

# **Unidade VI:**

## **Tipos Abstratos de Dados Flexíveis em C**



**PUC Minas**

Instituto de Ciências Exatas e Informática  
Departamento de Ciência da Computação

- Alocação dinâmica
- Pilha flexível
- Fila flexível
- Lista simples flexível (lista simplesmente encadeada)
- Lista dupla flexível (lista duplamente encadeada)

- **Alocação dinâmica**
- Pilha flexível
- Fila flexível
- Lista simples flexível (lista simplesmente encadeada)
- Lista dupla flexível (lista duplamente encadeada)

# Introdução

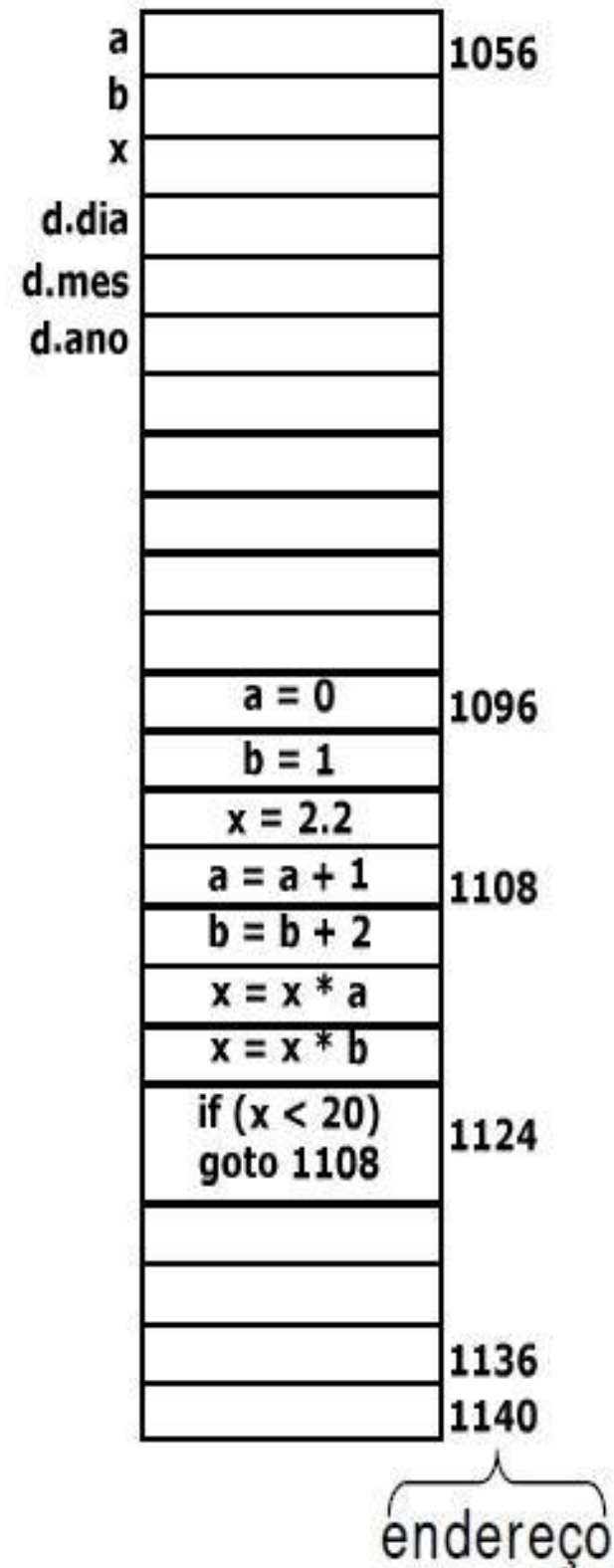
- Um programa utiliza duas áreas de memória: uma para os dados e outra para o código fonte (ou seja, as instruções)
- Uma variável possui conteúdo e endereço
  - O conteúdo de uma variável pode alterar durante a execução do programa
  - No entanto, seu endereço é constante

```

void main() { // CONTEÚDO    ENDEREÇO
    int a;           // lixo    1056
    int b;           // lixo    1060
    float x;         // lixo    1064
    Data d;          // lixo    1068

    a = 0;           // 0       1056
    b = 1;           // 1       1060
    x = 2.2f;        // 2.2    1064
    do {
        a = a + 1;
        b = b + 2;
        x = x * a;
        x = x * b;
    } while (x < 20.0f);

```



# Alocações Estática e Dinâmica

- A área para dados pode ser alocada de duas formas: Estática e Dinâmica
- Na estática, o SO reserva o espaço de memória das variáveis quando ele começa a executar um programa e essa reserva não pode ser alterada

**int a; int b[20];**

- Na dinâmica, o SO reserva esse espaço durante a execução do programa e essa reserva pode ser alterada

# Alocação Dinâmica

- A memória alocada dinamicamente está localizada em uma área chamada de *heap* e, basicamente, o programa aloca e desaloca porções de memória do *heap* durante a execução
- Acessamos as posições de memória alocadas dinamicamente através de apontadores ou ponteiros

# Ponteiros

- Os ponteiros são apenas variáveis que armazenam o endereço de uma área de memória
- Da mesma forma que um int armazena inteiro e um double, número real , os ponteiros armazenam endereços de memória
- Os ponteiros possuem tipo, ou seja, temos ponteiro para endereços de memória de um int, de um float, de um char...



