INE5609 - Lista Duplamente Encadeada

Generated by Doxygen 1.9.2

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 dll::Cursor Class Reference

`#include <cursor.hpp>`

Public Member Functions

- Cursor (Node *listHead=NULL)
- void proceedNPositions (int n)
- void regressNPositions (int n)
- void goToHead (void)
- void goToTail (void)
- Node * getCurrentNode (void)
- void setCurrentToNull (void)

3.1.1 Detailed Description

Classe representativa do cursor de uma lista encadeada

3.1.2 Constructor & Destructor Documentation

3.1.2.1 Cursor()

```
Cursor::Cursor (
            Node * listHead = NULL )
```

Construtor da classe (default constructor)

Parameters

listHead	HEAD da lista a qual o objeto cursor pertence

3.1.3 Member Function Documentation

3.1.3.1 getCurrentNode()

```
Node * Cursor::getCurrentNode (
            void  )
```

Retorna o elemento atual para qual o cursor aponta.

Returns

Ponteiro para elemento atual ou NULL em caso de uma lista que encontra-se vazia

3.1.3.2 goToHead()

```
void Cursor::goToHead (
            void  )
```

Move o cursor para a posicao HEAD

3.1.3.3 goToTail()

```
void Cursor::goToTail (
            void  )
```

Move o cursor para a posicao TAIL

3.1.3.4 proceedNPositions()

```
void Cursor::proceedNPositions (
            int n )
```

Move o cursor "n" posicoes em direcao a TAIL da lista

Parameters

n	Numero de posicoes a avancar

Note

Cursor sera posicionado em TAIL caso o numero de posicoes avancadas exceda o comprimento da lista ou caso o metodo seja executado em uma lista que encontra-se vazia

3.1.3.5 regressNPositions()

```
void Cursor::regressNPositions (
            int n )
```

Move o cursor "n" posicoes em direcao a HEAD da lista

Parameters

n	Numero de posicoes a regredir

Note

Cursor sera posicionado em HEAD caso o numero de posicoes regredidas exceda o comprimento da lista ou caso o metodo seja executado em uma lista que encontra-se vazia

3.1.3.6 setCurrentToNull()

```
void Cursor::setCurrentToNull (
            void  )
```

Metodo privado.

Atribui um ponteiro NULL para o atributo interno do elemento atual 'current'

The documentation for this class was generated from the following files:

- src/cursor.hpp
- src/cursor.cpp

3.2 dll::DoublyLinkedList Class Reference

```
#include <doublyLinkedList.hpp>
```

Public Member Functions

- DoublyLinkedList (void)
- Node * getCurrentNode (void)
- void insertBeforeCurrent (Node *node)
- void insertAfterCurrent (Node *node)
- void insertFirst (Node *node)
- void insertLast (Node *node)
- void insertAtIndex (long index, Node *node)
- void removeCurrent (void)
- void removeFirst (void)
- void removeLast (void)
- void removeByKey (long key)
- void removeFromIndex (long index)
- bool search (long key)
- bool isEmpty (void)
- bool isFull (void)
- long getIndexByKey (long key)

3.2.1 Detailed Description

Classe representativa de uma lista duplamente encadeada

3.2.2 Constructor & Destructor Documentation

3.2.2.1 DoublyLinkedList()

```
DoublyLinkedList::DoublyLinkedList (
            void  )
```

Construtor da classe (no-args constructor)

3.2.3 Member Function Documentation

3.2.3.1 getCurrentNode()

```
Node * DoublyLinkedList::getCurrentNode (
            void  )
```

Retorna elemento atual para qual o cursor da lista esta aponta

Returns

Ponteiro para nodo atual atribuido ao cursor ou NULL em caso de uma lista vazia

3.2.3.2 getIndexByKey()

```
long DoublyLinkedList::getIndexByKey (
            long key )
```

Metodo que retorna a chave de identificacao de um elemento com base em seu indice posicional dentro da lista encadeada

Parameters

key	Chave de identificacao numerica correspondente ao elemento desejado

Returns

Indice correspondente a posicao ocupada pelo elemento identificado pela chave 'key', caso presente na lista; caso contrario, '-1'

3.2.3.3 insertAfterCurrent()

```
void DoublyLinkedList::insertAfterCurrent (
            Node * node )
```

