

Pilhas

Estrutura de Dados — QXD0010



UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará

2º semestre/2023



Pilha

- São **listas lineares** que adotam a política LIFO para a manipulação de elementos.

Pilha

- São **listas lineares** que adotam a política LIFO para a manipulação de elementos.
- **LIFO** (*last-in first-out*): último a entrar é primeiro a sair. Remove primeiro objetos **inseridos há menos tempo**

Pilha

- São **listas lineares** que adotam a política LIFO para a manipulação de elementos.
- **LIFO** (*last-in first-out*): último a entrar é primeiro a sair. Remove primeiro objetos **inseridos há menos tempo**



É como uma pilha de pratos:

Pilha

- São **listas lineares** que adotam a política LIFO para a manipulação de elementos.
- **LIFO** (*last-in first-out*): último a entrar é primeiro a sair. Remove primeiro objetos **inseridos há menos tempo**



É como uma pilha de pratos:

- **Empilha** os pratos limpos sobre os que já estão na pilha

Pilha

- São **listas lineares** que adotam a política LIFO para a manipulação de elementos.
- **LIFO** (*last-in first-out*): último a entrar é primeiro a sair. Remove primeiro objetos **inseridos há menos tempo**



É como uma pilha de pratos:

- **Empilha** os pratos limpos sobre os que já estão na pilha
- **Desempilha** o prato de cima para usar

Pilha

Operações básicas:

Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha

Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo:



Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **push**(A)

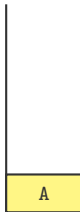


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **push**(A)



Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **push**(B)

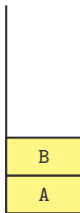


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **push**(B)

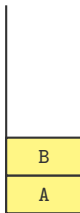


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **pop()**

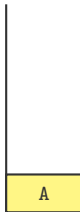


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **pop()**

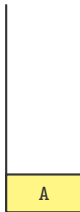


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **push**(C)

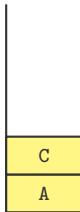


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **push**(C)

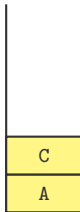


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **push**(D)

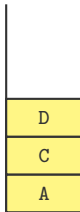


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **push**(D)

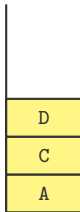


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **pop()**

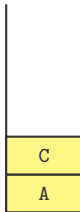


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **pop()**

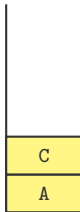


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **pop()**

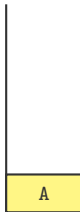


Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **pop()**



Pilha

Operações básicas:

- **push** (*empilhar*): adiciona no topo da pilha
- **pop** (*desempilhar*): remove do topo da pilha

Exemplo: **pop()**



Implementação de uma Pilha



Pilha usando Lista Encadeada

- Em algumas aplicações computacionais precisamos usar a estrutura de dados pilha, e não sabemos de antemão o tamanho da pilha.
- Nesses casos, a implementação da pilha pode ser feita de forma simples usando **alocação encadeada**.

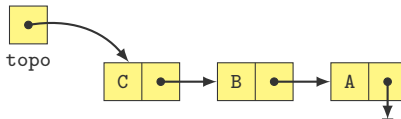
Pilha usando Lista Encadeada

- Em algumas aplicações computacionais precisamos usar a estrutura de dados pilha, e não sabemos de antemão o tamanho da pilha.
- Nesses casos, a implementação da pilha pode ser feita de forma simples usando **alocação encadeada**.
- Vamos implementar as operações:
 - criar uma pilha vazia
 - verificar se a pilha está vazia
 - retornar o tamanho atual da pilha
 - consultar elemento no topo
 - inserir elemento no topo
 - remover elemento do topo
 - liberar a memória da pilha
- Implementaremos a pilha como uma classe chamada **Stack**.

Implementação

Implementaremos a estrutura de dados Pilha como uma classe chamada **Stack**, usando como base uma **lista simplesmente encadeada sem nó sentinela**.