# Declaração de Ponteiros

**tipoPonteiro \*nomeVariável;**

- O asterisco na declaração de uma variável indica que essa não guardará um valor e sim um endereço para o tipo especificado
- Operador endereço ( **&** ) determina o endereço de uma variável
- Operador de conteúdo de um ponteiro ( **\*** ) determina o conteúdo da posição de memória endereçada pelo ponteiro

# Declaração de Ponteiros

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1		89Bh
x2		89Ch
x3		89Dh
p		89Eh

# Declaração de Ponteiros

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	?	89Bh
x2	?	89Ch
x3	?	89Dh
p	?	89Eh

# Declaração de Ponteiros

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	11	89Bh
x2	22	89Ch
x3	33	89Dh
p	?	89Eh

## Declaração de Ponteiros

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	11	89Bh
x2	22	89Ch
x3	33	89Dh
p	89Bh	89Eh

## Declaração de Ponteiros

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	11	89Bh
x2	11	89Ch
x3	33	89Dh
p	89Bh	89Eh

## Declaração de Ponteiros

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	<b>33</b>	89Bh
x2	11	89Ch
x3	33	89Dh
p	89Bh	89Eh

## Declaração de Ponteiros

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	33	89Bh
x2	11	89Ch
x3	33	89Dh
p	89Dh	89Eh



## Declaração de Ponteiros

```
int x1, x2, x3;  int *p;
```

```
x1 = 11;  x2 = 22;  x3 = 33;
```

```
p = &x1;
```

```
x2 = *p;
```

```
*p = x3;
```

```
p = &x3;
```

```
*p = 0;
```

```
printf("cont:%d %d %d %d", x1, x2, x3, *p);
```

```
printf("addr:%p %p %p %p", &x1, &x2, &x3, p);
```

Memória		
x1	33	89Bh
x2	11	89Ch
x3	0	89Dh
p	89Dh	89Eh

# Observações

- Os símbolos usados para notação de ponteiros em C/C++ não são tão claros como deveriam ser
- Descuidado com ponteiros  $\Rightarrow$  problemas
- **Atenção:** Sempre inicialize os ponteiros

# Observações

- $p1 = p2$ : faz com que eles apontem para o mesmo local
- $*p1 = *p2$ : Atribui o conteúdo apontado por p2 o por p1
- $p++$ ,  $p--$ ,  $p=p+5$  e  $p+=3$ : Incrementa e decrementa o valor do endereço apontado pelo ponteiro, fazendo com que o ponteiro antes  $\text{sizeof}(\text{tipoPonteiro})$  bytes na memória

# Observações

- $(*p)++$  e  $(*p) --$ : Incrementa / decrementa o conteúdo da variável apontada pelo ponteiro  $p$
- Os testes relacionais como  $>$ ,  $<$ ,  $>=$ ,  $<=$ ,  $==$  ou  $!=$  são aceitos apenas para ponteiros do mesmo tipo, contudo, eles comparam endereços

# Alocar Memória em C: malloc

- Protótipo da função malloc()

**void\*** malloc (**int** tamanho)

- O malloc aloca o número de bytes passados como parâmetro e retorna um ponteiro para a primeira posição da área alocada

# Desalocar Memória em C: free()

- Protótipo da função free()

**void free (void\*)**

- O free desaloca o espaço de memória apontado pelo ponteiro recebido como parâmetro

# Exemplo do malloc() e do free()

```
char* p1 = (char*) malloc (sizeof(char));
```

```
int* p2 = (int*) malloc (sizeof(int));
```

```
float* p3 = (float*) malloc (sizeof(float));
```

```
Cliente* p4 = (Cliente*) malloc (sizeof(Cliente));
```

```
int* p5 = (int*) malloc (MAXTAM * sizeof(int));
```

```
Cliente* p6 =(Cliente*) malloc (MAXTAM * sizeof(Cliente));
```

```
free(p1);
```

```
free(p2);
```

```
free(p3);
```

```
free(p4);
```

```
free(p5);
```

```
free(p6);
```

# Alocar/Desalocar Memória em C++: new e delete

```
char* p1 = new char;
```

```
int* p2 = new int;
```

```
float* p3 = new float;
```

```
Cliente* p4 = new Cliente;
```

```
int* p5 = new int[MAXTAM];
```

```
Cliente* p6 = new Cliente[MAXTAM];
```

```
delete p1;
```

```
delete p2;
```

```
delete p3;
```

```
delete p4;
```

```
delete [] p5;
```

```
delete [] p6;
```



# Erros Comuns

- Esquecer de alocar memória e tentar acessar o conteúdo da variável
- Copiar o valor do apontador quando deveria ser o conteúdo da variável apontada
- Esquecer de desalocar memória
  - O SO desaloca no final do programa ou da função onde a variável está declarada
- Tentar acessar o conteúdo da variável depois de desalocá-la

## Exercício

- Mostre a saída na tela

```
double a;  
double *p, *q;  
  
a = 3.14;  
printf("%f\n", a);  
p = &a;  
*p = 2.718;  
printf("%f\n", a);  
a = 5;  
printf("%f\n", *p);  
p = NULL;  
p = (double *)malloc(sizeof(double));  
*p = 20;  
q = p;  
printf("%f\n", *p);  
printf("%f\n", a);  
free(p);  
printf("%f\n", *q);
```

- `int *a` não é a declaração de um vetor de `int`?
  - Em C, todo vetor é um apontador, portanto pode-se fazer coisas como:

```
int a[10], *b;  
b = a;  
b[5] = 100;  
Printf("%d\n", a[5]);  
Printf("%d\n", b[5]);
```

100
100

```
int a[10], *b;  
b = (int *) malloc(10*sizeof(int));  
b[5] = 100;  
printf("%d\n", a[5]);  
Printf("%d\n", b[5]);
```

