

Pilha_Encadeada

Gerado por Doxygen 1.8.14

Sumário

1	Índice dos Componentes	1
1.1	Lista de Classes	1
2	Índice dos Arquivos	3
2.1	Lista de Arquivos	3
3	Classes	5
3.1	Referência da Estrutura ElementoLista	5
3.1.1	Descrição detalhada	5
3.2	Referência da Estrutura ItemType	5
3.2.1	Descrição detalhada	6
3.3	Referência da Estrutura pilha	6
3.3.1	Descrição detalhada	6
4	Arquivos	7
4.1	Referência do Arquivo include/pilha.h	7
4.1.1	Descrição detalhada	8
4.1.2	Funções	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	Referência do Arquivo Source/pilha.c	15
4.2.1	Descrição detalhada	15
4.2.2	Funções	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	Referência do Arquivo Source/teste_pilha.c	23
4.3.1	Descrição detalhada	23
Sumário		25

Capítulo 1

Índice dos Componentes

1.1 Lista de Classes

Aqui estão as classes, estruturas, uniões e interfaces e suas respectivas descrições:

ElementoLista	Estrutura de um elemento da pilha	5
ItemType	Estrutura do tipo de dado utilizado no programa no caso Dado como ItemType	5
pilha	Estrutura da cabeça de uma pilha	6

Capítulo 2

Índice dos Arquivos

2.1 Lista de Arquivos

Esta é a lista de todos os arquivos documentados e suas respectivas descrições:

include/ pilha.h	
Arquivo de definição de funções	7
Source/ pilha.c	
Arquivo de implementação das funções	15
Source/ teste_pilha.c	
Arquivo de testes das funções	23

Capítulo 3

Classes

3.1 Referência da Estrutura ElementoLista

Estrutura de um elemento da pilha.

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

Atributos Públicos

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

3.1.1 Descrição detalhada

Estrutura de um elemento da pilha.

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- include/[pilha.h](#)

3.2 Referência da Estrutura ItemType

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

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

Atributos Públicos

- int **x**

3.2.1 Descrição detalhada

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

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- `include/pilha.h`

3.3 Referência da Estrutura pilha

Estrutura da cabeça de uma pilha.

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

Atributos Públicos

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

3.3.1 Descrição detalhada

Estrutura da cabeça de uma pilha.

A documentação para essa estrutura foi gerada a partir do seguinte arquivo:

- `include/pilha.h`

Capítulo 4

Arquivos

4.1 Referência do Arquivo include/pilha.h

Arquivo de definição de funções.

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

Componentes

- 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.

Definições de Tipos

- 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.

Funções

- `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 Descrição detalhada

Arquivo de definição de funções.

Autor

Luís Eduardo

Data

2018-09-13

4.1.2 Funções

4.1.2.1 create_stack()

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

Aloca memória para a pilha.

Parâmetros

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

Retorna**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.

Parâmetros

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

Retorna**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.

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

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

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

int

```

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

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

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 Referência do Arquivo Source/pilha.c

Arquivo de implementação das funções.

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

Funções

- **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 Descrição detalhada

Arquivo de implementação das funções.

Autor

Luís Eduardo

Data

2018-09-13

4.2.2 Funções

4.2.2.1 create_stack()

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

Aloca memória para a pilha.

Parâmetros

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

Retorna**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.

Parâmetros

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

Retorna**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.

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

int

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

4.2.2.6 pop()

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

retira ultimo elemento da pilha

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

int

```

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

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

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.

Parâmetros

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

Retorna

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 Referência do Arquivo Source/teste_pilha.c

Arquivo de testes das funções.

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

Funções

- **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 Descrição detalhada

Arquivo de testes das funções.

Autor

Luís Eduardo

Data

2018-09-13

Índice Remissivo

- 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](#)