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

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
      - Testes (Pilha Encadeada)
      - **Implementação de Fila Encadeada**
        - Descrição
        - Enviar
        - Editar
        - Visualizar envios
      - 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, 16:32 por Atribuição automática de nota
Nota 100 / 100
Relatório de avaliação
[+]**Summary of tests**
Enviado em domingo, 9 Set 2018, 16:32 (Baixar)

linked_queue.h

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

```cpp
template<typename T>
void structures::LinkedQueue<T>::enqueue(const T& data) {
    Node* new_node = new Node(data, tail);
    if (new_node == nullptr) {
        throw std::out_of_range("A fila esta cheia.");
    }
    if (size_ == 0) {
        head = new_node;
    }
    tail = new_node;
    size_++;
}

template<typename T>
T structures::LinkedQueue<T>::dequeue() {
    if (empty()) {
        throw std::out_of_range("A fila esta vazia.");
    }
    Node* temp = tail;
    if (size_ == 1) {
        T deleted_data = temp -> data();
        tail = nullptr;
        head = nullptr;
        delete temp;
        size_--;
        return deleted_data;
    }
    while (temp -> next() != head) {
        temp = temp -> next();
    }
    T deleted_data = head -> data();
    delete head;
    temp -> next(nullptr);
    head = temp;
    size_--;
    return deleted_data;
}

template<typename T>
T& structures::LinkedQueue<T>::front() const {
    if (empty()) {
        throw std::out_of_range("A fila esta vazia.");
    }
    return head -> data();
}

template<typename T>
T& structures::LinkedQueue<T>::back() const {
    if (empty()) {
        throw std::out_of_range("A fila esta vazia.");
    }
    return tail -> data();
}

template<typename T>
bool structures::LinkedQueue<T>::empty() const {
    return size_ == 0;
}

template<typename T>
std::size_t structures::LinkedQueue<T>::size() const {
    return size_;
}

#endif

```