42657
100

Obs. Não se pode fazer `a = b`  
no exemplo acima

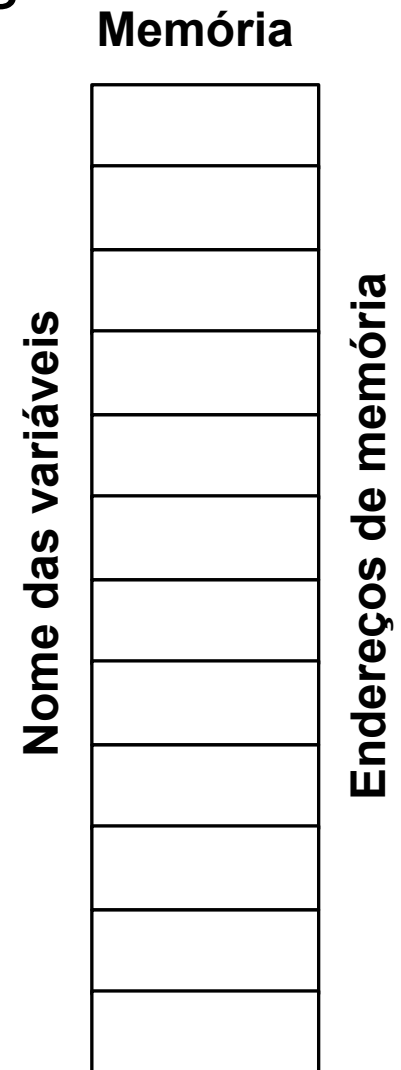
# Exercício sobre Alocação Dinâmica

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```

...
Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;
  
```

Representação gráfica



# Exercício sobre Alocação Dinâmica

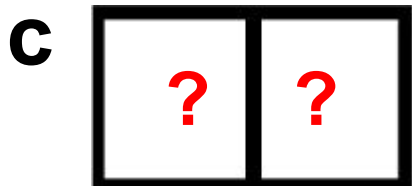
- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```

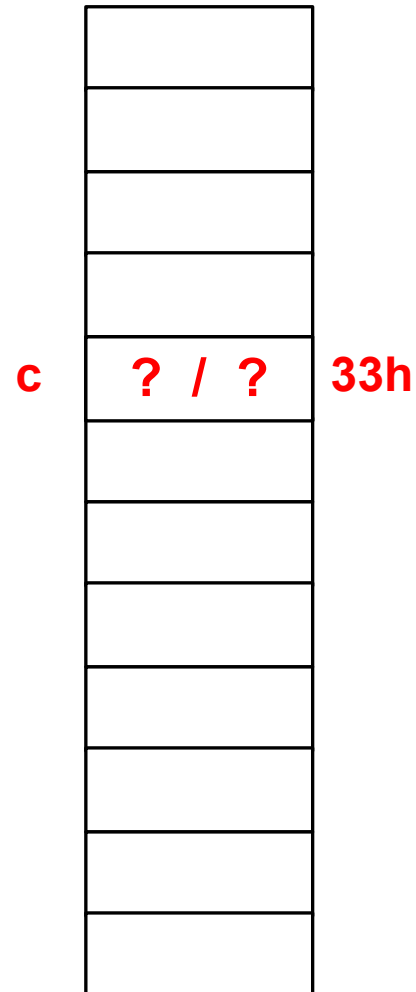
...
Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;

```

Representação gráfica



Memória



# Exercício sobre Alocação Dinâmica

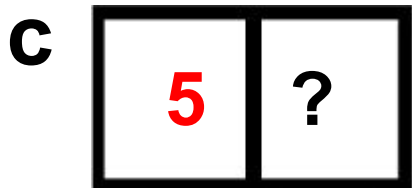
- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```

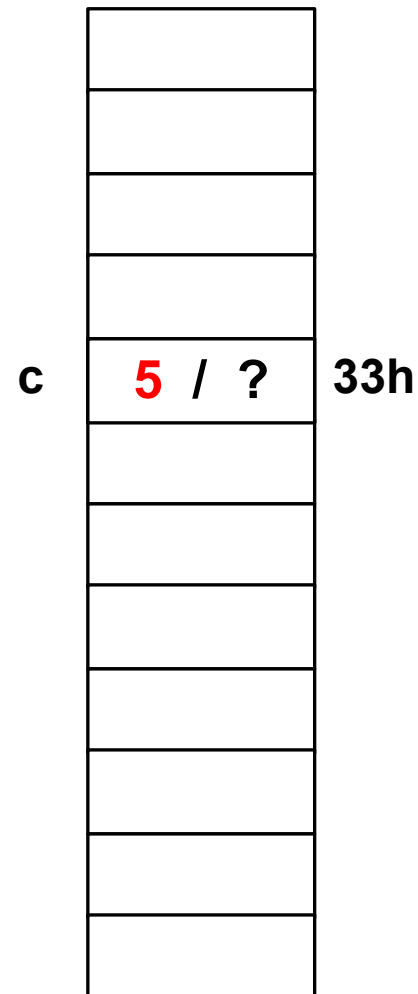
...
Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;

```

Representação gráfica



Memória



# Exercício sobre Alocação Dinâmica

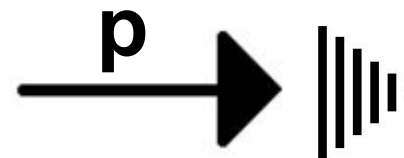
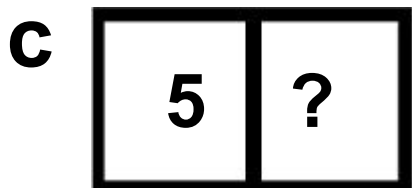
- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```

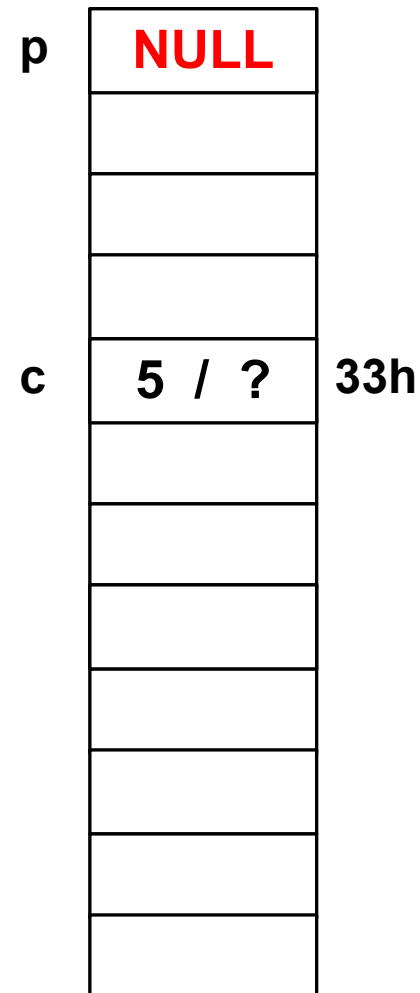
...
Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;

```

Representação gráfica



Memória



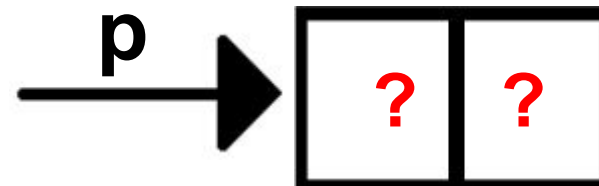
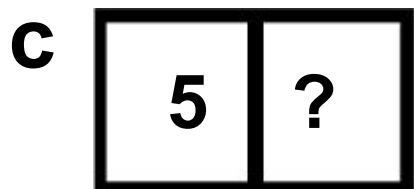
# Exercício sobre Alocação Dinâmica

