Pilhas

Estrutura de Dados — QXD0010

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

Universidade Federal do Ceará
Campus Quixadá

2º semestre/2021

Introdução

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:

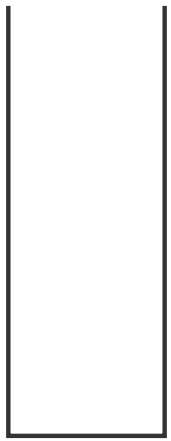
- push (*empilhar*): adiciona no topo da pilha
- pop (*desempilhar*): remove do topo da pilha

Pilha

Operações básicas:

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

Exemplo:

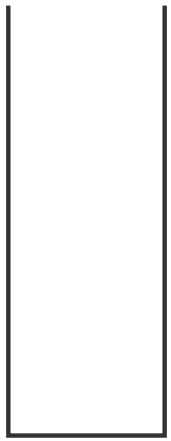


Pilha

Operações básicas:

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

Exemplo: `push(A)`

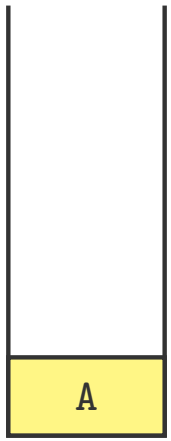


Pilha

Operações básicas:

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

Exemplo: `push(A)`



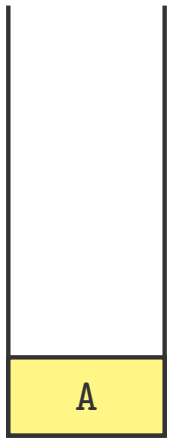
Pilha

Universidade Federal do Ceará
Campus Quixadá

Operações básicas:

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

Exemplo: `push(B)`

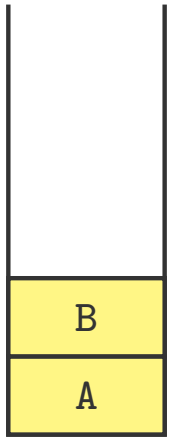


Pilha

Operações básicas:

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

Exemplo: `push(B)`



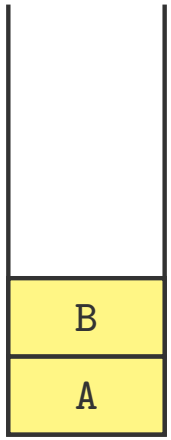
Pilha

Universidade Federal do Ceará
Campus Quixadá

Operações básicas:

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

Exemplo: `pop()`

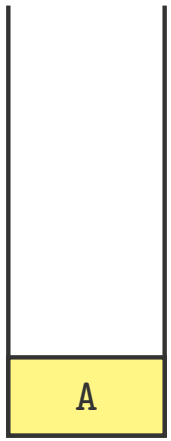


Pilha

Operações básicas:

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

Exemplo: `pop()`

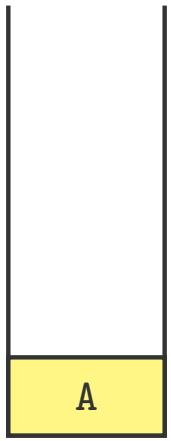


Pilha

Operações básicas:

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

Exemplo: `push(C)`



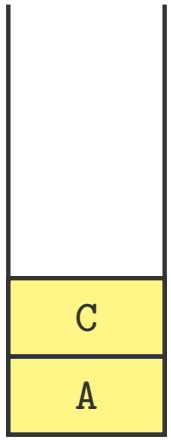
Pilha

Universidade Federal do Ceará
Campus Quixadá

Operações básicas:

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

Exemplo: `push(C)`

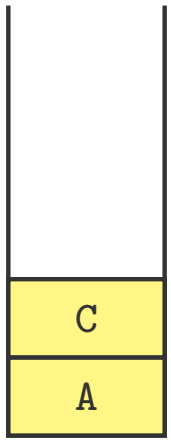


Pilha

Operações básicas:

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

Exemplo: `push(D)`

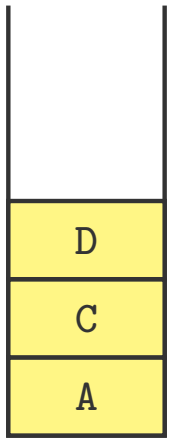


Pilha

Operações básicas:

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

Exemplo: `push(D)`

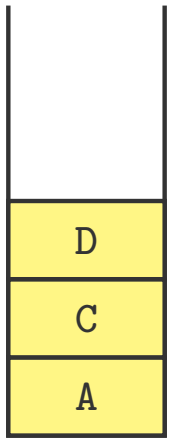


Pilha

Operações básicas:

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

Exemplo: `pop()`

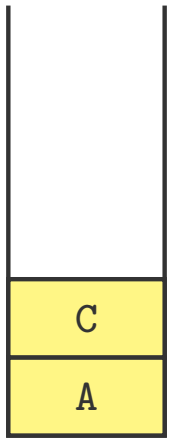


Pilha

Operações básicas:

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

Exemplo: `pop()`

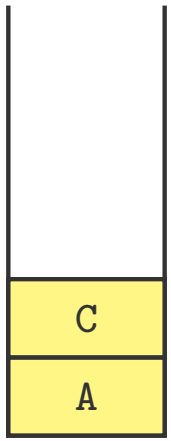


Pilha

Operações básicas:

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

Exemplo: `pop()`

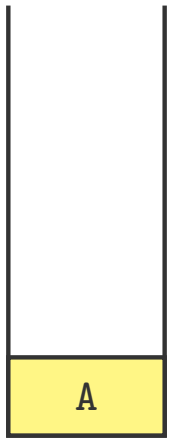


Pilha

Operações básicas:

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

Exemplo: `pop()`

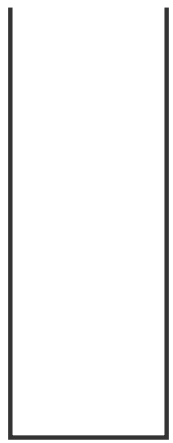


Pilha

Operações básicas:

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

Exemplo: `pop()`



Implementação de uma Pilha

Implementação de Pilha usando Listas

- 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 um alocação encadeada.

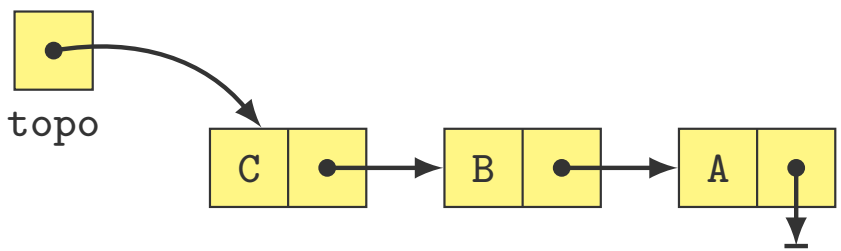
- Vamos implementar as operações:
  - criar uma pilha vazia
  - inserir elemento no topo
  - remover elemento do topo
  - verificar se a pilha está vazia
  - retornar o tamanho atual da pilha
  - liberar a memória da pilha

- 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  typedef int Item;
5
6  struct Node {
7      Item data;
8      Node *next;
9
10     Node(const Item& data, Node *ptr) {
11         this->data = data;
12         this->next = ptr;
13     }
14 };
15
16 #endif
```

Arquivo Stack.h


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

Arquivo Stack.cpp


```
1  #include <iostream>
2  #include <stdexcept>
3  #include "Node.h"
4  #include "Stack.h"
5
6  // Construtor
7  Stack::Stack() {
8      m_top = nullptr; // Pilha vazia
9  }
10
11 // Destrutor
12 Stack::~Stack() {
13     while(m_top != nullptr) {
14         Node *temp = m_top;
15         m_top = m_top->next;
16         delete temp;
17     }
18 }
```

Arquivo Stack.cpp


