



INE5408-03208A|INE5609-03238B (20182) - Estruturas de Dados

Painel ► Agrupamentos de Turmas ► INE5408-03208A|INE5609-03238B (20182) ► Tópico 10 ► Implementação de Pilha Encadeada

NAVEGAÇÃO



Painel

- Página inicial do site
- Moodle UFSC
- ▼ Curso atual
 - ▼ INE5408-03208A|INE5609-03238B (20182)
 - Participantes
 - Emblemas
 - Geral
 - Tópico 1
 - Tópico 2
 - Tópico 3
 - Tópico 4
 - Tópico 5
 - Tópico 6
 - Tópico 7
 - Tópico 8
 - Tópico 9
 - ▼ Tópico 10
 - ▼ Implementação de Pilha Encadeada
 - Descrição
 - Enviar
 - Editar
 - Visualizar envios
 - 🔍 Testes (Pilha Encadeada)
 - 🖨 Implementação de Fila Encadeada
 - 🔍 Testes (Fila Encadeada)
 - 🖨 Implementação de Lista
 - 🖨 Duplamente Encadeada
 - 🔍 Testes (Lista Duplamente Encadeada)
 - 📺 Videoaula de Pilha e Fila Encadeada
 - 📺 Lecture 8 (rev. 04-set)
- Meus cursos

ADMINISTRAÇÃO



- Administração do curso

Descrição Enviar Editar Visualizar envios

Nota

Revisado em domingo, 9 Set 2018, 16:04 por Atribuição automática de nota
Nota 100 / 100

Relatório de avaliação

[+]Summary of tests

Enviado em domingo, 9 Set 2018, 16:04 (Baixar)

linked_stack.h

```
1 1 //! Copyright 2018 Matheus Henrique Schaly
2
3 #ifndef STRUCTURES_LINKED_STACK_H
4 #define STRUCTURES_LINKED_STACK_H
5
6 #include <cstdint>
7 #include <stdexcept>
8
9
10 namespace structures {
11
12     //! Dynamic stack implementation
13     template<typename T>
14     class LinkedStack {
15     public:
16         //! Constructor
17         LinkedStack();
18
19         //! Destructor
20         ~LinkedStack();
21
22         //! Clears the stack
23         void clear();
24
25         //! Inserts an element at the rightmost of the stack
26         void push(const T& data);
27
28         //! Removes an element from the rightmost of the stack
29         T pop();
30
31         //! Returns the data at the rightmost node of the stack
32         T& top() const;
33
34         //! Returns true if stack is empty and false otherwise
35         bool empty() const;
36
37         //! Returns the current size of the queue
38         std::size_t size() const;
39
40     private:
41         class Node {
42         public:
43             //! Constructor with 1 parameter
44             explicit Node(const T& data):
45                 data_(data)
46             {}
47
48             //! Constructor with 2 parameters
49             Node(const T& data, Node* next):
50                 data_(data),
51                 next_(next)
52             {}
53
54             //! Info's getter
55             T& data() {
56                 return data_;
57             }
58
59             //! Info's constant getter
60             const T& data() const {
61                 return data_;
62             }
63
64             //! Next's getter
65             Node* next() {
66                 return next_;
67             }
68
69             //! Next's constant getter
70             const Node* next() const {
71                 return next_;
72             }
73
74             //! Next's setter
75             void next(Node* next) {
76                 next_ = next;
77             }
78
79     private:
80         //! Node's data
81         T data_;
82
83         //! Node's next node
84         Node* next_;
85     };
86
87     //! Stack's rightmost node
88     Node* top_{nullptr};
89
90     //! Stack's current size
91     std::size_t size_{0u};
92 };
93
94 // namespace structures
95
96 template<typename T>
97 structures::LinkedStack<T>::LinkedStack() {}
98
99 template<typename T>
100 structures::LinkedStack<T>::~LinkedStack() {
101     clear();
102 }
103
104 template<typename T>
105 void structures::LinkedStack<T>::clear() {
106     while (!empty()) {
107         pop();
108     }
109 }
110
111 template<typename T>
112 void structures::LinkedStack<T>::push(const T& data) {
113     Node* node = new Node(data, nullptr);
114     if (node == nullptr) {
115         throw std::out_of_range("A pilha esta cheia.");
116     }
117     ...
118 }
```

```

117     if (empty()) {
118         top_ = node;
119     } else {
120         node -> next(top_);
121         top_ = node;
122     }
123     size_++;
124 }
125
126 template<typename T>
127 T structures::LinkedStack<T>::pop() {
128     if (empty()) {
129         throw std::out_of_range("A pilha esta vazia.");
130     }
131     Node* node = top_;
132     T deleted_data = node -> data();
133     top_ = top_ -> next();
134     delete node;
135     size_--;
136     return deleted_data;
137 }
138
139 template<typename T>
140 T& structures::LinkedStack<T>::top() const {
141     if (empty()) {
142         throw std::out_of_range("A pilha esta vazia.");
143     }
144     return top_ -> data();
145 }
146
147 template<typename T>
148 bool structures::LinkedStack<T>::empty() const {
149     return size_ == 0;
150 }
151
152 template<typename T>
153 std::size_t structures::LinkedStack<T>::size() const {
154     return size_;
155 }
156
157 #endif
158

```