

Pilha\_Encadeada

Generated by Doxygen 1.8.14



# Contents

<b>1</b>	<b>Class Index</b>	<b>1</b>
1.1	Class List . . . . .	1
<b>2</b>	<b>File Index</b>	<b>3</b>
2.1	File List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	ElementoLista Struct Reference . . . . .	5
3.1.1	Detailed Description . . . . .	5
3.2	ItemType Struct Reference . . . . .	5
3.2.1	Detailed Description . . . . .	6
3.3	pilha Struct Reference . . . . .	6
3.3.1	Detailed Description . . . . .	6
<b>4</b>	<b>File Documentation</b>	<b>7</b>
4.1	include/pilha.h File Reference . . . . .	7
4.1.1	Detailed Description . . . . .	8
4.1.2	Function Documentation . . . . .	8
4.1.2.1	create_stack() . . . . .	8
4.1.2.2	empty() . . . . .	9
4.1.2.3	exists() . . . . .	9
4.1.2.4	free_stack() . . . . .	10
4.1.2.5	isFULL() . . . . .	11
4.1.2.6	pop() . . . . .	11
4.1.2.7	print_pilha() . . . . .	12

4.1.2.8	push()	12
4.1.2.9	set_size()	13
4.1.2.10	tamanho()	14
4.1.2.11	top()	14
4.2	Source/pilha.c File Reference	15
4.2.1	Detailed Description	15
4.2.2	Function Documentation	15
4.2.2.1	create_stack()	16
4.2.2.2	empty()	17
4.2.2.3	exists()	17
4.2.2.4	free_stack()	18
4.2.2.5	isFULL()	19
4.2.2.6	pop()	19
4.2.2.7	print_pilha()	20
4.2.2.8	push()	20
4.2.2.9	set_size()	21
4.2.2.10	tamanho()	22
4.2.2.11	top()	22
4.3	Source/teste_pilha.c File Reference	23
4.3.1	Detailed Description	23
<b>Index</b>		<b>25</b>

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">ElementoLista</a>	Estrutura de um elemento da pilha . . . . .	5
<a href="#">ItemType</a>	Estrutura do tipo de dado utilizado no programa no caso Dado como <a href="#">ItemType</a> . . . . .	5
<a href="#">pilha</a>	Estrutura da cabeça de uma pilha . . . . .	6



## Chapter 2

# File Index

### 2.1 File List

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

include/ <a href="#">pilha.h</a>	Arquivo de definição de funções . . . . .	7
Source/ <a href="#">pilha.c</a>	Arquivo de implementação das funções . . . . .	15
Source/ <a href="#">teste_pilha.c</a>	Arquivo de testes das funções . . . . .	23





## Chapter 3

# Class Documentation

### 3.1 ElementoLista Struct Reference

Estrutura de um elemento da pilha.

```
#include <pilha.h>
```

#### Public Attributes

- [Dado](#) **pedaco**
- struct [ElementoLista](#) \* **next**

#### 3.1.1 Detailed Description

Estrutura de um elemento da pilha.

The documentation for this struct was generated from the following file:

- include/[pilha.h](#)

### 3.2 ItemType Struct Reference

Estrutura do tipo de dado utilizado no programa no caso Dado como [ItemType](#).

```
#include <pilha.h>
```

#### Public Attributes

- int **x**

### 3.2.1 Detailed Description

Estrutura do tipo de dado utilizado no programa no caso Dado como [ItemType](#).

The documentation for this struct was generated from the following file:

- [include/pilha.h](#)

## 3.3 pilha Struct Reference

Estrutura da cabeça de uma pilha.

```
#include <pilha.h>
```

### Public Attributes

- [Elemento](#) \* **inicio**
- int **size**
- int **max**

### 3.3.1 Detailed Description

Estrutura da cabeça de uma pilha.

The documentation for this struct was generated from the following file:

- [include/pilha.h](#)

## Chapter 4

# File Documentation

### 4.1 include/pilha.h File Reference

Arquivo de definição de funções.

```
#include <stdlib.h>
#include <stdio.h>
```

#### Classes

- struct [ItemType](#)  
*Estrutura do tipo de dado utilizado no programa no caso Dado como [ItemType](#).*
- struct [ElementoLista](#)  
*Estrutura de um elemento da pilha.*
- struct [pilha](#)  
*Estrutura da cabeça de uma pilha.*

#### Typedefs

- typedef struct [ItemType](#) [Dado](#)  
*Estrutura do tipo de dado utilizado no programa no caso Dado como [ItemType](#).*
- typedef struct [ElementoLista](#) [Elemento](#)  
*Estrutura de um elemento da pilha.*
- typedef struct [pilha](#) [Pilha](#)  
*Estrutura da cabeça de uma pilha.*

## Functions

- `Pilha * create_stack (Pilha *, int max_tam)`  
*Aloca memória para a pilha.*
- `int push (Pilha *p, Dado elem)`  
*Adiciona um dado para a pilha.*
- `int pop (Pilha *p)`  
*retira ultimo elemento da pilha*
- `int empty (Pilha *p)`  
*Verifica se a pilha está vazia.*
- `int print_pilha (Pilha *p)`  
*Printa toda a pilha.*
- `Pilha * free_stack (Pilha *p)`  
*Apaga a pilha da memória.*
- `int tamanho (Pilha *p)`  
*Retorna o tamanho atual da pilha.*
- `int isFULL (Pilha *p)`  
*Verifica se a pilha atingiu o tamanho máximo.*
- `int exists (Pilha *p)`  
*Verifica se a pilha foi alocada Nesse caso se ela existe.*
- `int top (Pilha *p, Dado *elem)`  
*Retorna elemento do topo da pilha.*
- `Pilha * set_size (Pilha *p, int tam)`  
*Redefine o tamanho da pilha.*

### 4.1.1 Detailed Description

Arquivo de definição de funções.

#### Author

Luís Eduardo

#### Date

2018-09-13

### 4.1.2 Function Documentation

#### 4.1.2.1 create\_stack()

```
Pilha* create_stack (  
    Pilha * p,  
    int max_tam )
```

Aloca memória para a pilha.

## Parameters

<i>p</i>	
<i>max_tam</i>	

## Returns

Pilha\*

```
19 {
20     if (!exists(p))
21     {
22         p = (Pilha *)malloc(sizeof(Pilha));
23         p->inicio = NULL;
24         p->max = max_tam;
25         p->size = 0;
26         return p;
27     }
28     else
29     {
30         return p;
31     }
32 }
```

## 4.1.2.2 empty()

```
int empty (
    Pilha * p )
```

Verifica se a pilha está vazia.

## Parameters

<i>p</i>	
----------	--

## Returns

int

```
120 {
121     if (exists(p))
122     {
123         return (p->inicio == NULL);
124     }
125     else
126     {
127         return -1;
128     }
129 }
130 }
```

## 4.1.2.3 exists()

```
int exists (
    Pilha * p )
```

Verifica se a pilha foi alocada Nesse caso se ela existe.

**Parameters**

<i>p</i>	
----------	--

**Returns**

int

```

232 {
233     if (p == NULL)
234     {
235         return 0;
236     }
237
238     else
239     {
240         return 1;
241     }
242 }
```

**4.1.2.4 free\_stack()**

```

Pilha* free_stack (
    Pilha * p )
```

Apaga a pilha da memória.

**Parameters**

<i>p</i>	
----------	--

**Returns**

Pilha\*

```

165 {
166     if (exists(p))
167     {
168         Elemento *temp = p->inicio;
169         Elemento *temp2;
170         while (temp != NULL)
171         {
172
173             temp2 = temp->next;
174             free(temp); /*desalocando os nos*/
175             temp = NULL;
176             temp = temp2;
177         }
178         free(p); /*desalocando a pilha*/
179         p = NULL;
180         return p;
181     }
182
183     else
184     {
185         return p;
186     }
187 }
```

## 4.1.2.5 isFULL()

```
int isFULL (
    Pilha * p )
```

Verifica se a pilha atingiu o tamanho máximo.

## Parameters

<i>p</i>	
----------	--

## Returns

int

```
213 {
214     if (exists(p))
215     {
216         return (p->size == p->max);
217     }
218 }
219
220 else
221 {
222     return -1;
223 }
224 }
```

## 4.1.2.6 pop()

```
int pop (
    Pilha * p )
```

retira ultimo elemento da pilha

## Parameters

<i>p</i>	
----------	--

## Returns

int

```
95 {
96     if (exists(p))
97     {
98         Elemento *remov_elemento;
99         if (p->size == 0)
100             return -1;
101         remov_elemento = p->inicio;
102         p->inicio = p->inicio->next;
103         free(remov_elemento);
104         p->size--;
105         return 0;
106     }
107
108     else
```

```

109     {
110         return -1;
111     }
112 }

```

#### 4.1.2.7 print\_pilha()

```

int print_pilha (
    Pilha * p )

```

Printa toda a pilha.

##### Parameters

<i>p</i>	
----------	--

##### Returns

int

```

138 {
139     if (exists(p))
140     {
141         Elemento *corrente;
142         int i;
143         corrente = p->inicio;
144         for (i = 0; i < p->size; ++i)
145         {
146             printf("\t\t%i\n", corrente->pedaco);
147             corrente = corrente->next;
148         }
149         return 0;
150     }
151 }
152
153 else
154 {
155     return -1;
156 }
157 }

```

#### 4.1.2.8 push()

```

int push (
    Pilha * p,
    Dado elem )

```

Adiciona um dado para a pilha.

##### Parameters

<i>p</i>	
<i>elem</i>	



**Returns**

int

```

41 {
42     if (exists(p))
43     {
44         if (p->size < p->max)
45         {
46             Elemento *novo_elemento;
47             if ((novo_elemento = (Elemento *)malloc(sizeof(Elemento))) == NULL)
48                 return -1;
49             novo_elemento->pedaco = elem;
50             novo_elemento->next = p->inicio;
51             p->inicio = novo_elemento;
52             p->size++;
53             return 0;
54         }
55         else
56         {
57             return -1;
58         }
59     }
60
61     else
62     {
63         return -1;
64     }
65 }

```

**4.1.2.9 set\_size()**

```

Pilha* set_size (
    Pilha * p,
    int tam )

```

Redefine o tamanho da pilha.

**Parameters**

<i>p</i>	
<i>tam</i>	

**Returns**

Pilha\*

```

251 {
252     if (exists(p))
253     {
254         p->max = tam;
255         return p;
256     }
257     else
258     {
259         return p;
260     }
261 }

```

#### 4.1.2.10 tamanho()

```
int tamanho (
    Pilha * p )
```

Retorna o tamanho atual da pilha.

##### Parameters

<i>p</i>	
----------	--

##### Returns

int

```
195 {
196     if (exists(p))
197     {
198         return p->size;
199     }
200
201     else
202     {
203         return -1;
204     }
205 }
```

#### 4.1.2.11 top()

```
int top (
    Pilha * p,
    Dado * elem )
```

Retorna elemento do topo da pilha.

##### Parameters

<i>p</i>	
<i>elem</i>	

##### Returns

int

```
74 {
75     if (exists(p))
76     {
77         Elemento *a;
78         a = p->inicio;
79         *elem = a->pedaco;
80         return 0;
81     }
82
83     else
84     {
85         return -1;
86     }
87 }
```

## 4.2 Source/pilha.c File Reference

Arquivo de implementação das funções.

```
#include <stdio.h>
#include <stdlib.h>
#include "../include/pilha.h"
```

### Functions

- `Pilha * create_stack (Pilha *p, int max_tam)`  
*Aloca memória para a pilha.*
- `int push (Pilha *p, Dado elem)`  
*Adiciona um dado para a pilha.*
- `int top (Pilha *p, Dado *elem)`  
*Retorna elemento do topo da pilha.*
- `int pop (Pilha *p)`  
*retira ultimo elemento da pilha*
- `int empty (Pilha *p)`  
*Verifica se a pilha está vazia.*
- `int print_pilha (Pilha *p)`  
*Printa toda a pilha.*
- `Pilha * free_stack (Pilha *p)`  
*Apaga a pilha da memória.*
- `int tamanho (Pilha *p)`  
*Retorna o tamanho atual da pilha.*
- `int isFULL (Pilha *p)`  
*Verifica se a pilha atingiu o tamanho máximo.*
- `int exists (Pilha *p)`  
*Verifica se a pilha foi alocada Nesse caso se ela existe.*
- `Pilha * set_size (Pilha *p, int tam)`  
*Redefine o tamanho da pilha.*

### 4.2.1 Detailed Description

Arquivo de implementação das funções.

#### Author

Luís Eduardo

#### Date

2018-09-13

### 4.2.2 Function Documentation

#### 4.2.2.1 create\_stack()

```
Pilha* create_stack (  
    Pilha * p,  
    int max_tam )
```

Aloca memória para a pilha.

## Parameters

<i>p</i>	
<i>max_tam</i>	

## Returns

Pilha\*

```
19 {
20     if (!exists(p))
21     {
22         p = (Pilha *)malloc(sizeof(Pilha));
23         p->inicio = NULL;
24         p->max = max_tam;
25         p->size = 0;
26         return p;
27     }
28     else
29     {
30         return p;
31     }
32 }
```

## 4.2.2.2 empty()

```
int empty (
    Pilha * p )
```

Verifica se a pilha está vazia.

## Parameters

<i>p</i>	
----------	--

## Returns

int

```
120 {
121     if (exists(p))
122     {
123         return (p->inicio == NULL);
124     }
125
126     else
127     {
128         return -1;
129     }
130 }
```

## 4.2.2.3 exists()

```
int exists (
    Pilha * p )
```

Verifica se a pilha foi alocada Nesse caso se ela existe.

**Parameters**

<i>p</i>	
----------	--

**Returns**

int

```

232 {
233     if (p == NULL)
234     {
235         return 0;
236     }
237
238     else
239     {
240         return 1;
241     }
242 }
```

**4.2.2.4 free\_stack()**

```

Pilha* free_stack (
    Pilha * p )
```

Apaga a pilha da memória.

**Parameters**

<i>p</i>	
----------	--

**Returns**

Pilha\*

```

165 {
166     if (exists(p))
167     {
168         Elemento *temp = p->inicio;
169         Elemento *temp2;
170         while (temp != NULL)
171         {
172
173             temp2 = temp->next;
174             free(temp); /*desalocando os nos*/
175             temp = NULL;
176             temp = temp2;
177         }
178         free(p); /*desalocando a pilha*/
179         p = NULL;
180         return p;
181     }
182
183     else
184     {
185         return p;
186     }
187 }
```

## 4.2.2.5 isFULL()

```
int isFULL (
    Pilha * p )
```

Verifica se a pilha atingiu o tamanho máximo.

## Parameters

<i>p</i>	
----------	--

## Returns

int

```
213 {
214     if (exists(p))
215     {
216         return (p->size == p->max);
217     }
218 }
219
220 else
221 {
222     return -1;
223 }
224 }
```

## 4.2.2.6 pop()

```
int pop (
    Pilha * p )
```

retira ultimo elemento da pilha

## Parameters

<i>p</i>	
----------	--

## Returns

int

```
95 {
96     if (exists(p))
97     {
98         Elemento *remov_elemento;
99         if (p->size == 0)
100             return -1;
101         remov_elemento = p->inicio;
102         p->inicio = p->inicio->next;
103         free(remov_elemento);
104         p->size--;
105         return 0;
106     }
107 }
108 else
```

```

109     {
110         return -1;
111     }
112 }

```

#### 4.2.2.7 print\_pilha()

```

int print_pilha (
    Pilha * p )

```

Printa toda a pilha.

##### Parameters

<i>p</i>	
----------	--

##### Returns

int

```

138 {
139     if (exists(p))
140     {
141         Elemento *corrente;
142         int i;
143         corrente = p->inicio;
144         for (i = 0; i < p->size; ++i)
145         {
146             printf("\t\t%i\n", corrente->pedaco);
147             corrente = corrente->next;
148         }
149         return 0;
150     }
151 }
152
153 else
154 {
155     return -1;
156 }
157 }

```

#### 4.2.2.8 push()

```

int push (
    Pilha * p,
    Dado elem )

```

Adiciona um dado para a pilha.

##### Parameters

<i>p</i>	
<i>elem</i>	



**Returns**

int

```
41 {
42     if (exists(p))
43     {
44         if (p->size < p->max)
45         {
46             Elemento *novo_elemento;
47             if ((novo_elemento = (Elemento *)malloc(sizeof(Elemento))) == NULL)
48                 return -1;
49             novo_elemento->pedaco = elem;
50             novo_elemento->next = p->inicio;
51             p->inicio = novo_elemento;
52             p->size++;
53             return 0;
54         }
55         else
56         {
57             return -1;
58         }
59     }
60
61     else
62     {
63         return -1;
64     }
65 }
```

**4.2.2.9 set\_size()**

```
Pilha* set_size (
    Pilha * p,
    int tam )
```

Redefine o tamanho da pilha.

**Parameters**

<i>p</i>	
<i>tam</i>	

**Returns**

Pilha\*

```
251 {
252     if (exists(p))
253     {
254         p->max = tam;
255         return p;
256     }
257     else
258     {
259         return p;
260     }
261 }
```

## 4.2.2.10 tamanho()

```
int tamanho (
    Pilha * p )
```

Retorna o tamanho atual da pilha.

## Parameters

<i>p</i>	
----------	--

## Returns

int

```
195 {
196     if (exists(p))
197     {
198         return p->size;
199     }
200
201     else
202     {
203         return -1;
204     }
205 }
```

## 4.2.2.11 top()

```
int top (
    Pilha * p,
    Dado * elem )
```

Retorna elemento do topo da pilha.

