Pilha_vetor

Generated by Doxygen 1.8.14

Contents

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 ItemType Struct Reference

Tipo de elemento inserido na pilha.

`#include <pilha.h>`

Public Attributes

- int **x**

3.1.1 Detailed Description

Tipo de elemento inserido na pilha.

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

- include/pilha.h

3.2 pilha Struct Reference

Estrutura básica da pilha

`#include <pilha.h>`

Public Attributes

- Dado * **valor**
- int **max_tam**
- int **size**

3.2.1 Detailed Description

Estrutura básica da 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.

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

Classes

- struct ItemType

  Tipo de elemento inserido na pilha.
- struct pilha

  Estrutura básica da pilha

Typedefs

- typedef struct ItemType Dado

  Tipo de elemento inserido na pilha.
- typedef struct pilha Pilha

  Estrutura básica da pilha

Functions

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

  Criar estrutura básica da pilha com tamanho máximo definido.
- int top (Pilha *p, Dado element)

  Pega primeiro elemento da pilha.
- int push (Pilha *p, Dado elem)

  Adiciona elemento para a pilha.
- int pop (Pilha *p, Dado *elemento)

Retira ultimo elemento da pilha.

- int empty (Pilha *p)

  Verifica se a pilha esta vazia.
- int print_pilha (Pilha *p)

  printa todos os elementos da pilha
- Pilha * free_stack (Pilha *p)

  limpa a pilha
- int tamanho (Pilha *p)

  verifica tamalho da pilha
- int isFULL (Pilha *p)

  verifica se a pilha atinge o tamanho maximo
- int exists (Pilha *p)

  verifica se a pilha existe
- int top (Pilha *p, Dado *elemento)

  Acessa ultimo elemento da pilha.
- Pilha * set_size (Pilha *p, int tamanho)

  Redefine o tamanho maximo da pilha.

4.1.1 Detailed Description

Arquivo de definição.

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 )
```

Criar estrutura básica da pilha com tamanho máximo definido.

Parameters

p	
max_tam	

Returns

Pilha*

```
20 {
21     if (!exists(p))
22     {
23         p = (Pilha *)malloc(sizeof(Pilha));
24         p->valor = (Dado *)malloc(max_tam * sizeof(Dado));
25         p->max_tam = max_tam;
26         p->size = 0;
27         return p;
28     }
29
30     else
31     {
32         return p;
33     }
34 }
```

4.1.2.2 empty()

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

Verifica se a pilha esta vazia.

Parameters

p	

Returns

int

```
99 {
100
101     if (exists(p))
102     {
103         return (p->size == 0);
104     }
105
106     else
107     {
108         return -1;
109     }
110 }
```

4.1.2.3 exists()

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

verifica se a pilha existe

Parameters

p	

Returns

int

```
203 {
204     if (p == NULL)
205     {
206         return 0;
207     }
208 
209     else
210     {
211         return 1;
212     }
213 }
```

4.1.2.4 free_stack()

```
Pilha* free_stack (
            Pilha * p )
```

limpa a pilha

Parameters

p	

Returns

Pilha*

```
141 {
142     if (exists(p))
143     {
144         free(p->valor);
145         p->valor = NULL;
146         free(p);
147         p = NULL;
148         return p;
149     }
150 
151     else
152     {
153 
154         return p;
155     }
156 }
```

4.1.2.5 isFULL()

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

verifica se a pilha atinge o tamanho maximo

Parameters

p	

Returns

int

```
183 {
184     if (exists(p))
185     {
186 
187         return (p->size == p->max_tam);
188     }
189 
190     else
191     {
192 
193         return -1;
194     }
195 }
```

4.1.2.6 pop()

```
int pop (
            Pilha * p,
            Dado * elemento )
```

Retira ultimo elemento da pilha.

Parameters

p	
elemento	

Returns

int

```
74 {
75 
76     if (exists(p))
77     {
78 
79         if (p->size == 0)
80             return -1;
81         p->size--;
82         *elemento = p->valor[p->size];
83         return 0;
84     }
85 
86     else
87     {
88 
89         return -1;
90     }
91 }
```

4.1.2.7 print_pilha()

```
int print_pilha (
            Pilha * p )
```

printa todos os elementos da pilha

Parameters

p	

Returns

int

```
118 {
119     if (exists(p))
120     {
121 
122         for (int i = 0; i < p->size; ++i)
123         {
124             printf("\t\t%d\n", p->valor[i]);
125         }
126         return 0;
127     }
128 
129     else
130     {
131         return -1;
132     }
133 }
```

4.1.2.8 push()

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

Adiciona elemento para a pilha.

Parameters

p	
elem	

Returns

int

```
44 {
45     if (exists(p))
46     {
47         if (p->size < p->max_tam)
48         {
49             p->valor[p->size] = elem;
50             p->size++;
```

```
51              return 0;
52          }
53          else
54          {
55
56              return -1;
57          }
58      }
59
60      else
61      {
62
63          return -1;
64      }
65 }
```

4.1.2.9 set_size()

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

Redefine o tamanho maximo da pilha.

Parameters

p	
tamanho	

Returns

Pilha*

Parameters

p	
tam	

Returns

Pilha*

```
246 {
247     if (exists(p) && tam > p->size)
248     {
249
250         p->max_tam = tam;
251         return p;
252     }
253
254     else
255     {
256         return p;
257     }
258 }
```

4.1.2.10 tamanho()

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

verifica tamalho da pilha

Parameters

p	

Returns

int

```
164 {
165     if (exists(p))
166     {
167         return p->size;
168     }
169 
170     else
171     {
172 
173         return -1;
174     }
175 }
```

4.1.2.11 top() [1/2]

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

Pega primeiro elemento da pilha.

Parameters

p	
element	

Returns

int

4.1.2.12 top() [2/2]

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

Acessa ultimo elemento da pilha.

Parameters

p	
elemento	

Returns

int

```
222 {
223     if (exists(p))
224     {
225 
226         if (p->size == 0)
227             return -1;
228         *elemento = p->valor[p->size - 1];
229         return 0;
230     }
231 
232     else
233     {
234 
235         return -1;
236     }
237 }
```

4.2 Source/pilha.c File Reference

Arquivo de funções.

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

Functions

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

  Criar estrutura básica da pilha com tamanho máximo definido.
- int push (Pilha *p, Dado elem)

  Adiciona elemento para a pilha.
- int pop (Pilha *p, Dado *elemento)

  Retira ultimo elemento da pilha.
- int empty (Pilha *p)

  Verifica se a pilha esta vazia.
- int print_pilha (Pilha *p)

  printa todos os elementos da pilha
- Pilha * free_stack (Pilha *p)

  limpa a pilha
- int tamanho (Pilha *p)

  verifica tamalho da pilha
- int isFULL (Pilha *p)

  verifica se a pilha atinge o tamanho maximo
- int exists (Pilha *p)

  verifica se a pilha existe
- int top (Pilha *p, Dado *elemento)

  Acessa ultimo elemento da pilha.
- Pilha * set_size (Pilha *p, int tam)

  Redefine o tamanho maximo da pilha.

4.2.1 Detailed Description

Arquivo de 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 )
```

Criar estrutura básica da pilha com tamanho máximo definido.

Parameters

p	
max_tam	

Returns

Pilha*

```
20 {
21     if (!exists(p))
22     {
23         p = (Pilha *)malloc(sizeof(Pilha));
24         p->valor = (Dado *)malloc(max_tam * sizeof(Dado));
25         p->max_tam = max_tam;
26         p->size = 0;
27         return p;
28     }
29
30     else
31     {
32         return p;
33     }
34 }
```

4.2.2.2 empty()

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

Verifica se a pilha esta vazia.

Parameters

p	

Returns

int

```
99 {
100
101     if (exists(p))
102     {
103         return (p->size == 0);
104     }
105
106     else
107     {
108         return -1;
109     }
110 }
```

4.2.2.3 exists()

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

verifica se a pilha existe

Parameters

p	

Returns

int

```
203 {
204     if (p == NULL)
205     {
206         return 0;
207     }
208
209     else
210     {
211         return 1;
212     }
213 }
```

4.2.2.4 free_stack()

```
Pilha* free_stack (
              Pilha * p )
```

limpa a pilha

Parameters

p	

Returns

Pilha*

```
141 {
142     if (exists(p))
143     {
144         free(p->valor);
145         p->valor = NULL;
146         free(p);
147         p = NULL;
148         return p;
149     }
150 
151     else
152     {
153 
154         return p;
155     }
156 }
```

4.2.2.5 isFULL()

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

verifica se a pilha atinge o tamanho maximo

Parameters

p	

Returns

int

```
183 {
184     if (exists(p))
185     {
186 
187         return (p->size == p->max_tam);
188     }
189 
190     else
191     {
192 
193         return -1;
194     }
195 }
```

4.2.2.6 pop()

```
int pop (
            Pilha * p,
            Dado * elemento )
```

Retira ultimo elemento da pilha.

Parameters

p	
elemento	

Returns

int

```
74 {
75
76     if (exists(p))
77     {
78
79         if (p->size == 0)
80             return -1;
81         p->size--;
82         *elemento = p->valor[p->size];
83         return 0;
84     }
85
86     else
87     {
88
89         return -1;
90     }
91 }
```

4.2.2.7 print_pilha()

```
int print_pilha (
            Pilha * p )
```

printa todos os elementos da pilha

Parameters

p	

Returns

int

```
118 {
119     if (exists(p))
120     {
121
122         for (int i = 0; i < p->size; ++i)
```

```
123         {
124             printf("\t\t%d\n", p->valor[i]);
125         }
126         return 0;
127     }
128
129     else
130     {
131         return -1;
132     }
133 }
```

4.2.2.8 push()

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

Adiciona elemento para a pilha.

Parameters

p	
elem	

Returns

int

```
44 {
45     if (exists(p))
46     {
47         if (p->size < p->max_tam)
48         {
49             p->valor[p->size] = elem;
50             p->size++;
51             return 0;
52         }
53         else
54         {
55
56             return -1;
57         }
58     }
59
60     else
61     {
62
63         return -1;
64     }
65 }
```

4.2.2.9 set_size()

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

Redefine o tamanho maximo da pilha.

Parameters

p	
tam	

Returns

Pilha*

```
246 {
247     if (exists(p) && tam > p->size)
248     {
249 
250         p->max_tam = tam;
251         return p;
252     }
253 
254     else
255     {
256         return p;
257     }
258 }
```

4.2.2.10 tamanho()

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

verifica tamalho da pilha

Parameters

p	

Returns

int

```
164 {
165     if (exists(p))
166     {
167         return p->size;
168     }
169 
170     else
171     {
172 
173         return -1;
174     }
175 }
```

4.2.2.11 top()

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

Acessa ultimo elemento da pilha.

Parameters

p	
elemento	

Returns

int

```
222 {
223     if (exists(p))
224     {
225 
226         if (p->size == 0)
227             return -1;
228         *elemento = p->valor[p->size - 1];
229         return 0;
230     }
231 
232     else
233     {
234 
235         return -1;
236     }
237 }
```

4.3 Source/teste_pilha.c File Reference

Arquivo de testes.

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

Functions

- TEST (Pilha_exists_tests, test_existences)

  Constroi um novo objeto de Teste Teste da função exists.
- TEST (create_stack_test, Criar_pilha)

  Constroi um novo objeto de Teste Teste da função create_stack.
- TEST (push_pilha_test, Adicionar_elemento_na_pilha)

  Constroi um novo objeto de Teste Teste da função push_pilha.
- TEST (pop_pilha_test, retirar_elemento_na_pilha)

  Constroi um novo objeto de Teste Teste da função pop.
- TEST (empty_pilha_test, checar_pilha_vazia)

  Constroi um novo objeto de Teste Teste da função empty.
- TEST (isFULL_test, checar_pilha_esta_cheia)

  Constroi um novo objeto de Teste Teste da função isFULL.
- TEST (Pilha_exists_test, checar_se_a_pilha_existe)

  Constroi um novo objeto de Teste Teste da função exists 2.
- TEST (Pilha_top_access, checar_primeiro_elemento_da_pilha)

  Constroi um novo objeto de Teste Teste da função top.
- TEST (pilha_set_size_test, testar_alteracao_tamanho)

  Constroi um novo objeto de Teste Teste da função set_size.

- TEST (print_pilha_test, testar_print_da_pilha)

  Constroi um novo objeto de Teste Teste da função print.
- TEST (free_stack_test, testar_free_stack_function)

  Constroi um novo objeto de Teste Teste da função free_stack.
- TEST (tamanho_test, testar_tamanho_da_pilha)

  Constroi um novo objeto de Teste Teste da função tamanho.
- int **main** (int argc, char ∗∗argv)

4.3.1 Detailed Description

Arquivo de testes.

Author

Luís Eduardo

Date

2018-09-13

Index