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

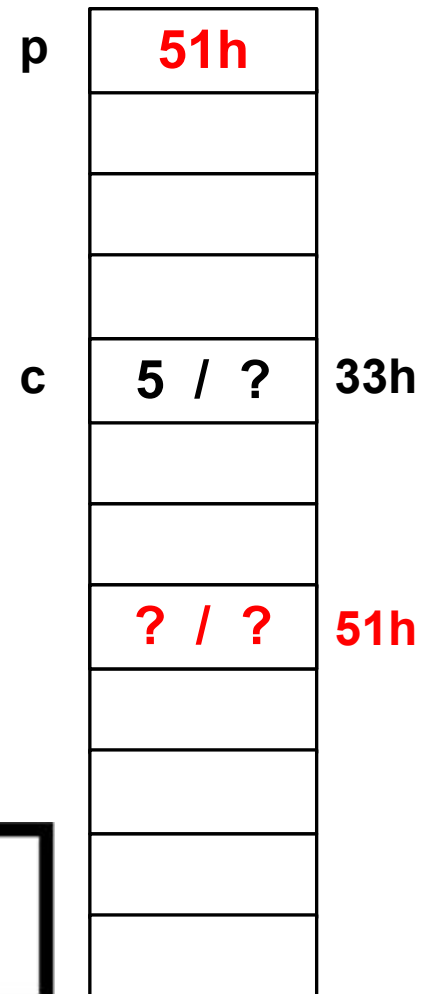
```

...
Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;
  
```

Representação gráfica



Memória





# Exercício sobre Alocação Dinâmica

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```

...
Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;

```

Memória

p	51h	33h
c	5 / ?	
	6 / ?	51h

Representação gráfica



# Exercício sobre Alocação Dinâmica

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```

...
Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;

```

Memória

p	51h	
p2	33h	
c	5 / ?	33h
	6 / ?	51h

Representação gráfica



# Exercício sobre Alocação Dinâmica

- Execute o programa abaixo, supondo os atributos código (int) e salário (double) para cada Cliente

```

...
Cliente c;
c.codigo = 5;
Cliente *p = NULL;
p = (Cliente*) malloc (sizeof(Cliente));
p->codigo = 6;
Cliente *p2 = &c;
p2->codigo = 7;

```

Memória

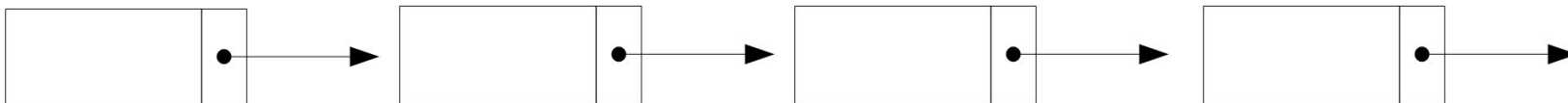
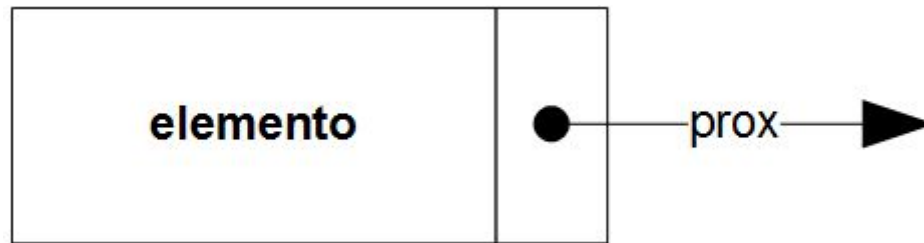
p	51h	
p2	33h	
c	7 / ?	33h
	6 / ?	51h

Representação gráfica



# Exercício sobre Alocação Dinâmica

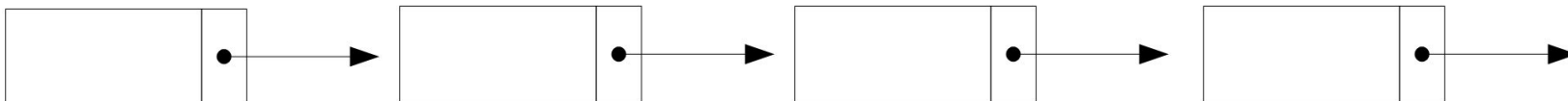
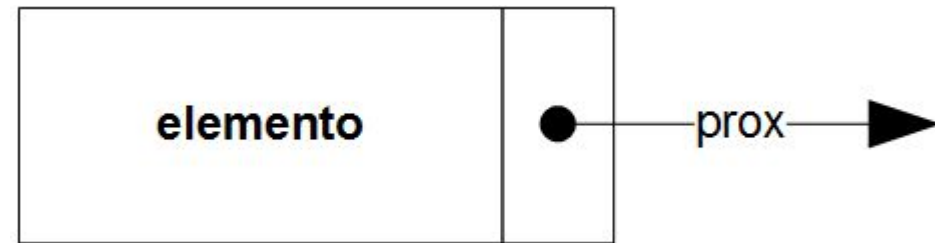
- Crie um registro célula contendo os atributos elemento (inteiro) e prox (apontador para outra célula)



# Exercício sobre Alocação Dinâmica

- Crie um registro célula contendo os atributos elemento (inteiro) e prox (apontador para outra célula)

```
typedef struct Celula {  
    int elemento;  
    struct Celula *prox;  
} Celula;  
  
Celula *novaCelula(int elemento) {  
    Celula *nova = (Celula*) malloc(sizeof(Celula));  
    nova->elemento = elemento;  
    nova->prox = NULL;  
    return nova;  
}
```



# Exercício sobre Alocação Dinâmica

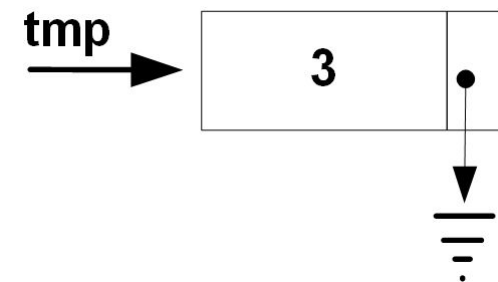
- Mostre o que acontece se um método tiver o comando *Celula*  $*tmp = novaCelula(3)$ .

```
typedef struct Celula {  
    int elemento;  
    struct Celula *prox;  
} Celula;  
  
Celula *novaCelula(int elemento) {  
    Celula *nova = (Celula*) malloc(sizeof(Celula));  
    nova->elemento = elemento;  
    nova->prox = NULL;  
    return nova;  
}
```

# Exercício sobre Alocação Dinâmica

- Mostre o que acontece se um método tiver o comando *Celula*  $*tmp = novaCelula(3)$ .

```
typedef struct Celula {  
    int elemento;  
    struct Celula *prox;  
} Celula;  
  
Celula *novaCelula(int elemento) {  
    Celula *nova = (Celula*) malloc(sizeof(Celula));  
    nova->elemento = elemento;  
    nova->prox = NULL;  
    return nova;  
}
```

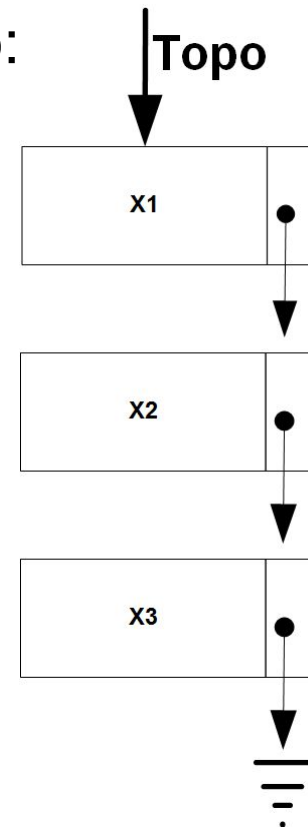


- Alocação dinâmica
- **Pilha flexível**
- Fila flexível
- Lista simples flexível (lista simplesmente encadeada)
- Lista dupla flexível (lista duplamente encadeada)



# Pilha Flexível

- Código fonte:
  - O método main é igual ao da estrutura sequencial
  - O código criará uma instância como:



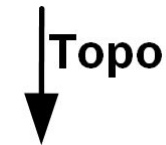
# Pilha Flexível

```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

## Pilha Flexível

```
Celula *topo;
```

```
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



## Pilha Flexível

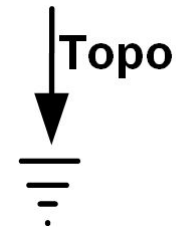
```
Celula *topo;
```

```
void start() {  
    topo = NULL;  
}
```

```
void inserir(int x) { ... }
```

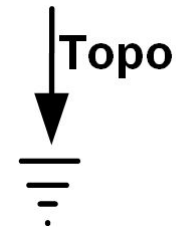
```
int remover() { ... }
```

```
void mostrar() { ... }
```



## Pilha Flexível

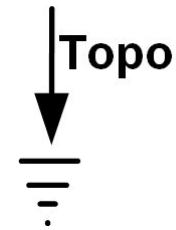
```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



```
void inserir(int x) {  
    Celula *tmp = novaCelula(x);  
    tmp->prox = topo;  
    topo = tmp;  
    tmp = NULL;  
}
```

## Pilha Flexível

```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



```
void inserir(int x) { //Inserir(3)  
    Celula *tmp = novaCelula(x);  
    tmp->prox = topo;  
    topo = tmp;  
    tmp = NULL;  
}
```

## Pilha Flexível

```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

```
void inserir(int x) { //Inserir(3)
```

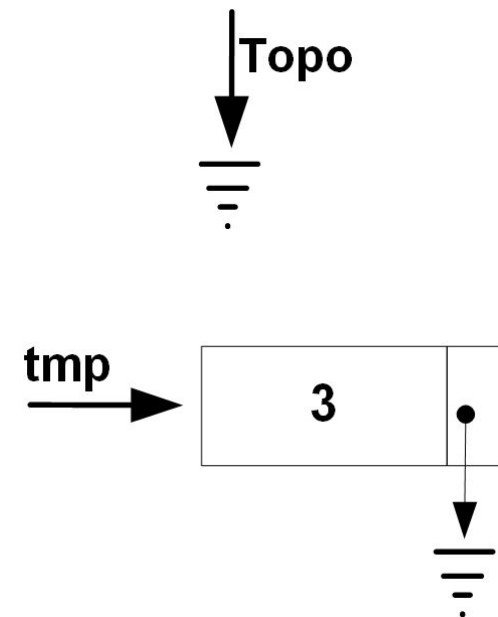
```
    Celula *tmp = novaCelula(x);
```

```
    tmp->prox = topo;
```

```
    topo = tmp;
```

```
    tmp = NULL;
```

```
}
```



## Pilha Flexível

```

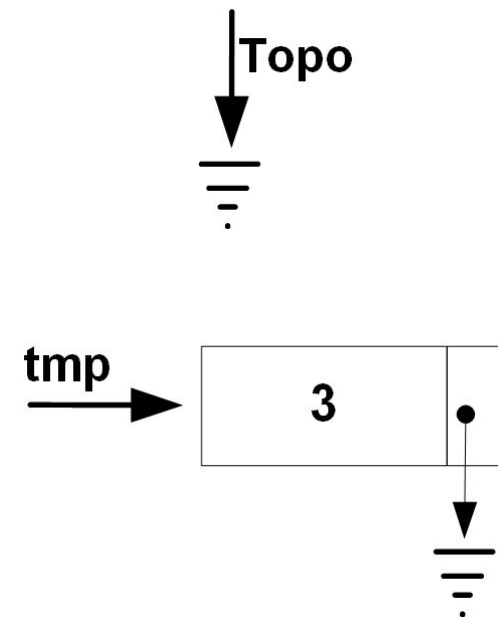
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void inserir(int x) { //Inserir(3)
    Celula *tmp = novaCelula(x);
    tmp->prox = topo;
    topo = tmp;
    tmp = NULL;
}

```

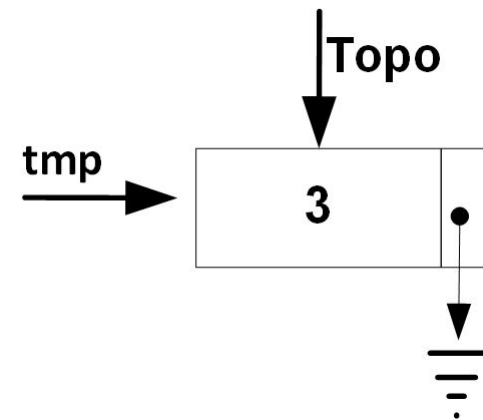


Como topo aponta para NULL, tmp->prox continua apontando para NULL



## Pilha Flexível

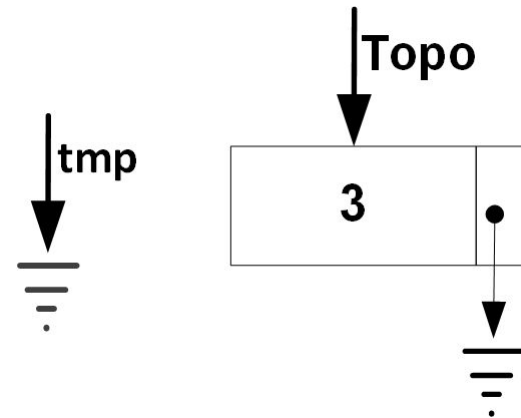
```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



```
void inserir(int x) { //Inserir(3)  
    Celula *tmp = novaCelula(x);  
    tmp->prox = topo;  
    topo = tmp;  
    tmp = NULL;  
}
```

## Pilha Flexível

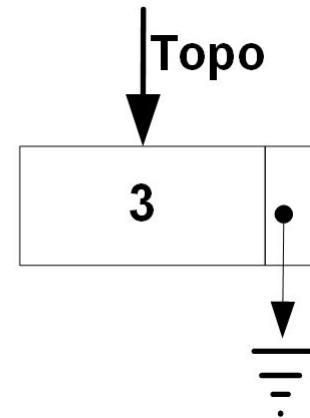
```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



```
void inserir(int x) { //Inserir(3)  
    Celula *tmp = novaCelula(x);  
    tmp->prox = topo;  
    topo = tmp;  
    tmp = NULL;  
}
```

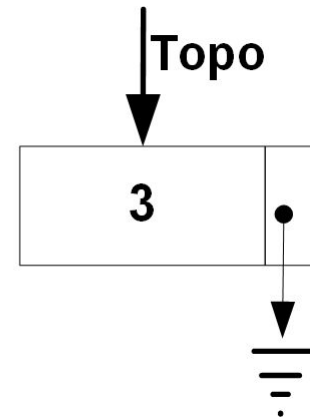
## Pilha Flexível

```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



## Pilha Flexível

```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



```
void inserir(int x) { //Inserir(5)  
    Celula *tmp = novaCelula(x);  
    tmp->prox = topo;  
    topo = tmp;  
    tmp = NULL;  
}
```

## Pilha Flexível

```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

```
void inserir(int x) { //Inserir(5)
```

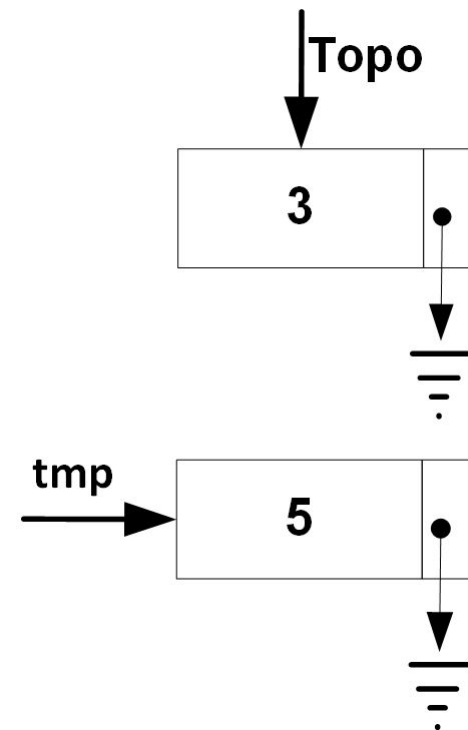
```
    Celula *tmp = novaCelula(x);
```

```
    tmp->prox = topo;
```

```
    topo = tmp;
```

```
    tmp = NULL;
```

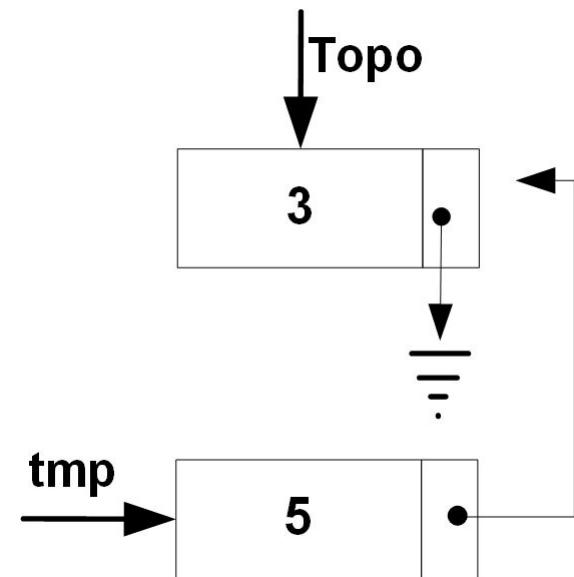
```
}
```



## Pilha Flexível

```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

```
void inserir(int x) { //Inserir(5)  
    Celula *tmp = novaCelula(x);  
    tmp->prox = topo;  
    topo = tmp;  
    tmp = NULL;  
}
```



## Pilha Flexível

```

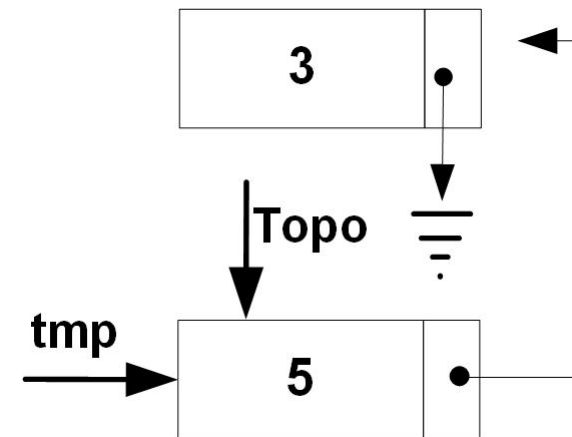
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void inserir(int x) { //Inserir(5)
    Celula *tmp = novaCelula(x);
    tmp->prox = topo;
    topo = tmp;
    tmp = NULL;
}

```



## Pilha Flexível

```

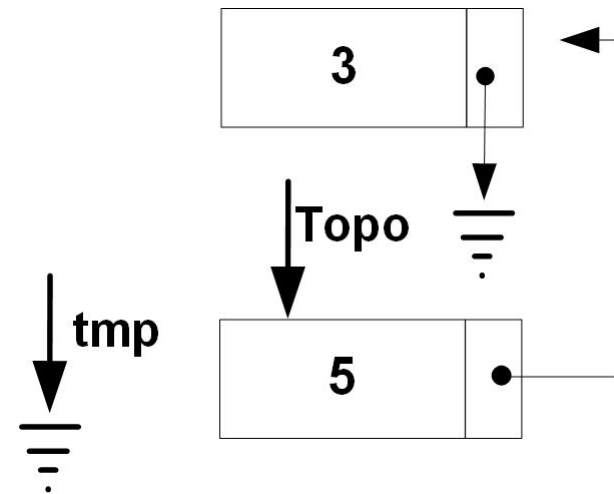
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void inserir(int x) { //Inserir(5)
    Celula *tmp = novaCelula(x);
    tmp->prox = topo;
    topo = tmp;
    tmp = NULL;
}

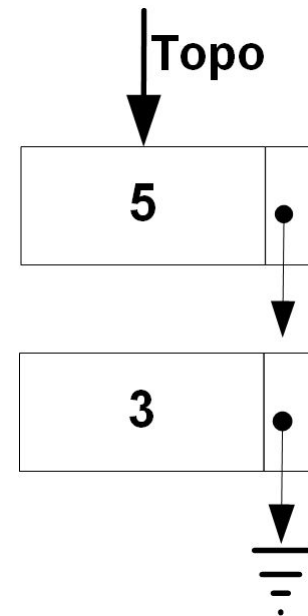
```





## Pilha Flexível

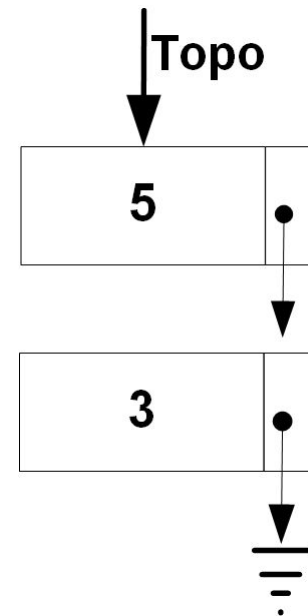
```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



## Pilha Flexível

```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

```
void inserir(int x) { //Inserir(7)  
    Celula *tmp = novaCelula(x);  
    tmp->prox = topo;  
    topo = tmp;  
    tmp = NULL;  
}
```



## Pilha Flexível

```

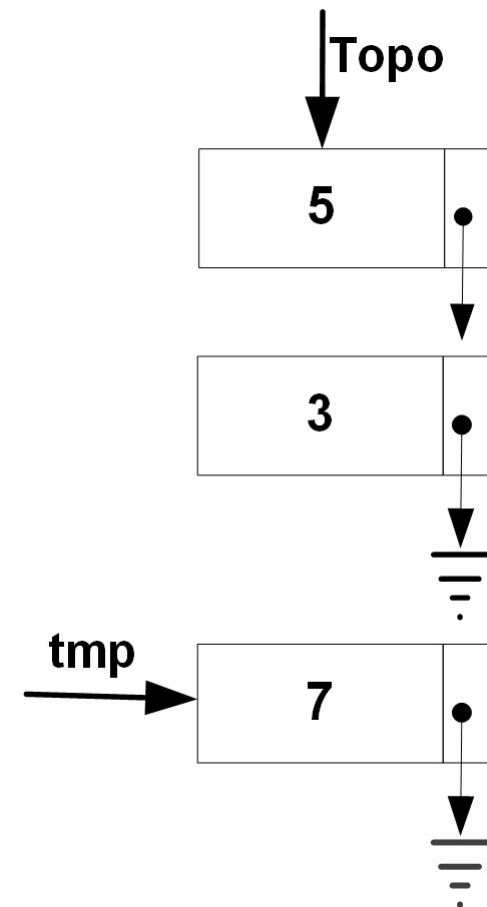
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void inserir(int x) { //Inserir(7)
    Celula *tmp = novaCelula(x);
    tmp->prox = topo;
    topo = tmp;
    tmp = NULL;
}

```



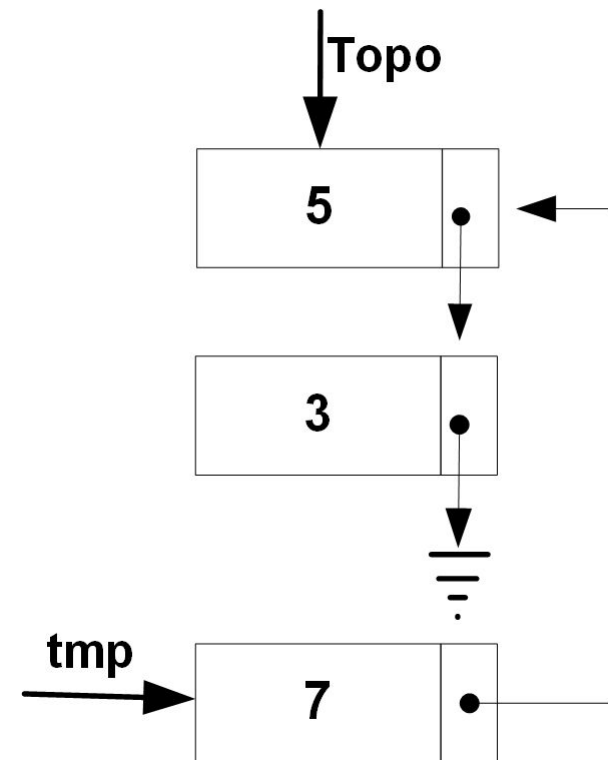
# Pilha Flexível

