

Estrutura de Dados

Aula 9 : Pilhas (Stack)

Prof. MSc. Fausto Sampaio

https://github.com/Fausto14/estrutura_de_dados

Centro Universitário UniFanor - Wyden

22 de novembro de 2019

- 1 Pilhas
 - Definição
 - Aplicações
 - Operações
- 2 Pilha Estática
 - Definição
 - Implementação
- 3 Pilha Dinâmica
 - Definição
 - Implementação
- 4 Exemplos
- 5 Referências

Pilhas

Definição

- Uma estrutura do tipo **Pilha** é uma sequência de elementos do mesmo tipo, como as Listas e Filas;
- Seus elementos possuem estrutura interna abstraída, ou seja, sua complexidade é arbitrária e não afeta o seu funcionamento.



Pilha



- Uma **Pilha** é um tipo especial de lista onde inserções e exclusões de elementos ocorrem apenas no início da lista (topo);
- São estruturas de dados do tipo LIFO (last-in first-out) - o último elemento a ser inserido, será o primeiro a ser retirado.
- A manipulação dos elementos é dada apenas por uma das extremidades da lista - topo;
- Para processar o penúltimo item inserido, deve-se remover o último.
- Exemplos de pilhas são:
 - pilha de pratos;
 - pilha de livros;
 - pilha de cartas de um baralho;
 - etc.

- Usa-se pilha em aplicações em que os dados são obtidos na ordem inversa àquela em que foram fornecidos.
- Exemplos:
 - Mecanismo de fazer/desfazer de um editor de texto;
 - Mecanismo de navegação de páginas na Internet (avançar e retornar).
 - Retirada de mercadorias de um caminhão de entregas;
 - Guardar um caminho de volta (**backtracking**) - labirinto;
 - Conversão de número decimal para binário;
 - etc.

- Em uma Pilha podemos realizar as seguintes operações básicas:
 - criação da pilha;
 - inserção de um elemento no topo - **PUSH**;
 - exclusão de um elemento no topo - **POP**;
 - acesso ou consulta ao elemento do topo - **TOP**;
 - destruição da pilha
- Essas operações dependem do tipo de alocação de memória usada:
 - estática;
 - dinâmica;

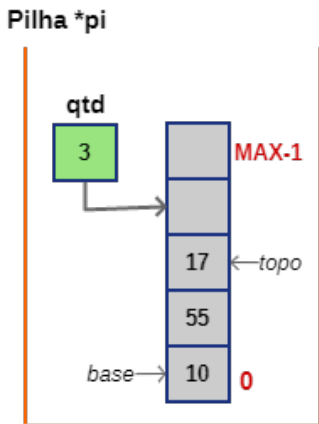
- O espaço de memória é alocado no momento da compilação;
- Exige a definição do número máximo de elementos da Pilha;
- Acesso sequencial: elementos consecutivos na memória;

- O espaço de memória é alocado em tempo de execução;
- A Pilha cresce à medida que novos elementos são armazenados, e diminui à medida que elementos são removidos;
- Acesso encadeado: cada elemento pode estar em uma área distinta da memória;
- Para acessar um elemento, é preciso percorrer todos os antecessores na Pilha.
- O acesso é através de ponteiros! Não tem índices!

Pilha Estática

Definição

- **Pilha Estática** : Tipo de Pilha onde o sucessor de um elemento ocupa a posição física seguinte do mesmo;
- Uso de vetores (array).



PilhaEstatica.h

- os protótipos das funções;
- o tipo de dado armazenado na pilha;
- o ponteiro "pilha".
- tamanho do vetor usado na pilha.

PilhaEstatica.c

- o tipo de dados "pilha";
- implementar as suas funções.

PilhaEstatica.h

```
2  #define MAX 100
3  struct aluno{
4      int matricula;
5      char nome[30];
6      float n1,n2,n3;
7  };
8
9  typedef struct pilha Pilha;
10
11 Pilha* cria_Pilha();
12 void libera_Pilha(Pilha* pi);
13 int consulta_topo_Pilha(Pilha* pi, struct aluno *al);
14 int insere_Pilha(Pilha* pi, struct aluno al);
15 int remove_Pilha(Pilha* pi);
16 int tamanho_Pilha(Pilha* pi);
17 int Pilha_vazia(Pilha* pi);
18 int Pilha_cheia(Pilha* pi);
19 void imprime_Pilha(Pilha* pi);
```

Definição - Pilha Estática

PilhaEstatica.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "PilhaEstatica.h" //inclui os Protótipos
4
5  //Definição do tipo Pilha
6  struct pilha{
7      int qtd;
8      struct aluno dados[MAX];
9  };
```

Criar e Liberar - Pilha Estática

PilhaEstatica.c

```
11 Pilha* cria_Pilha(){
12     Pilha *pi;
13     pi = (Pilha*) malloc(sizeof(struct pilha));
14     if(pi != NULL)
15         pi->qtd = 0;
16     return pi;
17 }
18
19 void libera_Pilha(Pilha* pi){
20     free(pi);
21 }
```

Informações da Pilha Estática

```
47 □ int tamanho_Pilha(Pilha* pi){  
48     if(pi == NULL)  
49         return -1;  
50     else  
51         return pi->qtd;  
52 }
```

(a) Tamanho da Pilha

```
54 □ int Pilha_cheia(Pilha* pi){  
55     if(pi == NULL)  
56         return -1;  
57     return (pi->qtd == MAX);  
58 }
```

(b) Pilha Cheia

```
60 □ int Pilha_vazia(Pilha* pi){  
61     if(pi == NULL)  
62         return -1;  
63     return (pi->qtd == 0);  
64 }
```

(c) Pilha Vazia

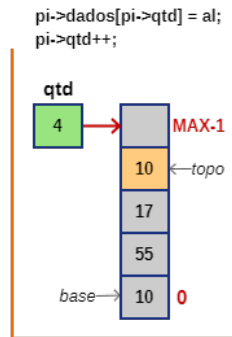
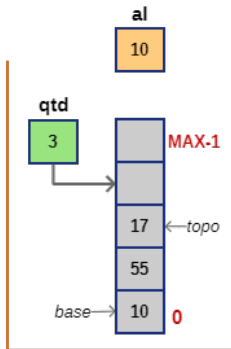

```
66 void imprime_Pilha(Pilha* pi){
67     if(pi == NULL)
68         return;
69     int i;
70     for(i=pi->qtd-1; i >=0; i--){
71         printf("Matricula: %d\n",pi->dados[i].matricula);
72         printf("Nome: %s\n",pi->dados[i].nome);
73         printf("Notas: %f %f %f\n",pi->dados[i].n1,
74             pi->dados[i].n2,
75             pi->dados[i].n3);
76         printf("-----\n");
77     }
78 }
```

Inserir Elemento - Pilha Estática

- Empilhar - **PUSH**;
- Em uma pilha a inserção é sempre no seu topo;
- Cuidado: não se pode inserir numa pilha cheia (estouro da pilha);


```
30 int insere_Pilha(Pilha* pi, struct aluno al){
31     if(pi == NULL)
32         return 0;
33     if(pi->qtd == MAX)//pilha cheia
34         return 0;
35     pi->dados[pi->qtd] = al;
36     pi->qtd++;
37     return 1;
38 }
```

Inserir Elemento - Pilha Estática

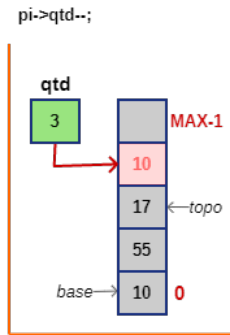
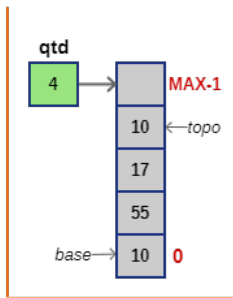


Remover Elemento - Pilha Estática

- Desempilhar - **POP**;
- Em uma pilha a remoção é sempre no seu topo;
- Cuidado: não se pode remover de uma pilha vazia;

```
40  int remove_Pilha(Pilha* pi){  
41     if(pi == NULL || pi->qtd == 0)  
42         return 0;  
43     pi->qtd--;  
44     return 1;  
45 }
```

Remover Elemento - Pilha Estática



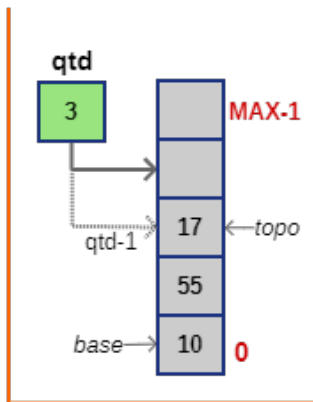
Consultar Elemento - Pilha Estática

- Em uma pilha a consulta se dá apenas ao elemento que está no seu topo;
- Consultar o topo - **TOP**;
- **Atenção:** a consulta não remove e também não insere elementos na pilha;

```
23 int consulta_topo_Pilha(Pilha* pi, struct aluno *al){  
24     if(pi == NULL || pi->qtd == 0)  
25         return 0;  
26     *al = pi->dados[pi->qtd-1];  
27     return 1;  
28 }
```

Consultar Elemento - Pilha Estática

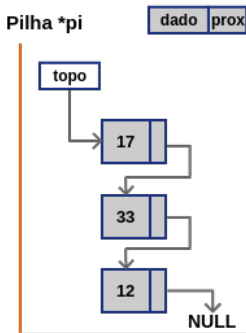
```
*al = pi->dados[pi->qtd-1];
```



Pilha Dinâmica

Definição

- **Pilha Dinâmica** : Tipo de Pilha onde cada elemento aponta para o seu sucessor na Pilha;
- Usa um ponteiro o primeiro elemento (topo) da pilha e uma indicação de final de pilha (NULL);
- Implementação muito semelhante às Listas Dinâmicas;



PilhaDin.h

- os protótipos das funções;
- o tipo de dado armazenado na pilha;
- o ponteiro "pilha".

PilhaDin.c

- o tipo de dados "pilha";
- implementar as suas funções.

PilhaDin.h

```
3 struct aluno{
4     int matricula;
5     char nome[30];
6     float n1,n2,n3;
7 };
8
9 typedef struct elemento* Pilha;
10
11 Pilha* cria_Pilha();
12 void libera_Pilha(Pilha* pi);
13 int consulta_topo_Pilha(Pilha* pi, struct aluno *al);
14 int insere_Pilha(Pilha* pi, struct aluno al);
15 int remove_Pilha(Pilha* pi);
16 int tamanho_Pilha(Pilha* pi);
17 int Pilha_vazia(Pilha* pi);
18 int Pilha_cheia(Pilha* pi);
19 void imprime_Pilha(Pilha* pi);
20
```

Definição - Pilha Dinâmica

PilhaDin.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "PilhaDin.h" //inclui os Protótipos
4
5  //Definição do tipo Pilha
6  struct elemento{
7      struct aluno dados;
8      struct elemento *prox;
9  };
10 typedef struct elemento Elem;
```

Criar e Liberar - Pilha Dinâmica

PilhaDin.c

```
12 Pilha* cria_Pilha(){
13     Pilha* pi = (Pilha*) malloc(sizeof(Pilha));
14     if(pi != NULL)
15         *pi = NULL;
16     return pi;
17 }
18
19 void libera_Pilha(Pilha* pi){
20     if(pi != NULL){
21         Elem* no;
22         while((*pi) != NULL){
23             no = *pi;
24             *pi = (*pi)->prox;
25             free(no);
26         }
27         free(pi);
28     }
29 }
```

Informações da Pilha Dinâmica

```
64 int tamanho_Pilha(Pilha* pi){
65     if(pi == NULL)
66         return 0;
67     int cont = 0;
68     Elem* no = *pi;
69     while(no != NULL){
70         cont++;
71         no = no->prox;
72     }
73     return cont;
74 }
```

(a) Tamanho da Pilha Dinâmica

```
76 int Pilha_cheia(Pilha* pi){
77     return 0;
78 }
```

(b) Pilha Dinâmica Cheia

```
80 int Pilha_vazia(Pilha* pi){
81     if(pi == NULL)
82         return 1;
83     if(*pi == NULL)
84         return 1;
85     return 0;
86 }
```

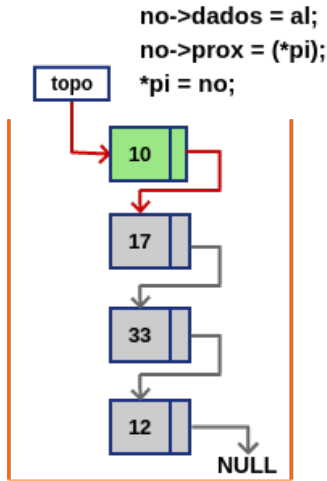
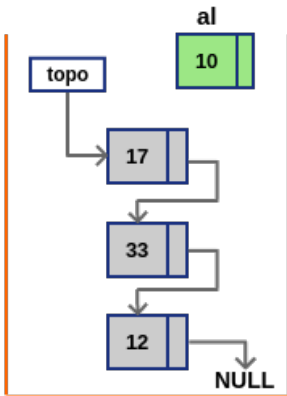
(c) Pilha Dinâmica Vazia