```
19 // Pilha esta vazia?
20 bool Stack::empty() const {
21     return m_top == nullptr;
22 }
23
24 int Stack::size() const {
25     Node *temp = m_top;
26     int contador = 0;
27     while(temp != nullptr) {
28         contador++;
29         temp = temp->next;
30     }
31     return contador;
32 }
33
34 // Insere elemento no topo
35 void Stack::push(const Item& data) {
36     Node *novo = new Node(data, m_top);
37     m_top = novo;
38 }
```

Arquivo Stack.cpp

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
39 // Remove elemento do topo
40 void Stack::pop() {
41     if(empty()) {
42         throw std::runtime_error("erro: pilha vazia");
43     }
44     Node *temp = m_top;
45     m_top = m_top->next;
46     delete temp;
47 }
48
49 // Retorna uma referência para o
50 // valor do elemento no topo
51 Item& Stack::top() {
52     if(empty) {
53         throw std::runtime_error("erro: pilha vazia");
54     }
55     return m_top->data;
56 }
```

Implementação usando Vetor

Implementação de Pilha usando um Vetor:

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- Em algumas aplicações computacionais que precisam de uma estrutura de dados pilha, é comum saber de antemão o tamanho da pilha.

- Nesses casos, a implementação da pilha pode ser feita de forma simples usando um vetor.

- Vamos implementar as operações:
  - criar uma pilha vazia
  - inserir elemento no topo
  - remover elemento do topo
  - verificar se a pilha está vazia
  - verificar se a pilha está cheia
  - liberar a estrutura de pilha

- Implementaremos a pilha como uma Classe chamada Stack.

Arquivo Stack.h


```
1  #ifndef STACK_H
2  #define STACK_H
3
4  typedef int Item;
5
6  class Stack {
7  private:
8      Item *vec; // Ponteiro para um vetor de Item
9      int m_top; // Posicao do proximo slot disponivel
10     int capacity; // Tamanho total do vetor
11 public:
12     Stack(int capacity); // Construtor
13     ~Stack(); // Destrutor
14     void push(const Item& data); // Inserir no topo
15     void pop(); // Remover elemento do topo
16     Item& top(); // Consulta elemento no topo
17     bool empty() const; // Pilha esta vazia?
18     bool full() const; // Pilha esta cheia?
19     int size() const; // Tamanho atual da pilha
20 };
21
22 #endif
```

Arquivo Stack.cpp


```
23 // Pilha esta vazia?
24 bool Stack::empty() const {
25     return m_top == 0;
26 }
27
28 // Pilha esta cheia?
29 bool Stack::full() const {
30     return m_top == capacity;
31 }
32
33 int Stack::size() const {
34     return m_top;
35 }
36
37 // Retorna referência para elemento no topo
38 Item& Stack::top() {
39     return vec[ m_top-1 ];
40 }
```

Arquivo Stack.cpp


```
41 // Insere elemento no topo
42 void Stack::push(const Item& data) {
43     if(full()) {
44         throw std::runtime_error("erro: pilha cheia");
45     }
46     vec[m_top] = data;
47     m_top++;
48 }
49
50 // Remove elemento do topo
51 void Stack::pop() {
52     if(empty()) {
53         throw std::runtime_error("erro: pilha vazia");
54     }
55     m_top--;
56 }
```

Arquivo main.cpp


```
1  #include <iostream>
2  #include "Stack.h"
3  using namespace std;
4
5  int main() {
6      Stack pilha(20);
7
8      int i = 1;
9      while (!pilha.full()) {
10         pilha.push(i++);
11     }
12
13     while (!pilha.empty()) {
14         cout << pilha.top() << " ";
15         pilha.pop();
16     }
17     cout << endl;
18
19 }
```

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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM