ESTRUTURA DE DADOS

Prof.ª Priscilla Abreu

priscilla.braz@rj.senac.br

Estrutura de dados

Roteiro de Aula

- Objetivo da aula
- Listas
  - Pilha sequencial
  - Fila sequencial

Estrutura de dados

Objetivo da aula

Manipular listas lineares com restrições de acesso.

Competência:
Desenvolver estruturas de dados lineares e não lineares.

REVISANDO...

Estrutura de dados

LISTA LINEAR

Listas lineares

Listas lineares gerais
SEM restrição de inserção e remoção de elementos

Listas particulares
COM restrição de inserção e remoção de elementos

Estrutura de dados

LISTA LINEAR

Casos particulares:

- Deque

  Inserção e remoção apenas nas extremidades;

- Pilha

  Inserção e remoção apenas em um extremo

- Fila

  Inserção em um extremo e remoção em outro extremo;

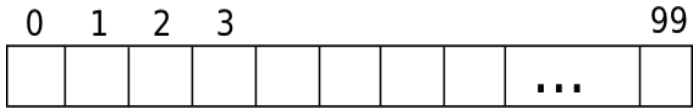
Estrutura de dados

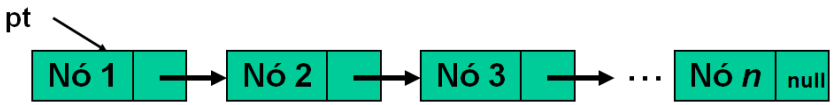
LISTA LINEAR: TIPO DE ARMAZENAMENTO

O tipo de armazenamento de uma lista linear pode ser classificado de acordo com a posição relativa na memória (contígua ou não) de cada dois nós consecutivos na lista.

Existem dois tipos de listas:

- Lista sequencial
- Lista encadeada





PILHAS SEQUENCIAIS

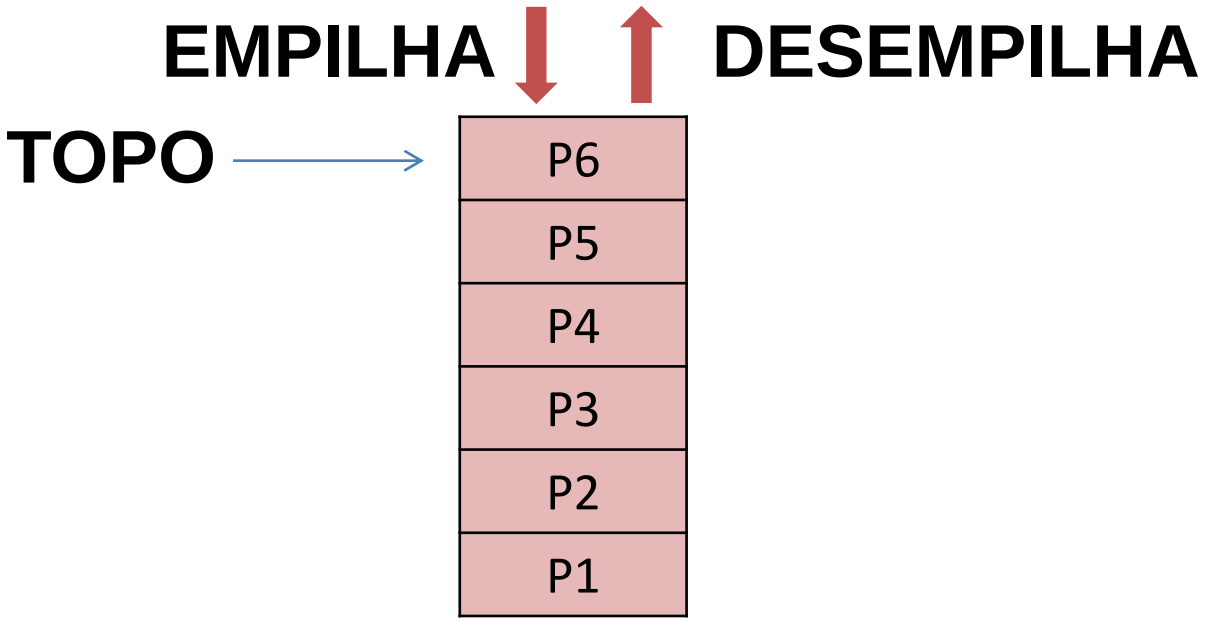
Estrutura de dados

PILHAS

- Estruturas de dados do tipo LIFO (last-in first-out): o último elemento a ser inserido, será o primeiro a ser removido.
- Manipulação dos elementos em apenas uma das extremidades: **topo**.
- Exemplos: pilha de pratos, pilha de livros, pilha de cartas de um baralho, etc.
  - Inserção: Empilha() ou Push()
  - Remoção: Desempilha() ou Pop()

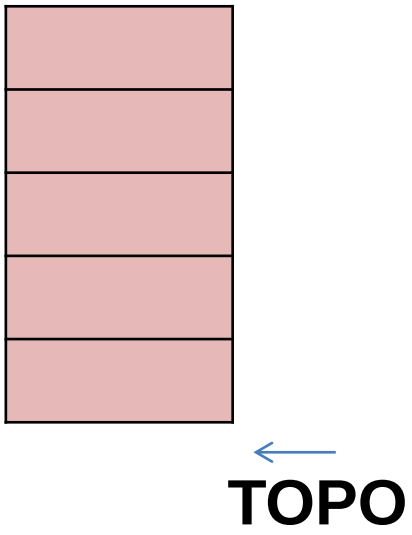
Estrutura de dados

PILHAS



Estrutura de dados

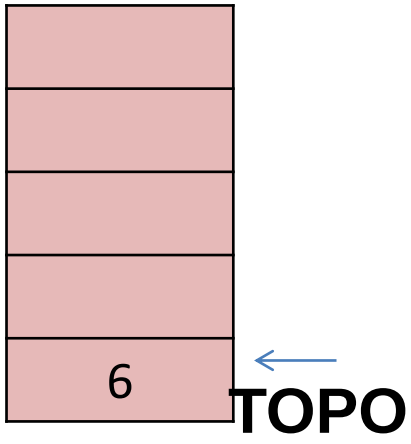
PILHA SEQUENCIAL



Estrutura de dados

PILHA SEQUENCIAL

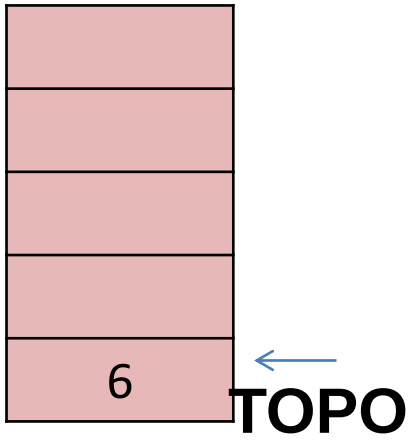
EMPILHA (6)



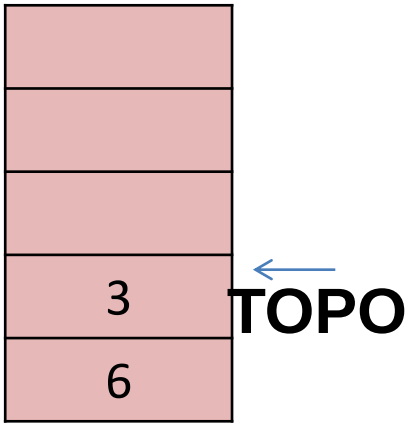
Estrutura de dados

PILHA SEQUENCIAL

EMPILHA (6)



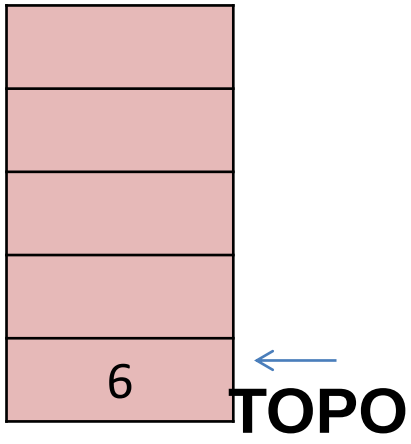
EMPILHA (3)