Inserir Elemento - Pilha Dinâmica

- Empilhar - **PUSH**;
- Em uma Pilha a inserção é sempre no seu topo;
- **Lembrar**: o início de uma pilha é equivalente ao seu topo;

```
40 int insere_Pilha(Pilha* pi, struct aluno al){
41     if(pi == NULL)
42         return 0;
43     Elem* no;
44     no = (Elem*) malloc(sizeof(Elem));
45     if(no == NULL)
46         return 0;
47     no->dados = al;
48     no->prox = (*pi);
49     *pi = no;
50     return 1;
51 }
```

Inserir Elemento - Pilha Dinâmica

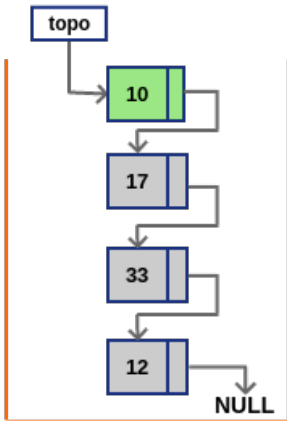


Remover Elemento - Pilha Dinâmica

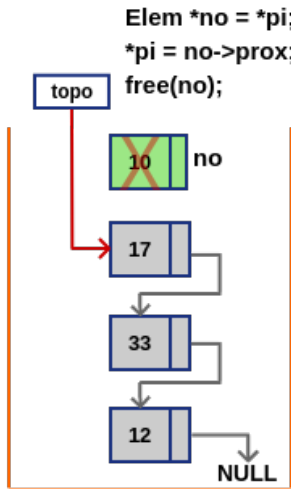
- Desempilhar - **POP**;
- Em uma pilha a remoção é sempre no seu início ou topo;
- Cuidado: não se pode remover de uma pilha vazia;

```
53 int remove_Pilha(Pilha* pi){  
54     if(pi == NULL)  
55         return 0;  
56     if((*pi) == NULL)  
57         return 0;  
58     Elem *no = *pi;  
59     *pi = no->prox;  
60     free(no);  
61     return 1;  
62 }
```

Remover Elemento - Pilha Dinâmica



.....

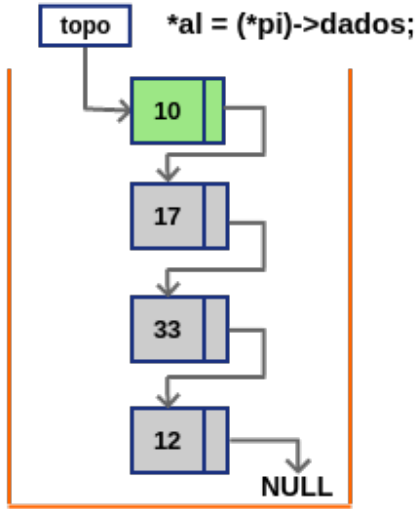


Consultar Elemento - Pilha Dinâmica

- Em uma pilha a consulta se dá apenas ao elemento que está no seu topo ou início;
- Consultar o topo - **TOP**;
- **Atenção:** a consulta não remove e também não insere elementos na pilha;

```
31 int consulta_topo_Pilha(Pilha* pi, struct aluno *al){
32     if(pi == NULL)
33         return 0;
34     if((*pi) == NULL)
35         return 0;
36     *al = (*pi)->dados;
37     return 1;
38 }
```

Consultar Elemento - Pilha Dinâmica



- Quais os procedimentos para imprimir todos os elementos de uma Pilha Dinâmica?

Imprimir - Pilha Dinâmica

```
88 void imprime_Pilha(Pilha* pi){
89     if(pi == NULL)
90         return;
91     Elem* no = *pi;
92     while(no != NULL){
93         printf("Matricula: %d\n", no->dados.matricula);
94         printf("Nome: %s\n", no->dados.nome);
95         printf("Notas: %f %f %f\n", no->dados.n1,
96             no->dados.n2,
97             no->dados.n3);
98         printf("-----\n");
99         no = no->prox;
100     }
101 }
```

Exemplos

- `https://portaldoprofessor.fct.unesp.br/projetos/cadilag/apps/structs/?list=card`

Referências

- André Ricardo Backes, CAPÍTULO 8 - Pilhas, Editor(s): André Ricardo Backes, **Estrutura de Dados Descomplicada em Linguagem C**, Elsevier Editora Ltda., 2016, Pages 193-220, ISBN 9788535285239.