Como exemplo ilustrativo, após empilhar **A**, **B** e **C**, a lista encadeada deve ter a seguinte configuração:



Arquivo Node.h

Cada nó da lista é um **struct** definido do seguinte modo:

```
1  #ifndef  NODE_H
2  #define  NODE_H
3
4  template<typename T>
5  struct Node {
6      T value;      // valor a ser empilhado
7      Node *next;  // ponteiro para o proximo elemento
8
9      // Construtor
10     Node(const T& val, Node *nxt) {
11         value = val;
12         next = nxt;
13     }
14
15     // Destrutor
16     ~Node() {
17         delete next;
18     }
19 };
20
21 #endif
```

Arquivo Stack.h

```
1  #ifndef STACK_H
2  #define STACK_H
3  #include <stdexcept>
4  #include "Node.h"
5
6  template<typename T>
7  class Stack {
8  private:
9      Node<T> *m_top {nullptr}; // Ponteiro para o topo da pilha
10     int m_size{0};
11
12 public:
13     Stack() = default; // Construtor default
14     ~Stack(); // Destrutor
15     int size() const; // Devolve tamanho da pilha
16     bool empty() const; // Pilha esta vazia?
17     void push(const T& val); // Inserir no topo
18     void pop(); // Remover elemento do topo
19     T& top(); // Consulta o elemento no topo
20     const T& top() const; // Consulta o elemento no topo
21 };
```

Arquivo Stack.h (cont.)

```
22  template<typename T>
23  Stack<T>::~~Stack() {
24      delete m_top;
25  }
26
27  template<typename T>
28  bool Stack<T>::empty() const {
29      return m_size == 0;
30  }
31
32  template<typename T>
33  int Stack<T>::size() const {
34      return m_size;
35  }
36
37  template<typename T>
38  void Stack<T>::push(const T& val) {
39      m_top = new Node(val, m_top);
40      m_size++;
41  }
```


Arquivo Stack.h (cont.)

```
42 template<typename T>
43 void Stack<T>::pop() {
44     if(m_size > 0) {
45         Node<T> *temp = m_top;
46         m_top = temp->next;
47         temp->next = nullptr;
48         delete temp;
49         m_size--;
50     }
51 }
```

Arquivo Stack.h (cont.)

```
52 template<typename T>
53 T& Stack<T>::top() {
54     if(m_size == 0) {
55         throw std::runtime_error("empty stack");
56     }
57     return m_top->value;
58 }
59
60 template<typename T>
61 const T& Stack<T>::top() const {
62     if(m_size == 0) {
63         throw std::runtime_error("empty stack");
64     }
65     return m_top->value;
66 }
67
68 #endif
```

Arquivo main.cpp

```
1 #include <iostream>
2 #include "Stack.h"
3 using namespace std;
4
5 int main() {
6     Stack<int> pilha;
7
8     for(int i = 1; i <= 9; i++)
9         pilha.push(i);
10
11     while (!pilha.empty()) {
12         cout << pilha.top() << " ";
13         pilha.pop();
14     }
15     cout << endl;
16 }
```

Exemplos de aplicações

Algumas aplicações de pilhas:

- Balanceamento de parênteses
 - expressões matemáticas
 - linguagens de programação
 - HTML...
- Cálculo e conversão de notações
 - pré-fixa
 - pós-fixa
 - infixa (com parênteses)
- Percurso de estruturas de dados complexas (árvores, grafos, etc.)
- Recursão

Exercícios



Exercícios

- (1) **Inversão de palavras.** Escreva uma função em C++ que inverta a ordem das letras de cada palavra de uma sentença, preservando a ordem das palavras. Suponha que as palavras da sentença são separadas por espaços. Por exemplo, a aplicação da sua operação à sentença AMU MEGASNEM ATERCES deve produzir a sentença UMA MENSAGEM SECRETA.
- (2) Escreva uma função que verifique se uma string de entrada é da forma $str_1 C str_2$ tal que str_1 é uma string composta apenas por caracteres A e B e str_2 é a string reversa de str_1 . Por exemplo, a cadeia $ABABBACABBABA$ é do formato especificado, enquanto as cadeias $ABABBACABB$, ABA , $BBBBCAA$ e $ABBACBAABBBBAB$ não seguem o formato. Sua função deve obedecer o seguinte protótipo:

```
bool str1Cstr2(string str);
```

Exercícios

- (3) Implemente uma pilha **usando exatamente DUAS filas**. A pilha deve armazenar números inteiros. Não é necessário implementar todas as operações do TAD Pilha, as QUATRO únicas operações que são obrigatórias para esta questão são as operações empilhar (push), desempilhar (pop), construir pilha (Construtor) e destruir Pilha (Destrutor).

Observação: Crie um arquivo **main.cpp** a fim de testar a estrutura de dados pilha que você criou.

Atenção: Não é para contruir a pilha suando listas encadeadas ou vetores. Pois nós já fizemos isso em aula. A única estrutura de dados permitida e que pode ser usada para resolver esse exercícios são as filas. Use EXATAMENTE duas filas. Nem mais nem menos que isso.

FIM