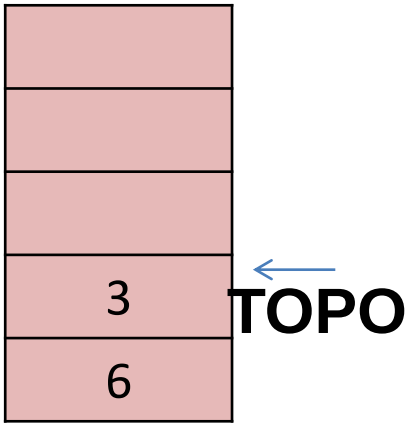
Estrutura de dados

PILHA SEQUENCIAL

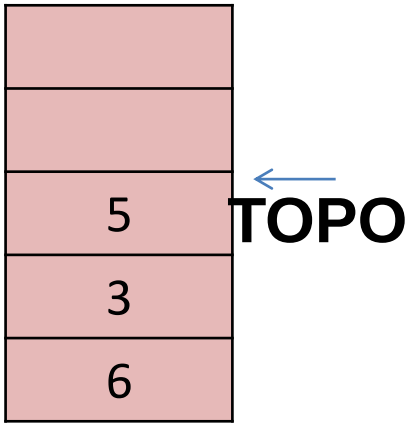
EMPILHA (6)



EMPILHA (3)



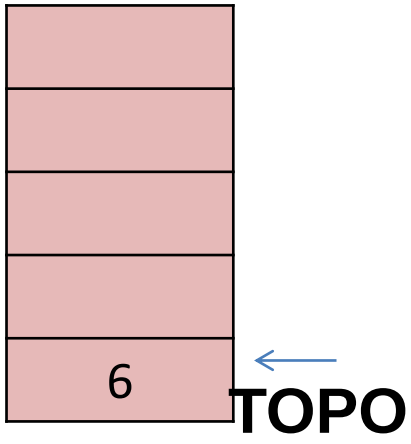
EMPILHA (5)



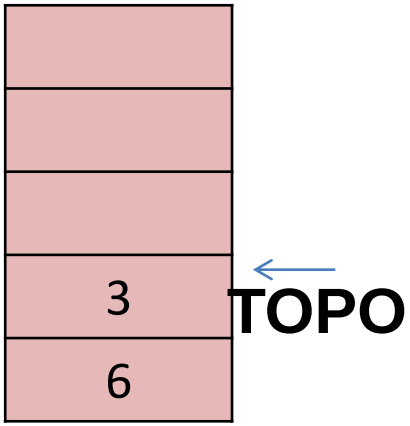
Estrutura de dados

PILHA SEQUENCIAL

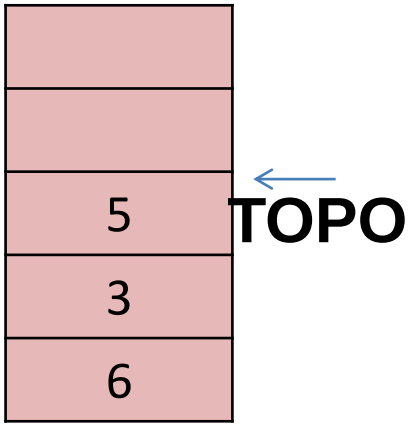
EMPILHA (6)



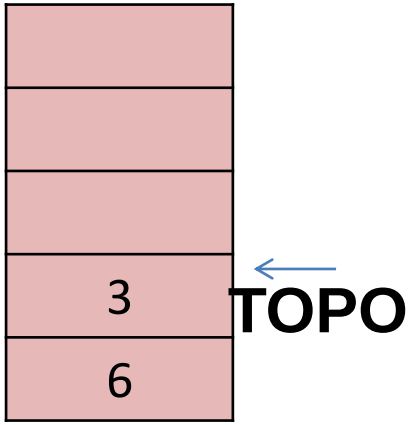
EMPILHA (3)



