

# Estrutura de Dados - I

## **Pilha**

Prof. MSc. Rafael Staiger Bressan  
[rafael.bressan@unicesumar.edu.br](mailto:rafael.bressan@unicesumar.edu.br)

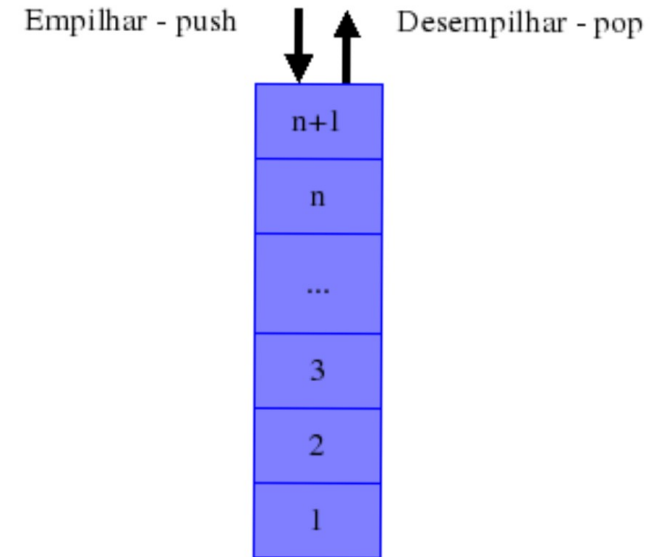


# Pilha

- O conceito de **pilha** é usado em muitos softwares de sistemas incluindo compiladores e interpretadores. (*A maioria dos compiladores C usa pilha quando passa argumentos para funções*)

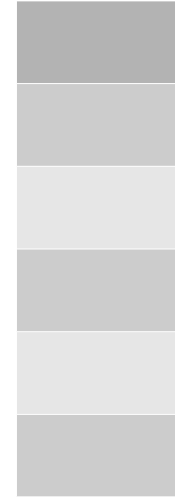
# Pilha

- Estruturas do tipo LIFO (last-in first-out)
  - último elemento a ser inserido, será o primeiro a ser retirado.
- Exemplos
  - Editores de texto:
    - desfazer/refazer
  - Compiladores;
  - Navegação entre páginas Web;
  - etc.



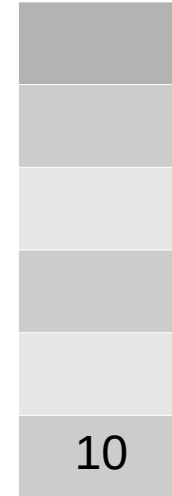
# Pilha

- Pilha Vazia



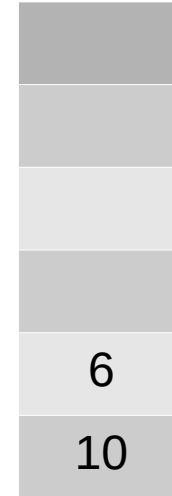
# Pilha

- Pilha Vazia
- `push(10)`



# Pilha

- Pilha Vazia
- push(10)
- push(6)



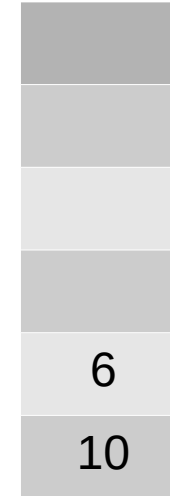
# Pilha

- Pilha Vazia
- push(10)
- push(6)
- push(8)



# Pilha

- Pilha Vazia
- push(10)
- push(6)
- push(8)
- pop()





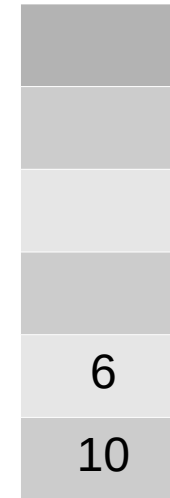
# Pilha

- Pilha Vazia
- push(10)
- push(6)
- push(8)
- pop()
- push(2)



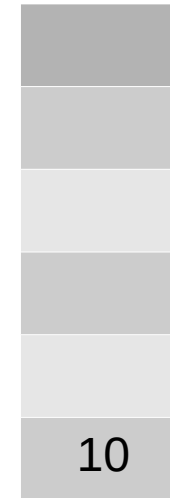
# Pilha

- Pilha Vazia
- push(10)
- push(6)
- push(8)
- pop()
- push(2)
- pop()



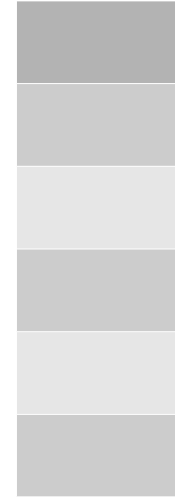
# Pilha

- Pilha Vazia
- push(10)
- push(6)
- push(8)
- pop()
- push(2)
- pop()
- pop()



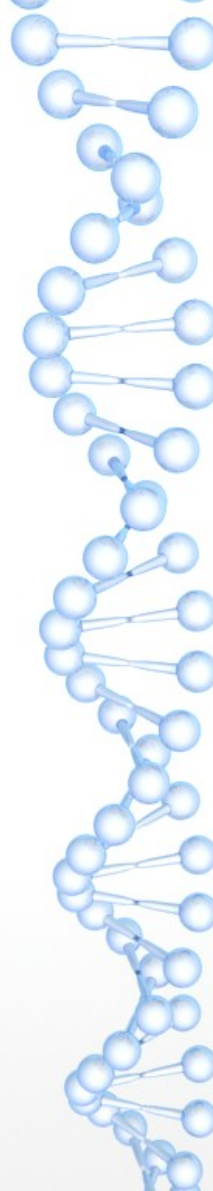
# Pilha

- Pilha Vazia
- push(10)
- push(6)
- push(8)
- pop()
- push(2)
- pop()
- pop()
- pop()
- Pilha Vazia

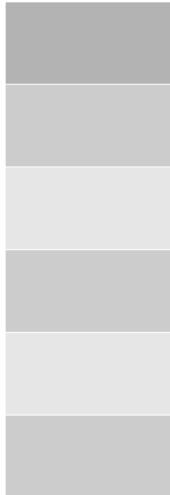


# Pilha

## Inserção (push)



```
11 void push(int conteudo, celula *ini) {  
12     if (ini->prox == NULL){  
13         celula *nova; nova = malloc(sizeof(celula));  
14         nova->conteudo = conteudo;  
15         nova->prox = NULL;  
16         ini->prox = nova;  
17     }else{  
18         push(conteudo, ini->prox);  
19     }  
20 }
```



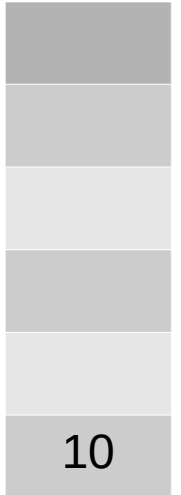
# Pilha

## Inserção (push)



push(10, ini);

```
11 void push(int conteudo, celula *ini) {  
12     if (ini->prox == NULL){  
13         celula *nova; nova = malloc(sizeof(celula));  
14         nova->conteudo = conteudo;  
15         nova->prox = NULL;  
16         ini->prox = nova;  
17     }else{  
18         push(conteudo, ini->prox);  
19     }  
20 }
```



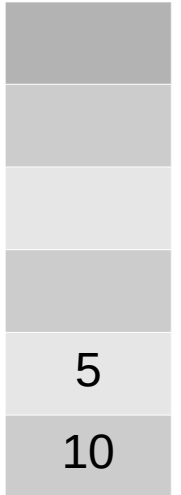
# Pilha

## Inserção (push)



push(5, ini);

```
11 void push(int conteudo, celula *ini) {  
12     if (ini->prox == NULL){  
13         celula *nova; nova = malloc(sizeof(celula));  
14         nova->conteudo = conteudo;  
15         nova->prox = NULL;  
16         ini->prox = nova;  
17     }else{  
18         push(conteudo, ini->prox);  
19     }  
20 }
```



# Pilha

## Inserção (push)



push(8, ini);

```
11 void push(int conteudo, celula *ini) {  
12     if (ini->prox == NULL){  
13         celula *nova; nova = malloc(sizeof(celula));  
14         nova->conteudo = conteudo;  
15         nova->prox = NULL;  
16         ini->prox = nova;  
17     }else{  
18         push(conteudo, ini->prox);  
19     }  
20 }
```





# Pilha

## Inserção (push)



push(16, ini);

```
11 void push(int conteudo, celula *ini) {  
12     if (ini->prox == NULL){  
13         celula *nova; nova = malloc(sizeof(celula));  
14         nova->conteudo = conteudo;  
15         nova->prox = NULL;  
16         ini->prox = nova;  
17     }else{  
18         push(conteudo, ini->prox);  
19     }  
20 }
```



# Pilha

## Remoção (pop)

```
23 void pop(celula *ini) {  
24     if (ini->prox == NULL){  
25         printf("Lista Vazia!\n");  
26     }else if(ini->prox->prox == NULL){  
27         celula *lixo;  
28         lixo = ini->prox;  
29         ini->prox = NULL;  
30         free(lixo);  
31     }else{  
32         pop(ini->prox);  
33     }  
34 }
```

16

8

5

10

# Pilha

## Remoção (pop)

→ pop(ini);

```
23 void pop(celula *ini) {  
24     if (ini->prox == NULL){  
25         printf("Lista Vazia!\n");  
26     }else if (ini->prox->prox == NULL){  
27         celula *lixo;  
28         lixo = ini->prox;  
29         ini->prox = NULL;  
30         free(lixo);  
31     }else{  
32         pop(ini->prox);  
33     }  
34 }
```

8

5

10

# Pilha

## Remoção (pop)

→ pop(ini);

```
23 void pop(celula *ini) {  
24     if (ini->prox == NULL){  
25         printf("Lista Vazia!\n");  
26     }else if(ini->prox->prox == NULL){  
27         celula *lixo;  
28         lixo = ini->prox;  
29         ini->prox = NULL;  
30         free(lixo);  
31     }else{  
32         pop(ini->prox);  
33     }  
34 }
```

5

10

# Pilha

## Remoção (pop)

→ pop(ini);

```
23 void pop(celula *ini) {  
24     if (ini->prox == NULL){  
25         printf("Lista Vazia!\n");  
26     }else if(ini->prox->prox == NULL){  
27         celula *lixo;  
28         lixo = ini->prox;  
29         ini->prox = NULL;  
30         free(lixo);  
31     }else{  
32         pop(ini->prox);  
33     }  
34 }
```

10

# Pilha

## Remoção (pop)

→ pop(ini);

```
23 void pop(celula *ini) {  
24     if (ini->prox == NULL){  
25         printf("Lista Vazia!\n");  
26     }else if(ini->prox->prox == NULL){  
27         celula *lixo;  
28         lixo = ini->prox;  
29         ini->prox = NULL;  
30         free(lixo);  
31     }else{  
32         pop(ini->prox);  
33     }  
34 }
```



# Atividade

- Desenvolva um programa que contenha as funções `push()` e `pop()` apresentadas no slide para manipulação de uma pilha.
- Seu programa deve conter as opções:
  - Push
  - Pop
  - Pilha vazia
  - Exibir dados