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
      - Lista em vetor - aplicação de lista de ponteiros p...
        - Descrição
        - Enviar
        - Editar
        - Visualizar envios
      - Testes (lista de strings)
    - Tópico 8
    - Tópico 9
    - Tópico 10
- Meus cursos

ADMINISTRAÇÃO

- Administração do curso

Descrição | Enviar | Editar | Visualizar envios

Nota

Revisado em quarta, 5 Set 2018, 21:13 por Atribuição automática de nota
Nota 100 / 100
Relatório de avaliação
[+]**Summary of tests**
Enviado em quarta, 5 Set 2018, 21:13 (Baixar)

string_list.h

```
1  //! Copyright 2018 Matheus Henrique Schaly
2
3  #ifndef STRUCTURES_STRING_LIST_H
4  #define STRUCTURES_STRING_LIST_H
5
6  #include <cstdint>
7  #include <stdexcept>  // C++ exceptions
8  #include <cstring>
9
10
11 namespace structures {
12
13 //! Static list with pointers
14 template<typename T>
15 class ArrayList {
16  public:
17     //! Constructor
18     ArrayList();
19     //! Constructor with size parameter
20     explicit ArrayList(std::size_t max_size);
21     //! Desctructor
22     ~ArrayList();
23     //! Clears the list
24     void clear();
25     //! Pushes an element to the back of the list
26     void push_back(const T& data);
27     //! Pushes an element to the front of the list
28     void push_front(const T& data);
29     //! Inserts an element at a specific index
30     void insert(const T& data, std::size_t index);
31     //! Inserts an element in a sorted position
32     void insert_sorted(const T& data);
33     //! Removes an element from a specific index
34     T pop(std::size_t index);
35     //! Removes an element from the back of the list
36     T pop_back();
37     //! Removes an element from the front of the list
38     T pop_front();
39     //! Removes the first element containing the data
40     void remove(const T& data);
41     //! Verifies if the list is full
42     bool full() const;
43     //! Verifies if the list is empty
44     bool empty() const;
45     //! Verifies if the list contains the data
46     bool contains(const T& data) const;
47     //! Returns the index of the first element containing the data, else size
48     std::size_t find(const T& data) const;
49     //! Returns the current size of the list
50     std::size_t size() const;
51     //! Returns the maximum size of the list
52     std::size_t max_size() const;
53     //! Returns the element at index
54     T& at(std::size_t index);
55     //! Overloads the [] operator
56     T& operator[](std::size_t index);
57     //! Returns the element at index as constant
58     const T& at(std::size_t index) const;
59     //! Overloads the [] operator, but returns it as a constant
60     const T& operator[](std::size_t index) const;
61
62  private:
63     T* contents;
64     std::size_t size_;
65     std::size_t max_size_;
66
67     static const auto DEFAULT_MAX = 10u;
68 };
69
70 //! ArrayListString e' uma especializacao da classe ArrayList
71 class ArrayListString : public ArrayList<char *> {
72  public:
73     //! Constructor
74     ArrayListString() : ArrayList() {}
75     //! Constructor with parameter
76     explicit ArrayListString(std::size_t max_size) : ArrayList(max_size) {}
77     //! Destructor
78     ~ArrayListString();
79     //! Clears the list
80     void clear();
81     //! Inserts an element to the back of the list
82     void push_back(const char *data);
83     //! Inserts an element to the front of the list
84     void push_front(const char *data);
85     //! Inserts an element at a specific index
86     void insert(const char *data, std::size_t index);
87     //! Inserts an element in a sorted position
88     void insert_sorted(const char *data);
89     //! Removes an element from a specific index
90     char *pop(std::size_t index);
91     //! Removes an element from the back of the list
92     char *pop_back();
93     //! Removes an element from the front of the list
94     char *pop_front();
```

```
    //! Removes an element containing the data
    void remove(const char *data);
    //! Verifies if the list contains the data
    bool contains(const char *data);
    //! Returns the index of the first element containg the data, else size
    std::size_t find(const char *data);
};

}  // namespace structures

structures::ArrayListString::~ArrayListString() {
    clear();
}

void structures::ArrayListString::clear() {
    ArrayList::clear();
}

void structures::ArrayListString::push_back(const char *data) {
    if (full()) {
        throw std::out_of_range("A lista esta cheia.");
    } else {
        char *datanew = new char[strlen(data) + 1];
        snprintf(datanew, strlen(data)+1, "%s", data);
        ArrayList::push_back(datanew);
    }
}

void structures::ArrayListString::push_front(const char *data) {
    if (full()) {
        throw std::out_of_range("A lista esta cheia.");
    } else {
        char *datanew = new char[strlen(data) + 1];
        snprintf(datanew, strlen(data)+1, "%s", data);
        ArrayList::push_front(datanew);
    }
}

void structures::ArrayListString::insert(const char *data,
                                         std::size_t index) {
    if (full() || (index < 0 || index >= ArrayList::size())) {
        throw std::out_of_range("A lista esta cheia.");
    } else {
        char *datanew = new char[strlen(data) + 1];
        snprintf(datanew, strlen(data)+1, "%s", data);
        ArrayList::insert(datanew, index);
    }
}

void structures::ArrayListString::insert_sorted(const char *data) {
    if (full()) {
        throw std::out_of_range("A lista esta cheia.");
    } else {
        char *datanew = new char[strlen(data) + 1];
        snprintf(datanew, strlen(data)+1, "%s", data);
        for (std::size_t i = 0; i < ArrayList::size(); i++) {
            if (strcmp(ArrayList::at(i), data) > 0) {
                ArrayList::insert(datanew, i);
                return;
            }
        }
        ArrayList::push_back(datanew);
    }
}

char* structures::ArrayListString::pop(std::size_t index) {
    if (empty() || (index < 0 || index >= ArrayList::size())) {
        throw std::out_of_range("A lista esta vazia.");
    } else {
        char* removed_element = ArrayList::pop(index);
        return removed_element;
    }
}

char* structures::ArrayListString::pop_back() {
    if (empty()) {
        throw std::out_of_range("A lista esta vazia");
    } else {
        char* removed_element = ArrayList::pop_back();
        return removed_element;
    }
}

char* structures::ArrayListString::pop_front() {
    if (empty()) {
        throw std::out_of_range("A lista esta vazia");
    } else {
        char* removed_element = ArrayList::pop_front();
        return removed_element;
    }
}

void structures::ArrayListString::remove(const char *data) {
    if (empty()) {
        throw std::out_of_range("A lista esta vazia");
    } else {
        for (std::size_t i = 0; i < ArrayList::size(); i++) {
            if (strcmp(ArrayList::at(i), data) == 0) {
                pop(i);
            }
        }
    }
}

bool structures::ArrayListString::contains(const char *data) {
    for (std::size_t i = 0; i < ArrayList::size(); i++) {
        if (strcmp(ArrayList::at(i), data) == 0) {
            return true;
        }
    }
    return false;
}

std::size_t structures::ArrayListString::find(const char *data) {
    for (std::size_t i = 0; i < ArrayList::size(); i++) {
        if (strcmp(ArrayList::at(i), data) == 0) {
            return i;
        }
    }
    return ArrayList::size();
}

// Super Class

template<typename T>
structures::ArrayList<T>::ArrayList() {
    ArrayList(DEFAULT_MAX);
}
```

```cpp
template<typename T>
structures::ArrayList<T>::ArrayList(std::size_t max_size) {
    size_ = 0;
    max_size_ = max_size;
    contents = new T[max_size_];
}

template<typename T>
structures::ArrayList<T>::~ArrayList() {
    delete[] contents;
}

template<typename T>
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>
```

```cpp
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>
std::size_t structures::ArrayList<T>::max_size() const {
    return max_size_;
}

template<typename T>
T& structures::ArrayList<T>::at(std::size_t index) {
    if (empty() || (index < 0 || index >= size_)) {
        throw std::out_of_range("Index invalido");
    }
    return contents[index];
}

template<typename T>
T& structures::ArrayList<T>::operator[](std::size_t index) {
    return contents[index];
}

template<typename T>
const T& structures::ArrayList<T>::at(std::size_t index) const {
    return contents[index];
}

template<typename T>
const T& structures::ArrayList<T>::operator[](std::size_t index) const {
    return contents[index];
}

#endif

```