EMPILHA (5)



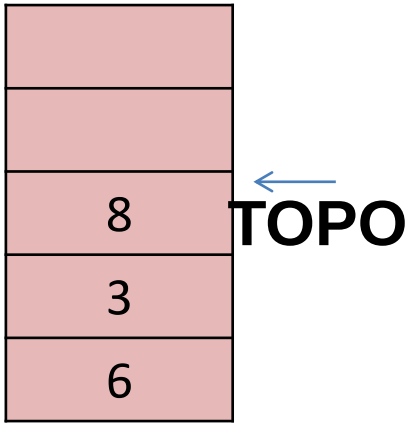
DESEMPILHA ()



Estrutura de dados

PILHA SEQUENCIAL

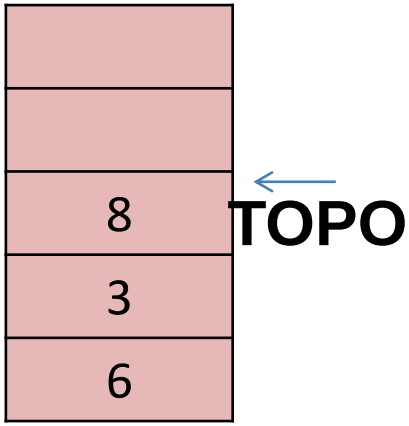
EMPILHA (8)



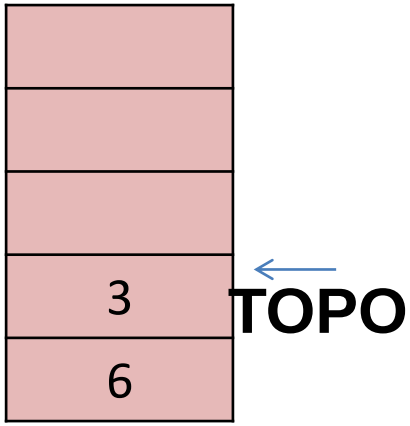
Estrutura de dados

PILHA SEQUENCIAL

EMPILHA (8)



DESEMPILHA ()



Estrutura de dados

PILHAS – APLICAÇÕES

- Uso em aplicações onde os dados são obtidos na ordem inversa àquela em que foram fornecidos.
- Exemplos:
  - Calculadora para expressões matemáticas;
  - Conversão de número decimal para binário;
  - Mecanismo de fazer/desfazer do Word;
  - Mecanismo de navegação de páginas na Internet (avançar e retornar).

Estrutura de dados

PILHAS – OPERAÇÕES BÁSICAS

- **Verificar se a pilha está cheia.**
- **Verificar se a pilha está vazia.**
- **Empilhamento**: consiste em inserir um valor no topo da pilha. É preciso verificar previamente se a pilha está cheia.
- **Desempilhamento**: consiste em retirar um valor do topo da pilha. É preciso verificar previamente se a pilha está vazia.
- **Mostrar o topo**

Estrutura de dados

PILHAS – OPERAÇÕES BÁSICAS

```
#include <stdio.h>
#define tam 100
//FUNÇÕES
...
int main(){
        int topo, pilha[tam];
        topo = -1;
        ...
}
```

Estrutura de dados

PILHAS – PILHA CHEIA

```
int pilha_cheia(int topo){
        if (topo == tam-1)
                return 1;
        return 0;
}
```

Estrutura de dados

PILHAS – PILHA VAZIA

```
int pilha_vazia(int topo){
        if (topo == -1)
                return 1;
        return 0;
}
```

Estrutura de dados

PILHAS – MOSTRAR TOPO

```
void mostrar_topo(int topo, int *pilha){
        if(!pilha_vazia(topo)){
                printf("\nTopo: %d.\n",pilha[topo]);
        }
        else{
                printf("\nPilha vazia!\n");
        }
}
```

Estrutura de dados

PILHAS – EMPILHAR

- Consiste em inserir um valor no topo da pilha, caso a pilha não esteja cheia.
- Atualiza a posição do topo;
- Atribui o valor a ser inserido no novo topo.

Estrutura de dados

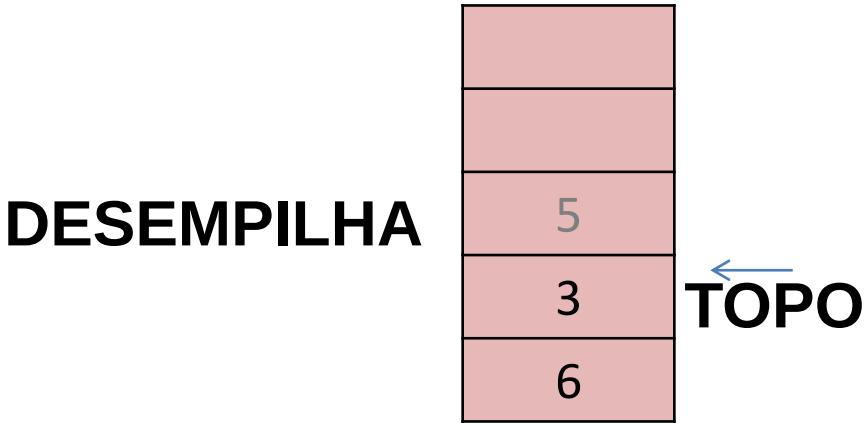
PILHAS – EMPILHAR

```
void empilhar(int *topo, int valor, int *pilha ){
        if(!pilha_cheia(*topo)){
                (*topo)++;
                pilha[*topo]=valor;
                printf("\nValor empilhado!\n");
        }
        else{
                printf("\nPilha cheia!\n");
        }
}
```

Estrutura de dados

PILHAS – DESEMPILHAR

Consiste em retirar um valor do topo da pilha e em seguida, ajustar o topo.
Só é possível se a pilha não estiver vazia.
A remoção de um elemento da pilha é realizada apenas alterando-se a informação da posição do topo.



Estrutura de dados

PILHAS – DESEMPILHAR

```
void desempilhar(int *topo, int *pilha){
        if(!pilha_vazia(*topo)){
                printf("\nValor                    %d
desempilhado!\n",pilha[*topo]);
                (*topo)--;
        }
        else{
                printf("\nPilha vazia!\n");
        }
}
```

Estrutura de dados

PILHAS – CONTINUAÇÃO

Com base nas funções relacionadas à pilha, você deve implementar um programa que implemente uma pilha, disponibilizando as seguintes opções através de um menu ao usuário:

1- Empilhar

2- Desempilhar

3- Mostrar o topo

4- Sair

Estrutura de dados

PILHAS – CONTINUAÇÃO

```
#include <stdio.h>
#define M 10
//funções
...
int main(){
        int pilha[M];
        int topo = -1;
        int valor, op;
```

Estrutura de dados

PILHAS – CONTINUAÇÃO

```
do{
        printf("\n1- Empilhar");
        printf("\n2- Desempilhar");
        printf("\n3- Mostrar topo");
        printf("\n4- Sair");
        printf("\nInforme sua opção: ");
        scanf("%d",&op);
```

Estrutura de dados

PILHAS – CONTINUAÇÃO

```
switch(op){
        case 1:{
                printf("Informe um valor: ");
                scanf("%d",&valor);
                empilhar(&topo,valor,pilha);
                break;
        }
        case 2:{
                desempilhar(&topo,pilha);
                break;
        }
```

Estrutura de dados

PILHAS – CONTINUAÇÃO

```
case 3:{
        mostrar_topo(topo,pilha);
        break;
}
case 4:{
        printf("\nSaindo...\n");
        break;
}
```

Estrutura de dados

PILHAS – CONTINUAÇÃO

```
                default:{
                        printf("\nOpção inválida!\n");
                        break;
                }
        }

    }while(op!=4);
}
```