Metodo que permite inserir um nodo na posicao posterior ao atual elemento da lista para qual o cursor esta apontando

Note

Elemento adicionado a posicao HEAD e TAIL no caso de uma lista vazia

Parameters

node	Elemento a ser adicionado a lista

3.2.3.4 insertAtIndex()

```
void DoublyLinkedList::insertAtIndex (
            long index,
            Node * node )
```

Metodo que permite inserir um elemento em dada posicao da lista com base em um indice

Note

Em caso de uma lista vazia, o elemento sera adicionado as posicoes HEAD e TAIL

Parameters

index	Indice numerico (base zero) representativo da posicao na lista onde o sera inserido
node	Elemento a ser adicionado a lista

3.2.3.5 insertBeforeCurrent()

```
void DoublyLinkedList::insertBeforeCurrent (
            Node * node )
```

Metodo que permite inserir um nodo na posicao anterior ao atual elemento da lista para qual o cursor esta apontando

Note

Elemento adicionado a posicao HEAD e TAIL no caso de uma lista vazia

Parameters

node	Elemento a ser adicionado a lista

3.2.3.6 insertFirst()

```
void DoublyLinkedList::insertFirst (
            Node * node )
```

Metodo que permite adicionar um elemento a primeira posicao (HEAD) da lista encadeada

Note

Elemento sera adicionado a posicao HEAD e TAIL no caso de uma lista vazia

Parameters

node	Elemento a ser adicionado a lista

3.2.3.7 insertLast()

```
void DoublyLinkedList::insertLast (
            Node * node )
```

Metodo que permite adicionar um elemento a ultima posicao (TAIL) da lista encadeada

Note

Elemento sera adicionado a posicao HEAD e TAIL no caso de uma lista vazia

Parameters

node	Elemento a ser adicionado a lista

3.2.3.8 isEmpty()

```
bool DoublyLinkedList::isEmpty (
            void  )
```

Metodo que permite checar se a lista esta vazia, ou seja, nao possui elemento nenhum no momento de execucao do metodo

Returns

'true' caso a lista nao possua elementos, caso contrario, 'false'

3.2.3.9 isFull()

```
bool DoublyLinkedList::isFull (
            void  )
```

!!! EM CONSTRUCAO

3.2.3.10 removeByKey()

```
void DoublyLinkedList::removeByKey (
            long key )
```

Metodo que permite remover (liberar) um elemento da lista com base na chave de identificacao unica do nodo

Note

Nenhuma alteracao sera realizada se executado em uma chave invalida ou lista vazia

Parameters

key	Chave de identificacao numerica correspondente ao elemento que se deseja remover

3.2.3.11 removeCurrent()

```
void DoublyLinkedList::removeCurrent (
            void  )
```

Metodo que permite remover o elemento atual para o qual o cursor da lista esta apontando

Note

O cursor sera movido para a posicao anterior na lista ao remover o elemento atual, HEAD em casos onde restam apenas um elemento na lista, ou NULL se executado em uma lista com apenas um elemento. Nao gerara excecoes se executado em uma lista previamente vazia

3.2.3.12 removeFirst()

```
void DoublyLinkedList::removeFirst (
            void  )
```

Metodo que permite remover (liberar) o elemento na primeira posicao da lista

Note

HEAD sera atribuido ao elemento seguinte na lista, ou NULL no caso do metodo ser executado em uma lista com um unico elemento. O mesmo comportamento aplica-se ao cursor caso este esteja apontando para o primeiro elemento da lista.

3.2.3.13 removeFromIndex()

```
void DoublyLinkedList::removeFromIndex (
            long index )
```

Metodo que permite remover (liberar) um elemento da lista com base no indice posicional do nodo dentro da lista

Note

Nenhuma alteracao sera realizada se executado em um indice invalido invalida ou lista vazia

Parameters

index	Indice numerico (sequencial, base-zero) de identificacao correspondente a posicao na lista do elemento que se deseja remover

3.2.3.14 removeLast()

```
void DoublyLinkedList::removeLast (
            void  )
```

Metodo que permite remover (liberar) o elemento na ultima posicao da lista

Note

TAIL sera atribuido ao elemento anterior na lista, ou NULL no caso do metodo ser executado em uma lista com um unico elemento. O mesmo comportamento aplica-se ao cursor caso este esteja apontando para o ultimo elemento da lista.

