
Tecnologia em Análise e Desenvolvimento de Sistemas

# 6 Algoritmos e Estruturas de Dados

## Estruturas Ligadas

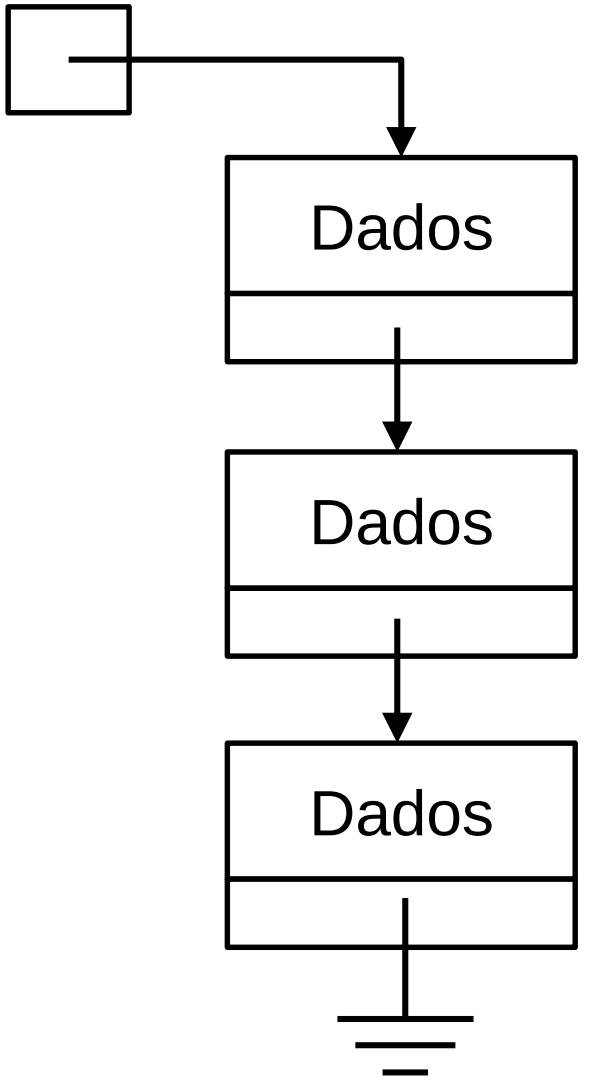
Prof. Ramide Dantas

# TADs usando Ponteiros

Estrutura auxiliar Nó/Node:

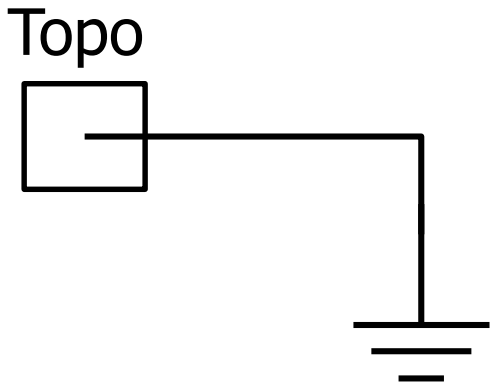
- Dados
- Ponteiro para próximo nó

```
typedef struct Node {
        Data dados;
        Node * prox;
};
```



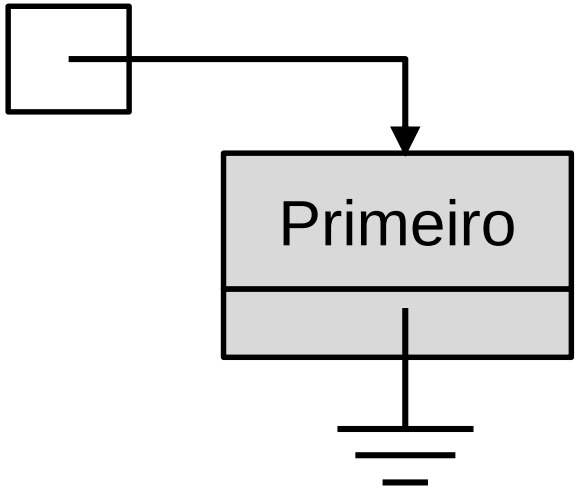
# Pilha usando Ponteiros

```
Stack stack;
```



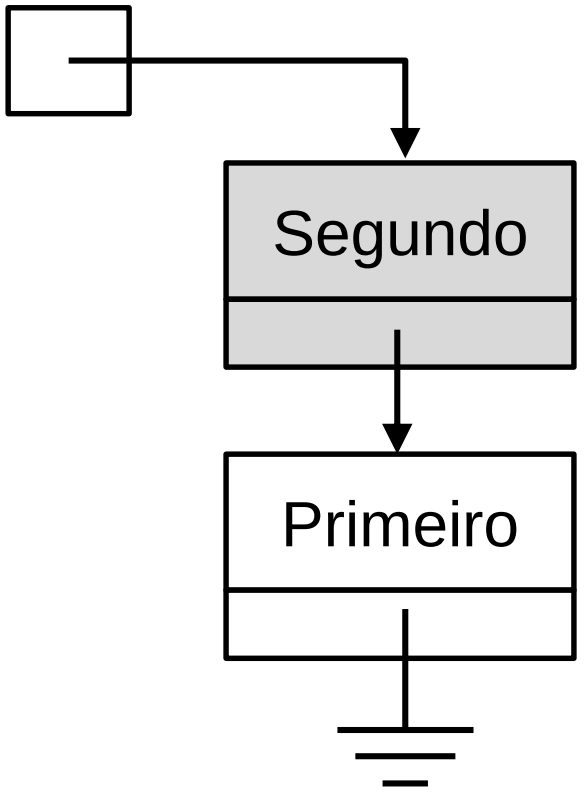

# Pilha usando Ponteiros

```
Stack stack;

stack.push("Primeiro");
```



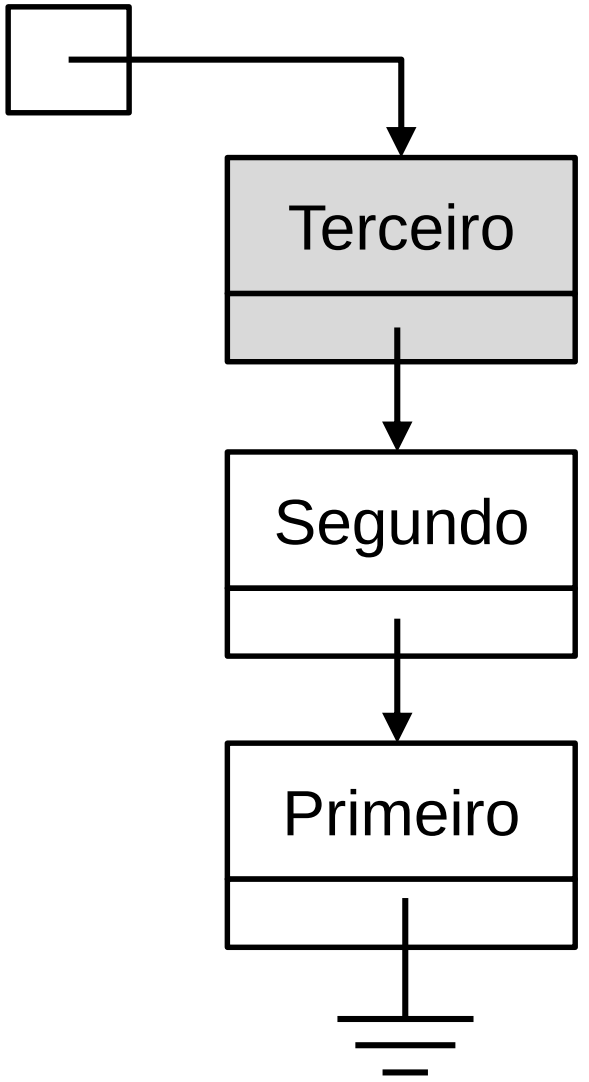

# Pilha usando Ponteiros

```
Stack stack;

stack.push(“Primeiro”);

stack.push(“Segundo”);
```



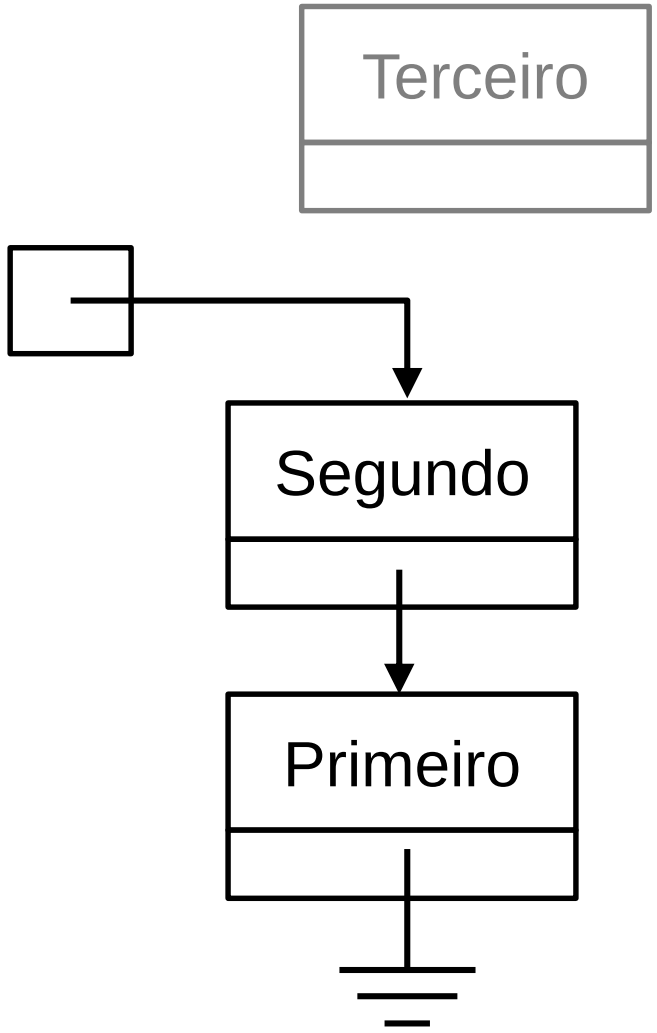

# Pilha usando Ponteiros

```
Stack stack;

stack.push("Primeiro");

stack.push("Segundo");

stack.push("Terceiro");
```



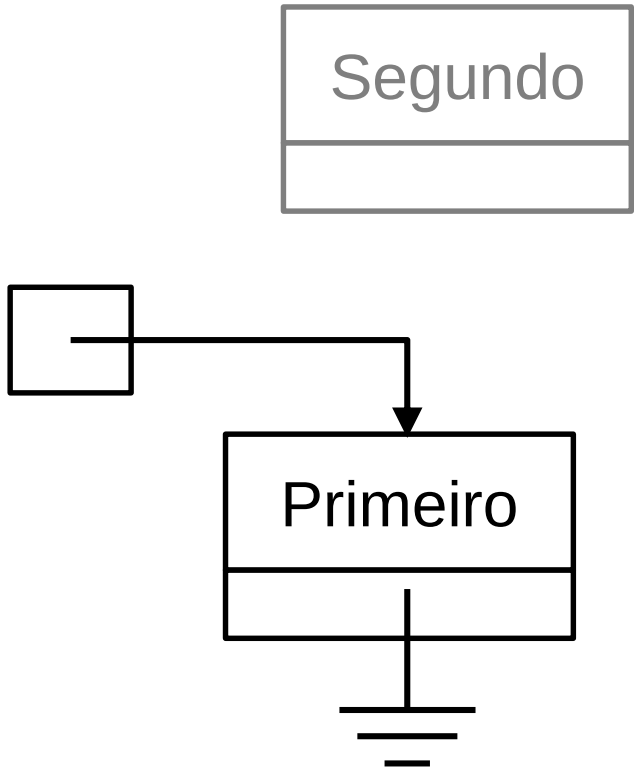
# Pilha usando Ponteiros

```
Stack stack;

stack.push("Primeiro");

stack.push("Segundo");

stack.push("Terceiro");

stack.pop();
```