## Parameters

<i>p</i>	
<i>elem</i>	

## Returns

int

```
74 {
75     if (exists(p))
76     {
77         Elemento *a;
78         a = p->inicio;
79         *elem = a->pedaco;
80         return 0;
81     }
82
83     else
84     {
85         return -1;
86     }
87 }
```

## 4.3 Source/teste\_pilha.c File Reference

Arquivo de testes das funções.

```
#include <iostream>
#include "../include/pilha.h"
#include <gtest/gtest.h>
```

### Functions

- **TEST** (existence\_condition, testar\_existencia\_da\_pilha)  
*Constroi um novo objeto de Teste Testa condição de existência da pilha.*
- **TEST** (existence\_condition\_all\_functions, testar\_existencia\_todas\_funcoes)  
*Constroi um novo objeto de Teste Testa condição de existência de todas as funções.*
- **TEST** (create\_stack\_test, Criar\_pilha)  
*Constroi um novo objeto de Teste Testa a criação da pilha.*
- **TEST** (free\_pilha\_test, teste\_desalocamento\_da\_pilha)  
*Constroi um novo objeto de Teste Testa função free\_stack.*
- **TEST** (push\_pilha\_test, Adicionar\_elemento\_na\_pilha)  
*Constroi um novo objeto de Teste Testa função push.*
- **TEST** (pop\_pilha\_test, retirar\_elemento\_na\_pilha)  
*Constroi um novo objeto de Teste Testa função pop.*
- **TEST** (empty\_pilha\_test, checar\_pilha\_vazia)  
*Constroi um novo objeto de Teste Testa função empty.*
- **TEST** (isFULL\_test, checar\_pilha\_esta\_cheia)  
*Constroi um novo objeto de Teste Testa função is FULL.*
- **TEST** (set\_size\_test, checar\_se\_tamanho\_foi\_modificado)  
*Constroi um novo objeto de Teste Testa função set\_size.*
- **TEST** (top\_function\_test, testar\_top)  
*Constroi um novo objeto de Teste Testa função top.*
- **TEST** (tamanho\_function\_test, testar\_tamanho)  
*Constroi um novo objeto de Teste Testa função tamanho.*
- **TEST** (print\_function\_test, testar\_print)  
*Constroi um novo objeto de Teste Testa função print.*
- int **main** (int argc, char \*\*argv)

### 4.3.1 Detailed Description

Arquivo de testes das funções.

Author

Luís Eduardo

Date

2018-09-13



# Index

- create\_stack
  - [pilha.c](#), [15](#)
  - [pilha.h](#), [8](#)
- ElementoLista, [5](#)
- empty
  - [pilha.c](#), [17](#)
  - [pilha.h](#), [9](#)
- exists
  - [pilha.c](#), [17](#)
  - [pilha.h](#), [9](#)
- free\_stack
  - [pilha.c](#), [18](#)
  - [pilha.h](#), [10](#)
- include/pilha.h, [7](#)
- isFULL
  - [pilha.c](#), [18](#)
  - [pilha.h](#), [10](#)
- ItemType, [5](#)
- [pilha](#), [6](#)
- [pilha.c](#)
  - [create\\_stack](#), [15](#)
  - [empty](#), [17](#)
  - [exists](#), [17](#)
  - [free\\_stack](#), [18](#)
  - [isFULL](#), [18](#)
  - [pop](#), [19](#)
  - [print\\_pilha](#), [20](#)
  - [push](#), [20](#)
  - [set\\_size](#), [21](#)
  - [tamanho](#), [21](#)
  - [top](#), [22](#)
- [pilha.h](#)
  - [create\\_stack](#), [8](#)
  - [empty](#), [9](#)
  - [exists](#), [9](#)
  - [free\\_stack](#), [10](#)
  - [isFULL](#), [10](#)
  - [pop](#), [11](#)
  - [print\\_pilha](#), [12](#)
  - [push](#), [12](#)
  - [set\\_size](#), [13](#)
  - [tamanho](#), [13](#)
  - [top](#), [14](#)
- pop
  - [pilha.c](#), [19](#)
  - [pilha.h](#), [11](#)
- print\_pilha
  - [pilha.c](#), [20](#)
  - [pilha.h](#), [12](#)
- push
  - [pilha.c](#), [20](#)
  - [pilha.h](#), [12](#)
- set\_size
  - [pilha.c](#), [21](#)
  - [pilha.h](#), [13](#)
- Source/pilha.c, [15](#)
- Source/teste\_pilha.c, [23](#)
- tamanho
  - [pilha.c](#), [21](#)
  - [pilha.h](#), [13](#)
- top
  - [pilha.c](#), [22](#)
  - [pilha.h](#), [14](#)