

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

Painel ▸ Agrupamentos de Turmas ▸ INE5408-03208A | INE5609-03238B (20182) ▸ Tópico 5 ▸ Implementação de Lista em vetor

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

Descrição Enviar Editar Visualizar envios

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 <stdint>
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 specific index
35         void insert(const T& data, std::size_t index);
36
37         ///! Inserts an element in a sorted position
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 is 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 containing 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 = 100u;
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>
```

```

108 void structures::ArrayList<T>::clear() {
109     size_ = 0;
110 }
111
112 template<typename T>
113 void structures::ArrayList<T>::push_back(const T& data) {
114     if (full()) {
115         throw std::out_of_range("A lista esta cheia.");
116     } else {
117         contents[size_] = data;
118         size_++;
119     }
120 }
121
122 template<typename T>
123 void structures::ArrayList<T>::push_front(const T& data) {
124     if (full()) {
125         throw std::out_of_range("A lista esta cheia.");
126     } else {
127         for (int i = 0; i < size_; i++) {
128             contents[size_ - i] = contents[size_ - i - 1];
129         }
130         size_++;
131         contents[0] = data;
132     }
133 }
134
135 template<typename T>
136 void structures::ArrayList<T>::insert(const T& data, std::size_t index) {
137     if (full() || (index < 0 || index >= size_)) {
138         throw std::out_of_range("A lista esta cheia.");
139     } else {
140         if (index == 0) {
141             push_front(data);
142             return;
143         }
144         if (index == size_) {
145             push_back(data);
146             return;
147         }
148         for (int i = 0; i < size_ - index; i++) {
149             contents[size_ - i] = contents[size_ - i - 1];
150         }
151         size_++;
152         contents[index] = data;
153     }
154 }
155
156 template<typename T>
157 void structures::ArrayList<T>::insert_sorted(const T& data) {
158     if (full()) {
159         throw std::out_of_range("A lista esta cheia.");
160     } else {
161         for (int i = 0; i < size_; i++) {
162             if (contents[i] >= data) {
163                 insert(data, i);
164                 return;
165             }
166         }
167         push_back(data);
168     }
169 }
170
171 template<typename T>
172 T structures::ArrayList<T>::pop(std::size_t index) {
173     if (empty() || (index < 0 || index >= size_)) {
174         throw std::out_of_range("A lista esta vazia.");
175     } else {
176         T removed_element = contents[index];
177         for (int i = index; i < size_ - 1; i++) {
178             contents[i] = contents[i + 1];
179         }
180         size_--;
181         return removed_element;
182     }
183 }
184
185 template<typename T>
186 T structures::ArrayList<T>::pop_back() {
187     if (empty()) {
188         throw std::out_of_range("A lista esta vazia.");
189     } else {
190         size_--;
191         return contents[size_];
192     }
193 }
194
195 template<typename T>
196 T structures::ArrayList<T>::pop_front() {
197     if (empty()) {
198         throw std::out_of_range("A lista esta vazia.");
199     } else {
200         T removed_element = contents[0];
201         for (int i = 0; i < size_ - 1; i++) {
202             contents[i] = contents[i + 1];
203         }
204         size_--;
205         return removed_element;
206     }
207 }
208
209 template<typename T>
210 void structures::ArrayList<T>::remove(const T& data) {
211     if (empty()) {
212         throw std::out_of_range("A lista esta vazia.");
213     } else {
214         for (int i = 0; i < size_; i++) {
215             if (contents[i] == data) {
216                 pop(i);
217             }
218         }
219     }
220 }
221
222 template<typename T>
223 bool structures::ArrayList<T>::full() const {
224     return (size_ == max_size_);
225 }
226
227 template<typename T>
228 bool structures::ArrayList<T>::empty() const {
229     return (size_ == 0);
230 }
231
232 template<typename T>
233 bool structures::ArrayList<T>::contains(const T& data) const {
234     for (int i = 0; i < size_; i++) {
235         if (contents[i] == data) {
236             return true;
237         }
238     }
239     return false;
240 }
241
242 template<typename T>
243 std::size_t structures::ArrayList<T>::find(const T& data) const {
244     for (int i = 0; i < size_; i++) {
245         if (contents[i] == data) {
246             return i;
247         }
248     }
249     return size_;
250 }
251
252 template<typename T>
253 std::size_t structures::ArrayList<T>::size() const {
254     return size_;
255 }
256
257 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 inválido");
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

```