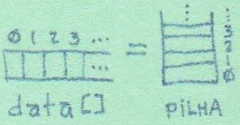
E o equivalente em Java seria
```
final int STACK_CAPACITY = 128;
```

Com essas duas variáveis (ex. `int count` e `<type> data[STACK_CAPACITY]`) no código, podemos implementar as operações fundamentais:

PUSH

```
void push(<type> value) { // O(1)
  // TODO: O que fazer quando a pilha
  // estiver cheia?

  data[count] = value;
  ++count;
}
```



POP

```
<type> pop() { // O(1)
  // TODO: O que fazer quando a pilha
  // estiver vazia?

  --count;
  <type> top = data[count];
  // No lugar de null, use o equivalente
  // para o tipo de dado da pilha. A ideia
  // é reiniciar o valor do elemento
  // removido.
  data[count] = null;
  return top;
}
```

TOP

```
<type> top() { // O(1)
  // TODO: O que fazer quando a pilha
  // estiver vazia?

  return data[count - 1];
}
```

AGORA É A SUA VEZ!

1. Implemente as operações fundamentais da **pilha** na linguagem de programação que você está estudando e escreva um exemplo de código que insere, remove e consulta os elementos da pilha.

2. Implemente as operações extras `create()`, `size()`, `count()`, `isEmpty()`, `isFull()` e `clear()`.
A operação `create()` pode ser o construtor da classe da pilha, caso esteja usando uma linguagem orientada a objetos e baseada em classes.

3. Nos pseudocódigos, há comentários `TODO` ("a fazer"), indicando que devemos verificar os casos em que a pilha está cheia ou vazia. Remova esses comentários e implemente as verificações para que a pilha funcione corretamente.

4. Quais são os prós e contras da pilha sequencial?

5. Pesquise e descreva algumas aplicações computacionais onde a estrutura pilha é usada.

6. Explique o que é a estrutura de dados pilha, mas com as suas próprias palavras. É um bom exercício para verificar se você entendeu o conceito de pilha. Aproveite e consulte mais referências para se aprofundar no assunto!

REFERÊNCIAS E MAIS...

Essa zine foi criada com base nas minhas notas de aulas sobre programação e estrutura de dados que elaborei há cerca de uma década [e que continuo atualizando de tempos em tempos] e em implementações que escrevi enquanto desenvolvedor de software.

Falhei miseravelmente em manter um histórico de todas as referências que consultei no decorrer dos anos (algumas referências foram pessoas e código-fonte de projetos que trabalhei, além dos professores que tive durante meus estudos), mas há alguns livros que foram muito importantes para o meu aprendizado e sempre indico como referências. Omiti ano de publicação e edição, pois recomendo as edições mais recentes.

» CELES, W., CERQUEIRA, R. RANGEL, J. L. **Introdução a Estruturas de Dados - com técnicas de programação em C**. Editora Campus.
» CORMEN, T. et al., **Introduction to Algorithms**. The MIT Press.
» PENTON, R. **Data Structures for Game Programmers**. Premier Press.
» SEDGEWICK, R. **Algorithms in C**. Addison-Wesley.

Além dos livros acima, implementar (praticar!) em diversas linguagens de programação foi o que me fez compreender e fixar os conceitos de estruturas de dados, algo que recomendo a você fazer também!