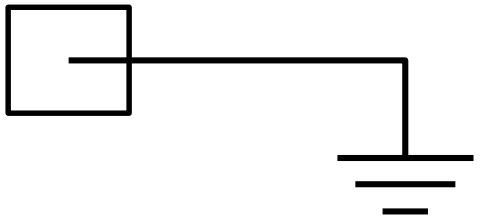

# Pilha usando Ponteiros

```
Stack stack;

stack.push(“Primeiro”);

stack.push(“Segundo”);

stack.push(“Terceiro”);

stack.pop();

stack.pop();
```



# Pilha usando Ponteiros

```
Stack stack;

stack.push(“Primeiro”);

stack.push(“Segundo”);

stack.push(“Terceiro”);

stack.pop();

stack.pop();

stack.pop();
```

# Pilha usando Ponteiros

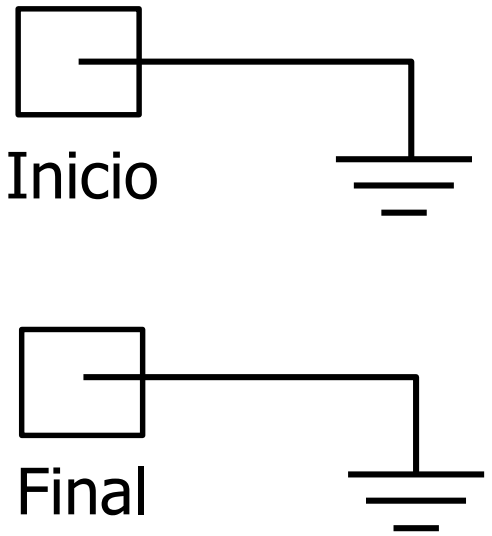
```
void Stack::push(Data dados) {
        // alocar novo NÓ;
        //    se falhar -> "stack overflow"
        // colocar dados em NÓ->dados
        // fazer NÓ->prox apontar para TOPO
        // fazer TOPO apontar para NÓ
}
```

# Pilha usando Ponteiros

```
Data Stack::pop() {
      // verificar se o TOPO é NULO;
      //    se for -> "stack underflow"
      // pegar TOPO e salvar em variável TMP
      // fazer TOPO igual a TMP->prox
      // desalocar TMP
      // retorna dados
}
```

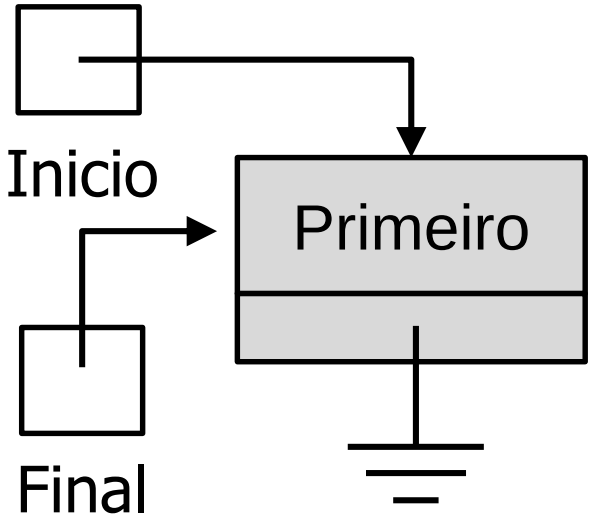
# Fila usando Ponteiros

```
Queue queue;
```



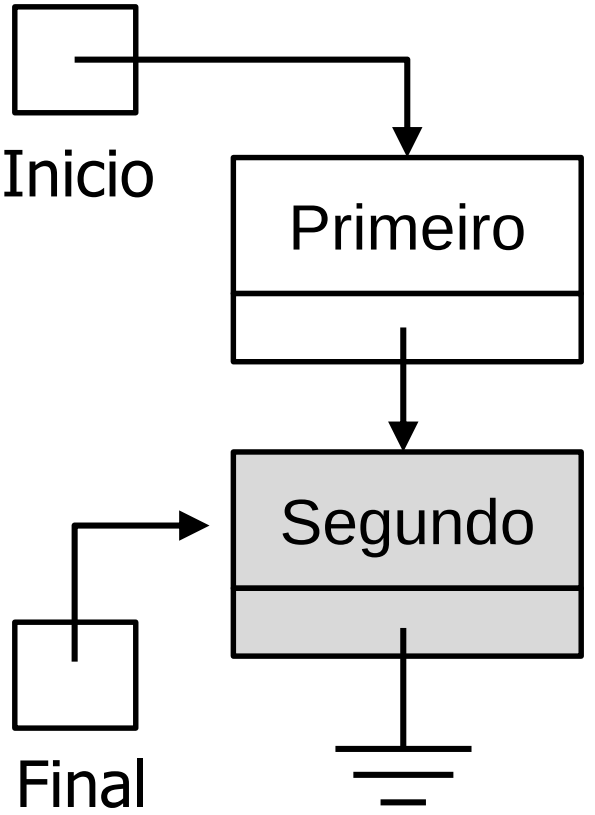
# Fila usando Ponteiros

```
Queue queue;

queue.enqueue("Primeiro");
```



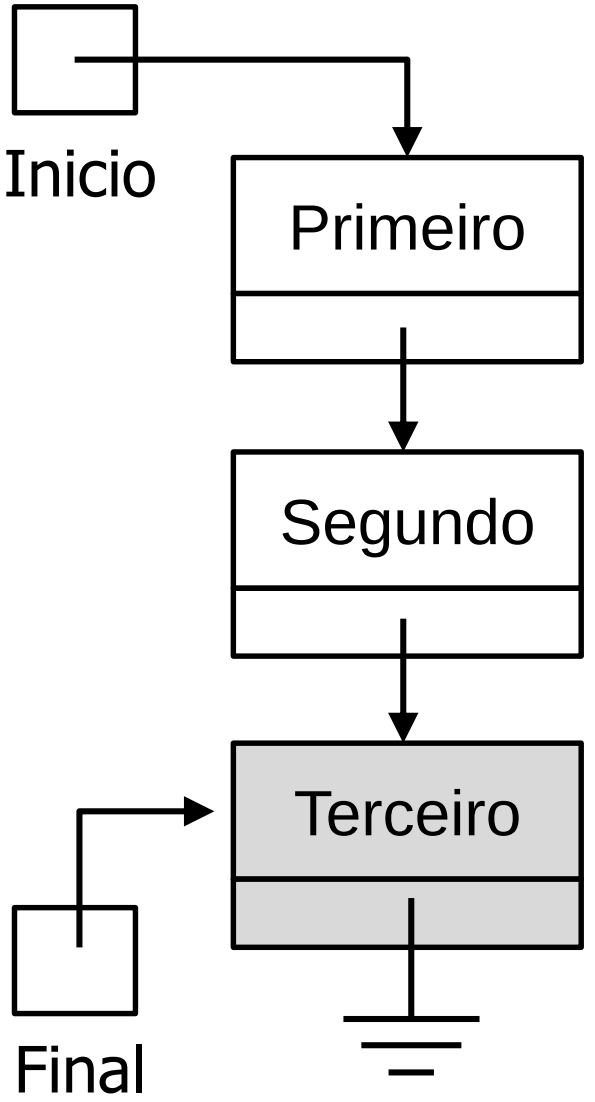
# Fila usando Ponteiros

```
Queue queue;

queue.enqueue("Primeiro");

queue.enqueue("Segundo");
```



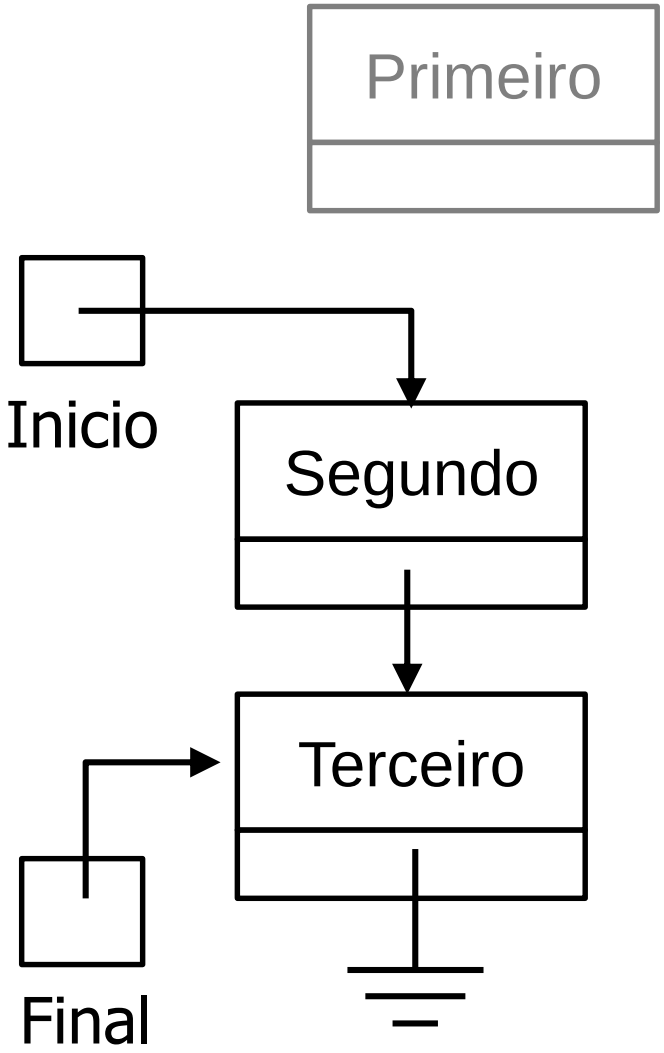
# Fila usando Ponteiros

```
Queue queue;

queue.enqueue("Primeiro");

queue.enqueue("Segundo");

queue.enqueue("Terceiro");
```



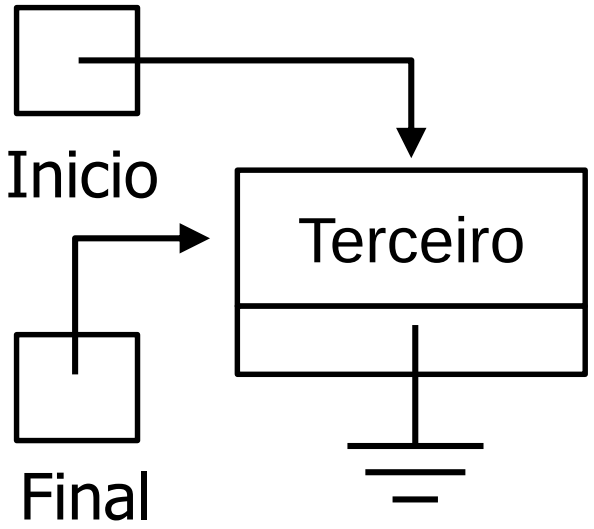
# Fila usando Ponteiros

```
Queue queue;

queue.enqueue("Primeiro");

queue.enqueue("Segundo");

queue.enqueue("Terceiro");

queue.dequeue();
```



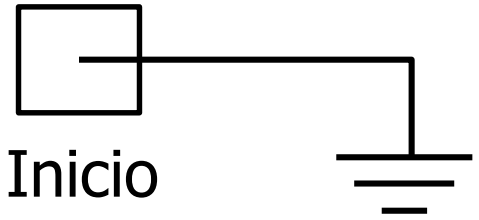
# Fila usando Ponteiros

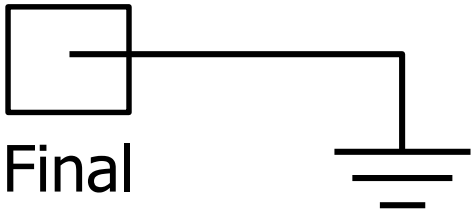
```
Queue queue;

queue.enqueue(“Primeiro”);

queue.enqueue(“Segundo”);

queue.enqueue(“Terceiro”);

queue.dequeue();

queue.dequeue();
```


# Fila usando Ponteiros

```
Queue queue;

queue.enqueue(“Primeiro”);

queue.enqueue(“Segundo”);

queue.enqueue(“Terceiro”);

queue.dequeue();

queue.dequeue();

queue.dequeue();
```


# Fila usando Ponteiros

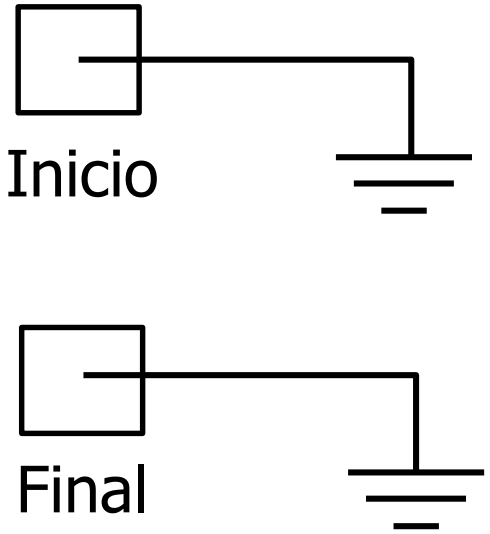
```
void Queue::enqueue(Data dados) {
      // alocar novo NÓ;
      //    se falhar -> "queue overflow"
      // colocar dados em NÓ
      // fazer NÓ->prox apontar para a NULO
      // SE (INICIO é NULO) ENTÃO -- Fila vazia
      //    fazer INICIO apontar para NÓ
      // SENÃO
      //    fazer FINAL->prox apontar para NÓ
      // FIM-SE
      // fazer FINAL apontar para NÓ
}
```

# Fila usando Ponteiros

```
Data Queue::dequeue() {
      // verificar se o INICIO é NULO;
      //    se for -> "queue underflow"
      // pegar INICIO e salvar em TMP
      // fazer INICIO apontar para INICIO->prox
      // SE (INICIO é NULO) ENTÃO -- Fila vazia
      //    fazer FINAL apontar para NULO
      // FIM-SE
      // desaloca TMP (salvar dados antes)
      // retornar dados
}
```

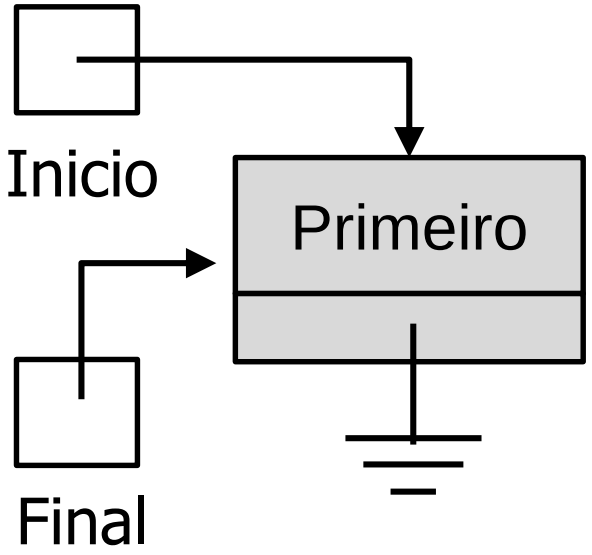
# Lista usando Ponteiros

```
List list;
```



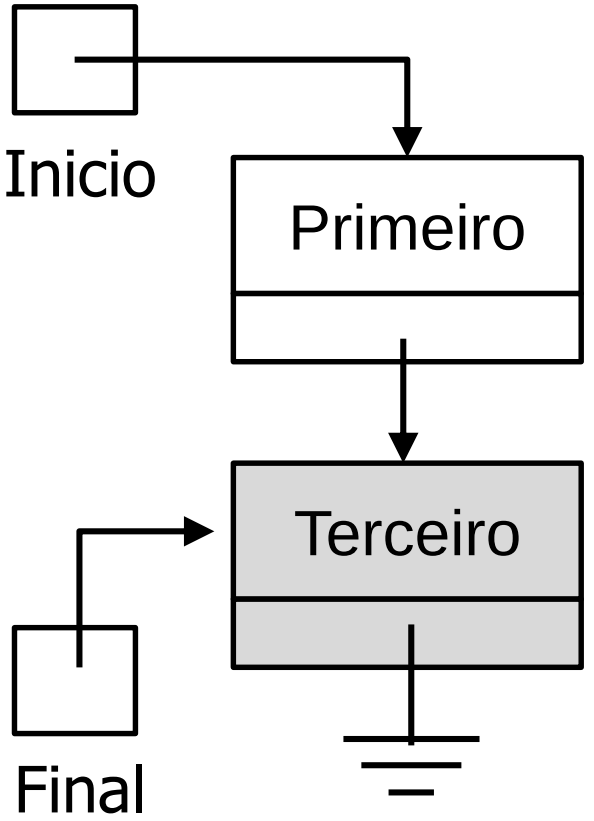
# Lista usando Ponteiros

```
List list;

list.add("Primeiro");
```



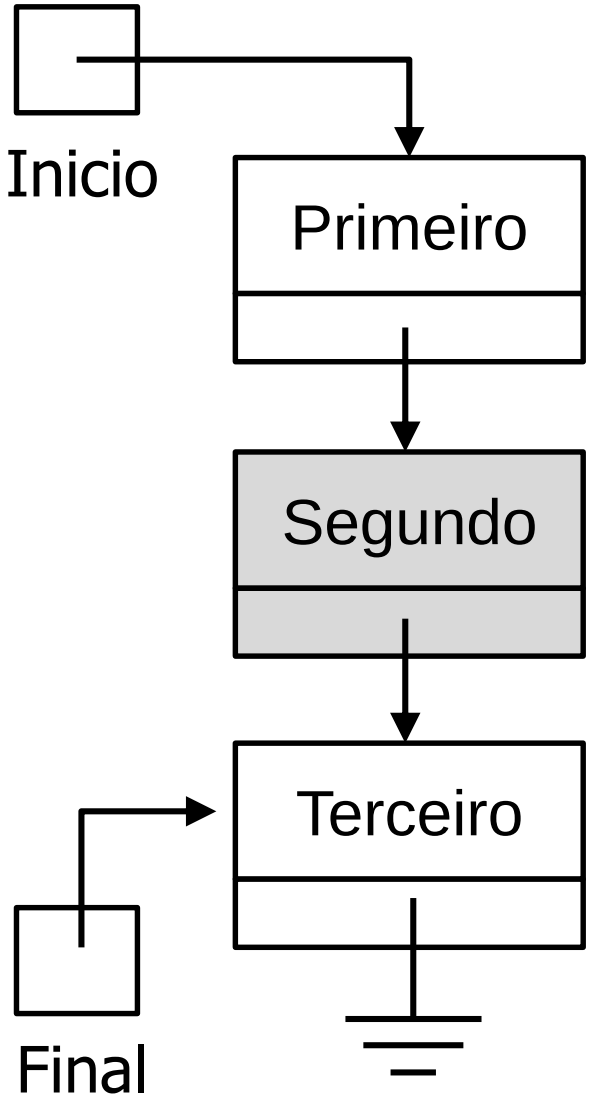

# Lista usando Ponteiros

```
List list;

list.add("Primeiro");

list.add("Terceiro");
```



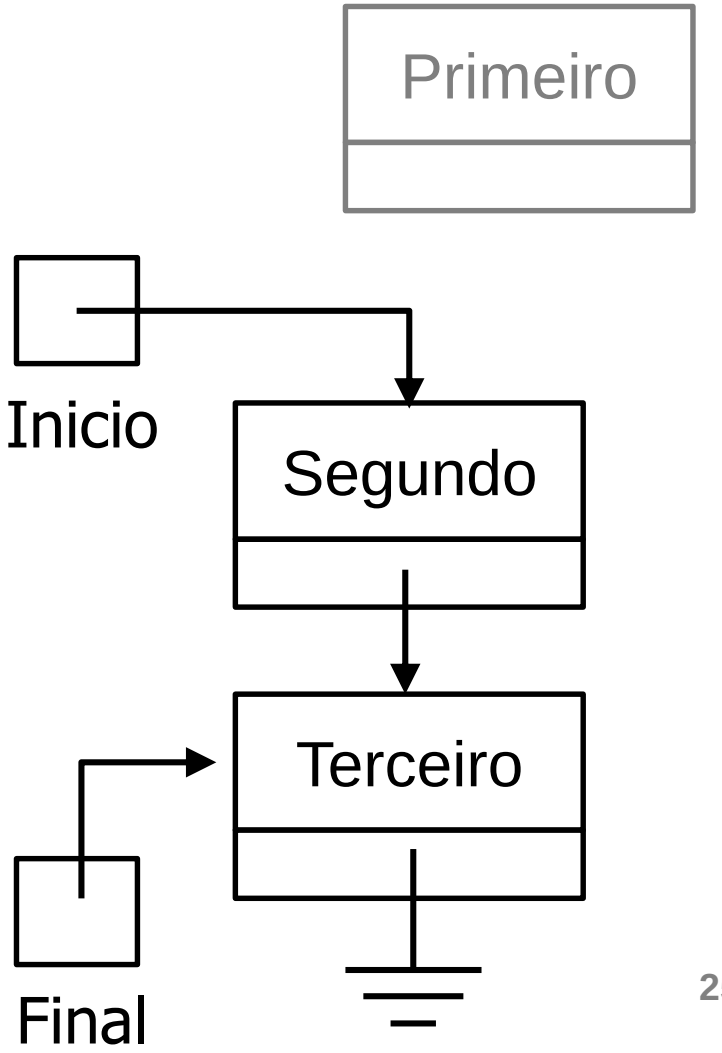
# Lista usando Ponteiros

```
List list;

list.add("Primeiro");

list.add("Terceiro");

list.insert(2, "Segundo");
```



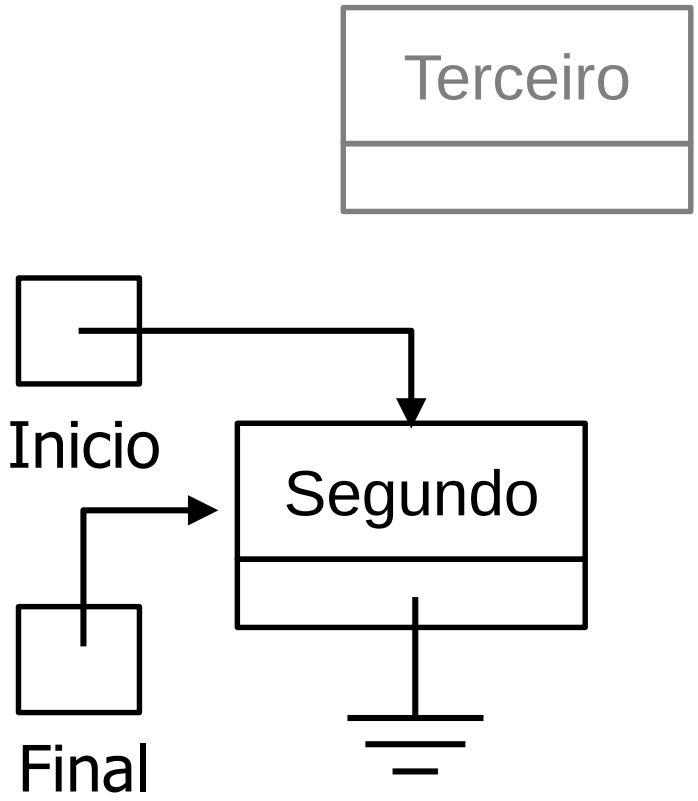
# Lista usando Ponteiros

```
List list;

list.add("Primeiro");

list.add("Terceiro");

list.insert(2, "Segundo");

list.remove(1);
```



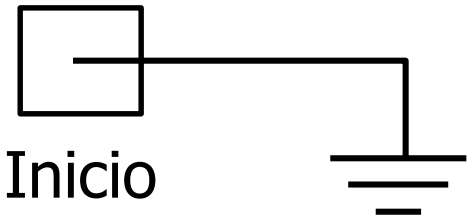
# Lista usando Ponteiros

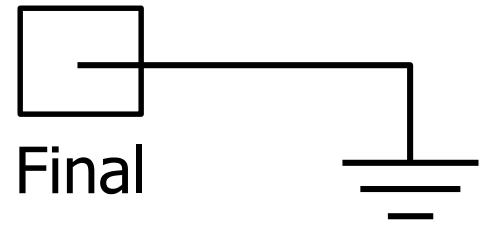
```
List list;

list.add("Primeiro");

list.add("Terceiro");

list.insert(2, "Segundo");

list.remove(1);

list.remove(2);
```



# Lista usando Ponteiros

```
List list;

list.add("Primeiro");

list.add("Terceiro");

list.insert(2, "Segundo");

list.remove(1);

list.remove(2);

list.remove(1);
```

# Lista usando Ponteiros

```
void List::insert(int idx, Data dados) {
      // alocar novo NÓ;
      //    se falhar -> "list overflow"
      // colocar dados em NÓ
      // SE (IDX é 1) ENTÃO -- Primeiro elemento
      //    fazer NÓ->prox apontar para INICIO
      //    fazer INICIO apontar para NÓ
      // SENÃO
      //    pegar nó ANTERIOR à posição IDX
      //    fazer NÓ->prox apontar p/ ANTERIOR->prox
      //    fazer ANTERIOR->prox apontar para NÓ
      // FIM-SE
}
```

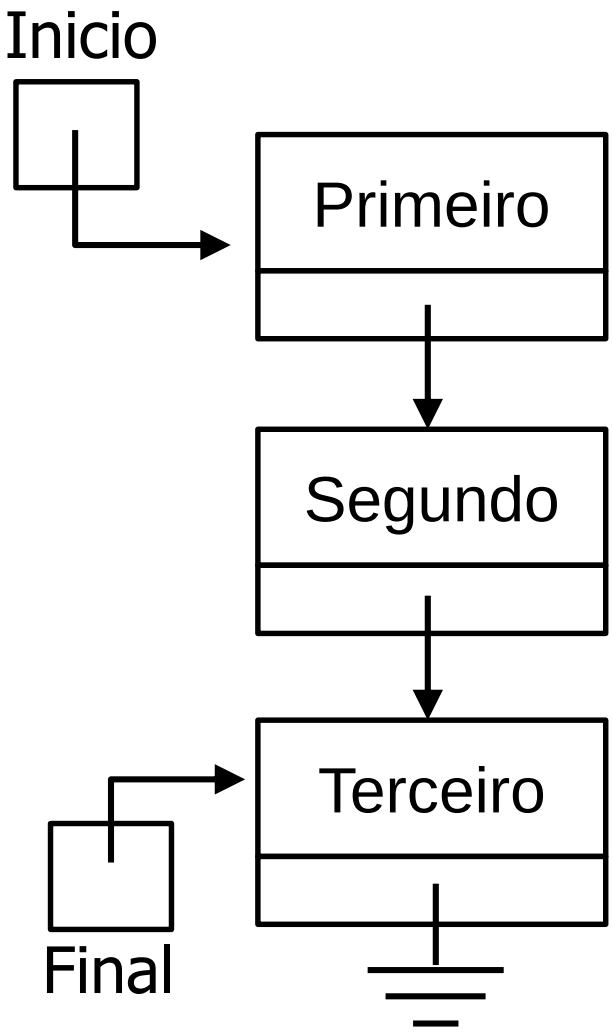
```
// remove da posição IDX e retorna dados
Data List::remove(int idx) {
      // verificar se índice é válido;
      //    do contrário -> "índice inválido"
      // SE idx == 1 ENTÃO -- Remover do começo
      //    fazer TMP apontar para INICIO
      //    fazer INICIO apontar para INICIO->prox
      //    SE (FINAL == TMP) ENTÃO -- 1° e único
      //          fazer FINAL apontar para NULL
      // SENÃO -- Remover do meio
      //    pegar nó ANTERIOR à posição IDX
      //    fazer TMP apontar para ANTERIOR->prox
      //    fazer ANTERIOR->prox apontar para
      //          ANTERIOR->prox->prox
      //    SE (FINAL == TMP) ENTÃO -- último item
      //          fazer FINAL apontar para ANTERIOR
      // FIM-SE
      // desalocar TMP (salvar DADOS antes)
      // retornar DADOS
}
```

# Lista usando Ponteiros

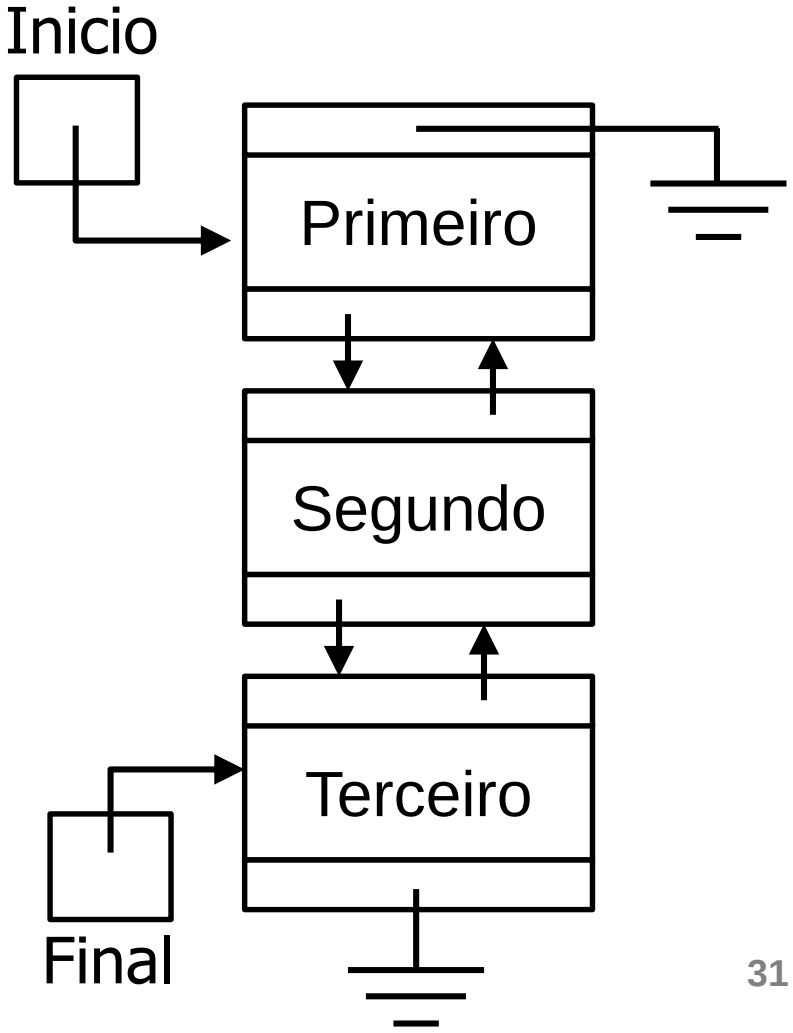
```
// retorna NÓ na posição IDX (começando de 1)
Node * List::find(int idx) {
      //    fazer NÓ apontar para INICIO
      //    decrementar IDX
      //    ENQUANTO (IDX não é zero)
      //          fazer NÓ apontar para NÓ->prox
      //          decrementar IDX
      //    retornar NÓ
}
```

# Lista usando Ponteiros

## Lista Encadeada

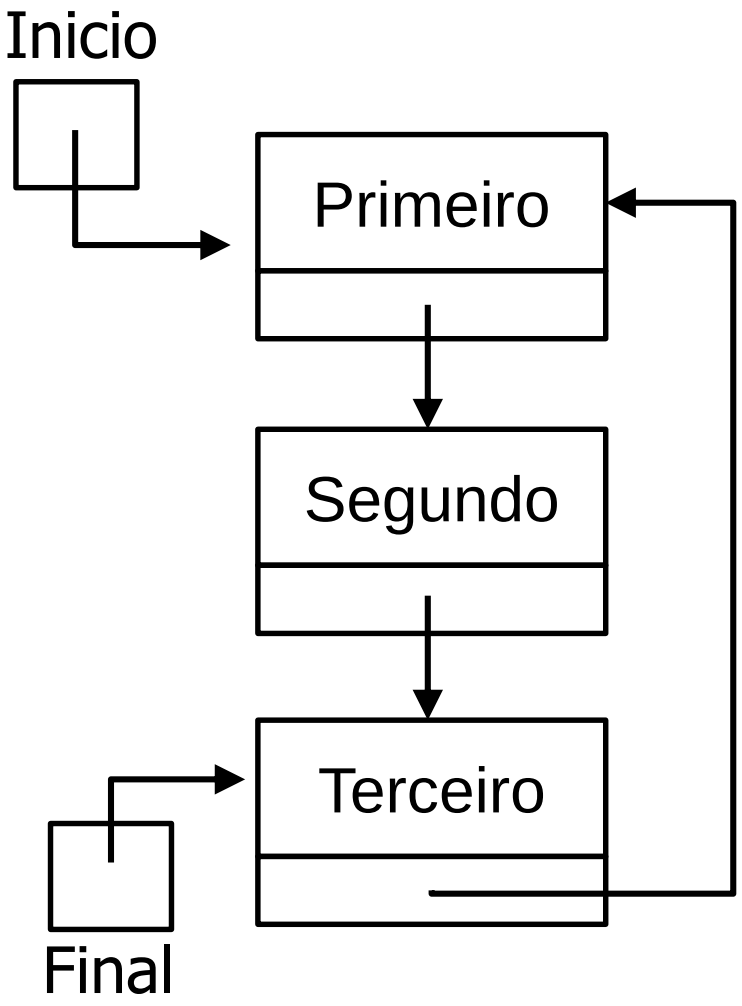



## Lista Duplamente Encadeada

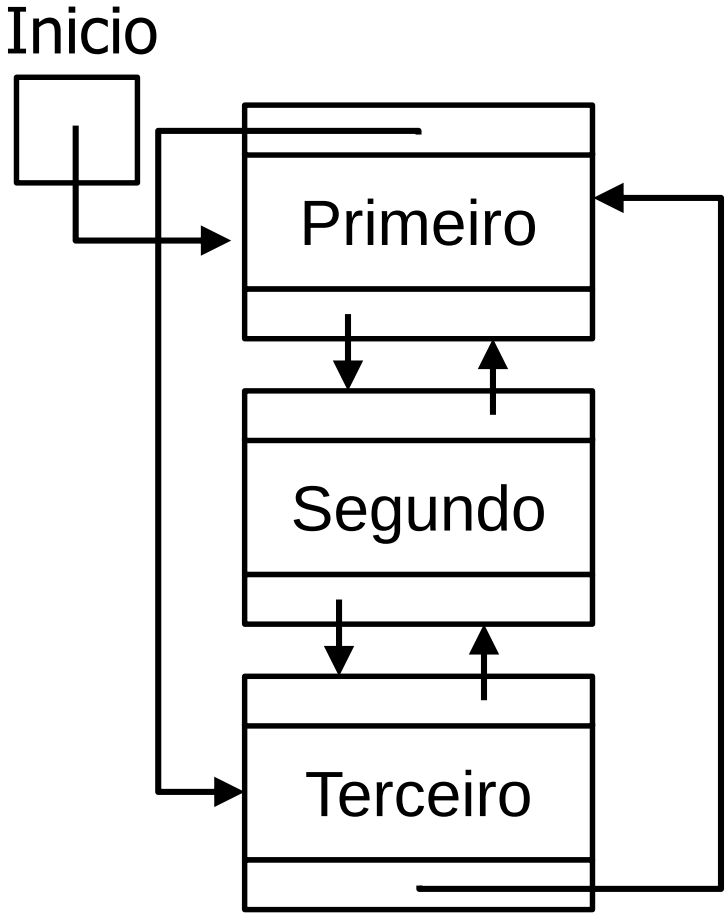


# Lista usando Pointeiros

**Lista Encadeada Circular**




**Lista Duplamente Encadeada Circular**



# TADs usando Arranjos

## Vantagens

- Quantidade de memória conhecida previamente
- Acesso aleatório através do índice
- Economia de memória (não usa ponteiros)

## Desvantagens

- Tamanho fixo (ou necessidade de realocar tudo)
- Custo de movimentação de itens (ex.: inserção ou remoção da lista)

# TADs usando Ponteiros

Estruturas **Ligadas** ou **Encadeadas**

Vantagens

- Alocação dinâmica: cresce/diminui com a demanda
- Facilidade de inserção/remoção no meio

Desvantagens

- Custo adicional dos ponteiros
- Cuidado extra com alocação/desalocação
- Tempo de acesso aleatório (ex.: item 4 da lista)


Tecnologia em Análise e Desenvolvimento de Sistemas

# 6 Algoritmos e Estruturas de Dados

## Estruturas Ligadas

Prof. Ramide Dantas