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
      - Implementação de Lista em vetor
        - Descrição
        - Enviar
        - Editar
        - Visualizar envios
      - Testes (Lista)
    - Tópico 6
    - Tópico 7
    - Tópico 8
- Meus cursos

ADMINISTRAÇÃO

- Administração do curso

Nota

Revisado em quarta, 29 Ago 2018, 07:41 por Atribuição automática de nota
Nota 100 / 100
Relatório de avaliação
[-]Summary of tests

```
+------------------------------+
| 22 tests run/22 tests passed |
+------------------------------+
```

Enviado em quarta, 29 Ago 2018, 07:41 (Baixar)

array_list.h

```
1  //! Copyright 2018 Matheus Henrique Schaly
2
3  #ifndef STRUCTURES_ARRAY_LIST_H
4  #define STRUCTURES_ARRAY_LIST_H
5
6  #include <cstdint>
7  #include <stdexcept>
8
9
10 namespace structures {
11
12 //! Static List
13 template<typename T>
14 class ArrayList {
15  public:
16 //! Constructor
17 ArrayList();
18
19 //! Constructor with parameter
20 ArrayList(std::size_t max_size);
21
22 //! Destructor
23 ~ArrayList();
24
25 //! Clears the list
26 void clear();
27
28 //! Pushes an element to the back of the list
29 void push_back(const T& data);
30
31 //! Pushes an element to the front of the list
32 void push_front(const T& data);
33
34 //! Inserts an element at a specfic index
35 void insert(const T& data, std::size_t index);
36
37 //! Inserts an element in a sorted possition
38 void insert_sorted(const T& data);
39
40 //! Removes an element from a specific index
41 T pop(std::size_t index);
42
43 //! Removes an element from the back of the list
44 T pop_back();
45
46 //! Removes an element from the front of the list
47 T pop_front();
48
49 //! Removes the first element containing the data
50 void remove(const T& data);
51
52 //! Verifies if the list if full
53 bool full() const;
54
55 //! Verifies if the list is empty
56 bool empty() const;
57
58 //! Verifies if the list contains the data
59 bool contains(const T& data) const;
60
61 //! Returns the index of the first element containg the data, else return size
62 std::size_t find(const T& data) const;
63
64 //! Returns the current size of the list
65 std::size_t size() const;
66
67 //! Returns the maximum size of the list
68 std::size_t max_size() const;
69
70 //! Returns the element at index
71 T& at(std::size_t index);
72
73 //! Overloads the [] operator
74 T& operator[](std::size_t index);
75
76 //! Returns the element at index as constant
77 const T& at(std::size_t index) const;
78
79 //! Overloads the [] operator, but returns it as a constant
80 const T& operator[](std::size_t index) const;
81
82  private:
83 T* contents;
84 std::size_t size_;
85 std::size_t max_size_;
86 static const auto DEFAULT_MAX = 10u;
87 };
88 }  // namespace structures
89
90 template<typename T>
91 structures::ArrayList<T>::ArrayList() {
92     ArrayList(DEFAULT_MAX);
93 }
94
95 template<typename T>
96 structures::ArrayList<T>::ArrayList(std::size_t max_size) {
97     size_ = 0;
98     max_size_ = max_size;
99     contents = new T[max_size_];
100 }
101
102 template<typename T>
103 structures::ArrayList<T>::~ArrayList() {
104     delete[] contents;
105 }
106
107 template<typename T>
```

```cpp
void structures::ArrayList<T>::clear() {
    size_ = 0;
}

template<typename T>
void structures::ArrayList<T>::push_back(const T& data) {
    if (full()) {
        throw std::out_of_range("A lista esta cheia.");
    } else {
        contents[size_] = data;
        size_++;
    }
}

template<typename T>
void structures::ArrayList<T>::push_front(const T& data) {
    if (full()) {
        throw std::out_of_range("A lista esta cheia.");
    } else {
        for (int i = 0; i < size_; i++) {
            contents[size_ - i] = contents[size_ - i - 1];
        }
        size_++;
        contents[0] = data;
    }
}

template<typename T>
void structures::ArrayList<T>::insert(const T& data, std::size_t index) {
    if (full() || (index < 0 || index >= size_)) {
        throw std::out_of_range("A lista esta cheia.");
    } else {
        if (index == 0) {
            push_front(data);
            return;
        }
        if (index == size_) {
            push_back(data);
            return;
        }
        for (int i = 0; i < size_ - index; i++) {
            contents[size_ - i] = contents[size_ - i - 1];
        }
        size_++;
        contents[index] = data;
    }
}

template<typename T>
void structures::ArrayList<T>::insert_sorted(const T& data) {
    if (full()) {
        throw std::out_of_range("A lista esta cheia.");
    } else {
        for (int i = 0; i < size_; i++) {
            if (contents[i] >= data) {
                insert(data, i);
                return;
            }
        }
        push_back(data);
    }
}

template<typename T>
T structures::ArrayList<T>::pop(std::size_t index) {
    if (empty() || (index < 0 || index >= size_)) {
        throw std::out_of_range("A lista esta vazia.");
    } else {
        T removed_element = contents[index];
        for (int i = index; i < size_ - 1; i++) {
            contents[i] = contents[i + 1];
        }
        size_--;
        return removed_element;
    }
}

template<typename T>
T structures::ArrayList<T>::pop_back() {
    if (empty()) {
        throw std::out_of_range("A lista esta vazia");
    } else {
        size_--;
        return contents[size_];
    }
}

template<typename T>
T structures::ArrayList<T>::pop_front() {
    if (empty()) {
        throw std::out_of_range("A lista esta vazia");
    } else {
        T removed_element = contents[0];
        for (int i = 0; i < size_ - 1; i++) {
            contents[i] = contents[i + 1];
        }
        size_--;
        return removed_element;
    }
}

template<typename T>
void structures::ArrayList<T>::remove(const T& data) {
    if (empty()) {
        throw std::out_of_range("A lista esta vazia");
    } else {
        for (int i = 0; i < size_; i++) {
            if (contents[i] == data) {
                pop(i);
            }
        }
    }
}

template<typename T>
bool structures::ArrayList<T>::full() const {
    return (size_ == max_size_);
}

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

template<typename T>
bool structures::ArrayList<T>::contains(const T& data) const {
    for (int i = 0; i < size_; i++) {
        if (contents[i] == data) {
            return true;
        }
    }
    return false;
}

template<typename T>
std::size_t structures::ArrayList<T>::find(const T& data) const {
    for (int i = 0; i < size_; i++) {
        if (contents[i] == data) {
            return i;
        }
    }
    return size_;
}

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

template<typename T>
```

```
258 std::size_t structures::ArrayList<T>::max_size() const {
259     return max_size_;
260 }
261
262 template<typename T>
263 T& structures::ArrayList<T>::at(std::size_t index) {
264     if (empty() || (index < 0 || index >= size_)) {
265         throw std::out_of_range("Index invalido");
266     }
267     return contents[index];
268 }
269
270 template<typename T>
271 T& structures::ArrayList<T>::operator[](std::size_t index) {
272     return contents[index];
273 }
274
275 template<typename T>
276 const T& structures::ArrayList<T>::at(std::size_t index) const {
277     return contents[index];
278 }
279
280 template<typename T>
281 const T& structures::ArrayList<T>::operator[](std::size_t index) const {
282     return contents[index];
283 }
284
285 #endif
286
```