3.2.3.15 search()

```
bool DoublyLinkedList::search (
            long key )
```

Metodo que permite buscar por um elemento em especifico na lista com base na chave unica de identificacao associada ao nodo em questao

Parameters

key	Chave de identificacao numerica correspondente ao elemento desejado

Returns

'true' se o elemento estiver presente na lista, caso contrario, 'false'

The documentation for this class was generated from the following files:

- src/doublyLinkedList.hpp
- src/doublyLinkedList.cpp

3.3 dll::Node Class Reference

```
#include <node.hpp>
```

Public Member Functions

- Node (int val=0, Node *nextNode=NULL, Node *prevNode=NULL)
- int getValue (void)

  Inicializando variavel estatica.
- long getKey (void)
- Node * getPrevNode (void)
- Node * getNextNode (void)
- void setValue (int val)
- void setPrevNode (Node *node)
- void setNextNode (Node *node)

3.3.1 Detailed Description

Classe representativa do elemento de uma lista duplamente encadeada

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Node()

```
Node::Node (
            int val = 0,
            Node * nextNode = NULL,
            Node * prevNode = NULL )
```

Construtor da classe (default constructor)

3.3.3 Member Function Documentation

3.3.3.1 getKey()

```
long Node::getKey (
            void  )
```

Metodo que retorna a chave de identificacao unica do nodo

Returns

Valor armazenado no campo privado 'key' do elemento

3.3.3.2 getNextNode()

```
Node * Node::getNextNode (
            void  )
```

Metodo que retorna o elemento posterior na lista encadeada

Returns

Ponteiro para nodo armazenado no campo privado 'next' do elemento ou NULL quando nao ha um elemento associado

3.3.3.3 getPrevNode()

```
Node * Node::getPrevNode (
            void  )
```

Metodo que retorna o elemento anterior na lista encadeada

Returns

Ponteiro para nodo armazenado no campo privado 'prev' do elemento ou NULL quando nao ha um elemento associado

3.3.3.4 getValue()

```
int Node::getValue (
            void  )
```

Inicializando variavel estatica.

Metodo que retorna o valor associado com o nodo

Returns

Valor numerico armazenado no campo privado 'value' do elemento

3.3.3.5 setNextNode()

```
void Node::setNextNode (
            Node * node )
```

Metodo que permite encadear um elemento posteriormente ao nodo na lista encadeada

Parameters

node	Ponteiro para nodo a ser armazenado no campo privado 'next' do elemento

3.3.3.6 setPrevNode()

```
void Node::setPrevNode (
            Node * node )
```

Metodo que permite encadear um elemento anteriormente ao nodo na lista encadeada

Parameters

node	Ponteiro para nodo a ser armazenado no campo privado 'prev' do elemento

3.3.3.7 setValue()

```
void Node::setValue (
            int val )
```

Metodo que permite atribuir um valor numerico ao nodo

Parameters

val	Valor a ser armazenado no campo privado 'value' do elemento

The documentation for this class was generated from the following files:

- src/node.hpp
- src/node.cpp

Chapter 4

File Documentation

4.1 src/cursor.cpp File Reference

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

```
#include "cursor.hpp"
```

4.1.1 Detailed Description

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

Author

Pedro Santi Binotto Contact: pedro.binotto@grad.ufsc.br
Joao Pedro Ziliotto Martinez Contact: joao.ziliotto@grad.ufsc.br

4.2 src/cursor.hpp File Reference

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

```
#include "node.hpp"
```

Classes

- class dll::Cursor

4.2.1 Detailed Description

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

Author

Pedro Santi Binotto Contact: pedro.binotto@grad.ufsc.br
Joao Pedro Ziliotto Martinez Contact: joao.ziliotto@grad.ufsc.br

4.3 cursor.hpp

Go to the documentation of this file.

```
1
13 #ifndef CURSOR_H
14 #define CURSOR_H
15
16 #include "node.hpp"
17
18 namespace dll {
22     class Cursor {
23     private:
24         Node **head;
25         Node *current;
26
27         bool regress(void);
28         bool proceed(void);
29     public:
35         Cursor(Node *listHead=NULL);
36
46         void proceedNPositions(int n);
47
57         void regressNPositions(int n);
58
62         void goToHead(void);
63
67         void goToTail(void);
68
75         Node *getCurrentNode(void);
76
83         void setCurrentToNull(void);
84     };
85 }
86
87 #endif
88
```

4.4 src/doublyLinkedList.cpp File Reference

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

```
#include "doublyLinkedList.hpp"
#include <iostream>
```

4.4.1 Detailed Description

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

Author

Pedro Santi Binotto Contact: pedro.binotto@grad.ufsc.br

Joao Pedro Ziliotto Martinez Contact: joao.ziliotto@grad.ufsc.br

4.5 src/doublyLinkedList.hpp File Reference

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

```
#include "cursor.hpp"
```

Classes

- class dll::DoublyLinkedList

4.5.1 Detailed Description

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

Author

Pedro Santi Binotto Contact: pedro.binotto@grad.ufsc.br
Joao Pedro Ziliotto Martinez Contact: joao.ziliotto@grad.ufsc.br

4.6 doublyLinkedList.hpp

Go to the documentation of this file.

```
1
13 #ifndef LINKEDLIST_H
14 #define LINKEDLIST_H
15
16 #include "cursor.hpp"
17
18 namespace dll {
22     class DoublyLinkedList {
23     private:
24         Cursor cursor;
25         Node **head;
26         Node **tail;
27
28         // TODO: implementar destructor -> freeList(void)->[tambem por fazer]
29     public:
33         DoublyLinkedList(void);
34
41         Node *getCurrentNode(void);
42
51         void insertBeforeCurrent(Node *node);
52
61         void insertAfterCurrent(Node *node);
62
71         void insertFirst(Node *node);
72
81         void insertLast(Node *node);
82
93         void insertAtIndex(long index, Node *node);
94
105         void removeCurrent(void);
106
116         void removeFirst(void);
117
127         void removeLast(void);
128
138         void removeByKey(long key);
139
150         void removeFromIndex(long index);
151
162         bool search(long key);
163
171         bool isEmpty(void);
172
176         bool isFull(void);
177
188         long getIndexByKey(long key);
189     };
190 }
191
192 #endif
193
```

4.7 src/main.cpp File Reference

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

```
#include <iostream>
```

Functions

- int **main** (void)

4.7.1 Detailed Description

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

Author

Pedro Santi Binotto Contact: pedro.binotto@grad.ufsc.br
Joao Pedro Ziliotto Martinez Contact: joao.ziliotto@grad.ufsc.br

4.8 src/node.cpp File Reference

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

```
#include "node.hpp"
```

4.8.1 Detailed Description

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

Author

Pedro Santi Binotto Contact: pedro.binotto@grad.ufsc.br
Joao Pedro Ziliotto Martinez Contact: joao.ziliotto@grad.ufsc.br

4.9 src/node.hpp File Reference

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

```
#include <string>
```

Classes

- class dll::Node

4.9.1 Detailed Description

Parte do projeto para a disciplina de Estruturas de Dados cod. INE5609.

Author

Pedro Santi Binotto Contact: pedro.binotto@grad.ufsc.br

Joao Pedro Ziliotto Martinez Contact: joao.ziliotto@grad.ufsc.br

4.10 node.hpp

Go to the documentation of this file.

```
1
13 #ifndef NODE_H
14 #define NODE_H
15
16 #include <string>
17
18 namespace dll {
22     class Node {
23     private:
24         long key;
25         int value;
26         Node *prev;
27         Node *next;
28
29         static long nextId;
30         static long generateNextKey(void);
31     public:
35         Node(int val=0, Node *nextNode=NULL, Node *prevNode=NULL);
36
43         int getValue(void);
44
51         long getKey(void);
52
60         Node *getPrevNode(void);
61
69         Node *getNextNode(void);
70
77         void setValue(int val);
78
86         void setPrevNode(Node *node);
87
95         void setNextNode(Node *node);
96
97         // TODO: Documentar generateNextKey()
98     };
99 }
100
101 // TODO: implementar NodeFactory
102
103 #endif
104
```

Index