```

Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }
    
```

```

void inserir(int x) { //Inserir(7)
    Celula *tmp = novaCelula(x);
    tmp->prox = topo;
    topo = tmp;
    tmp = NULL;
}
    
```



## Pilha Flexível

```

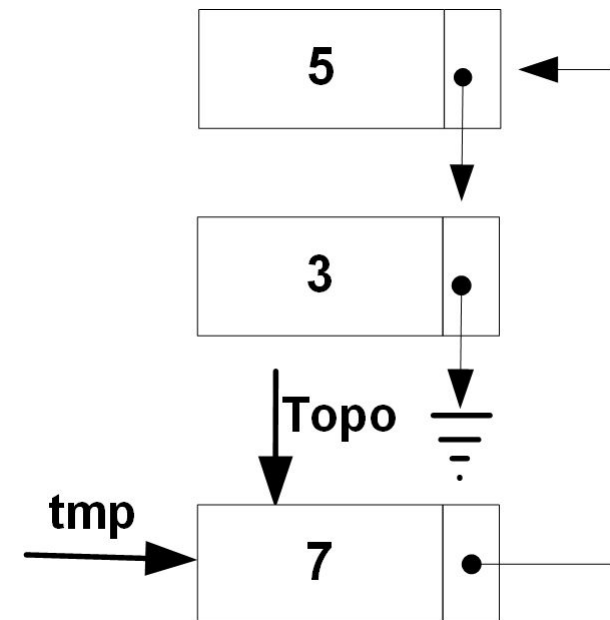
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void inserir(int x) { //Inserir(7)
    Celula *tmp = novaCelula(x);
    tmp->prox = topo;
    topo = tmp;
    tmp = NULL;
}

```



## Pilha Flexível

```

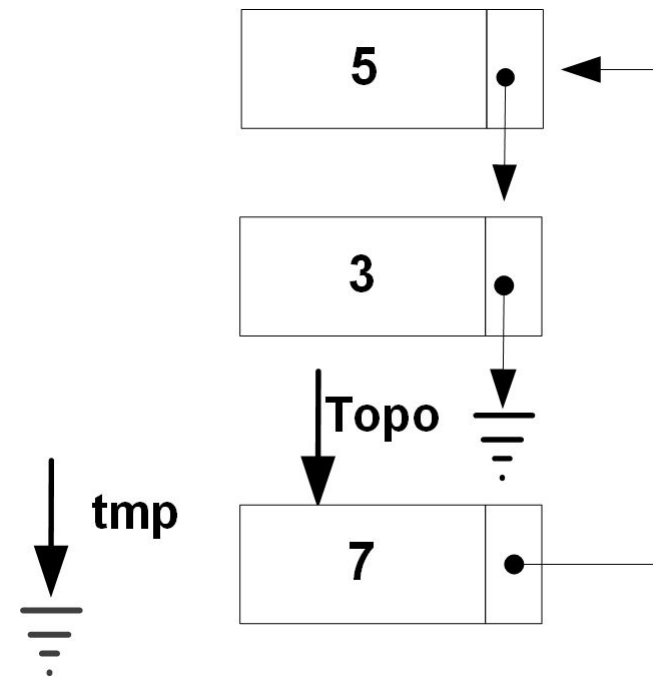
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

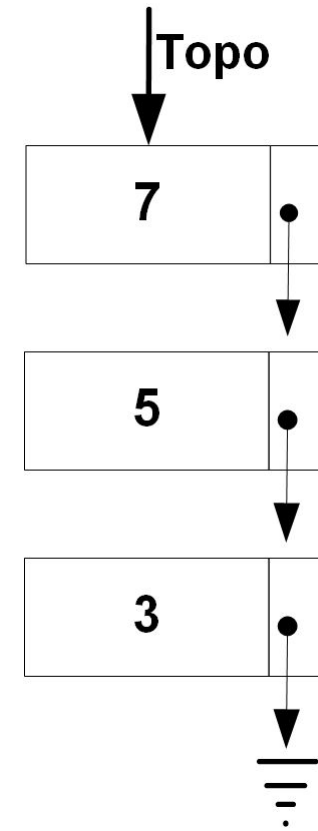
void inserir(int x) { //Inserir(7)
    Celula *tmp = novaCelula(x);
    tmp->prox = topo;
    topo = tmp;
    tmp = NULL;
}

```



## Pilha Flexível

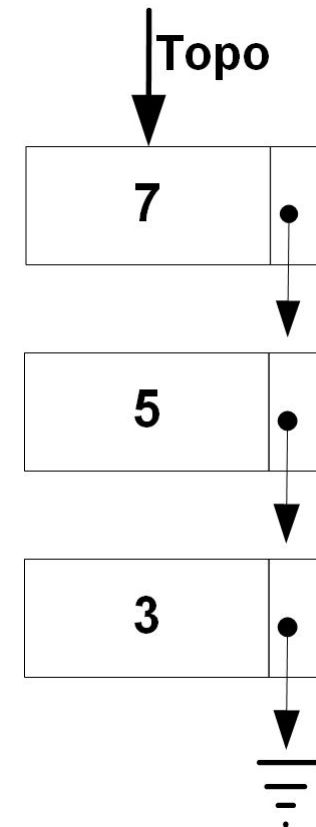
```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



## Pilha Flexível

```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

```
int remover() {  
    if (topo == NULL)  
        errx(1, "Erro!");  
    int elemento = topo->elemento;  
    Celula *tmp = topo;  
    topo = topo->prox;  
    tmp->prox = NULL;  
    free(tmp);          tmp = NULL;  
    return elemento;  
}
```





## Pilha Flexível

```

Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

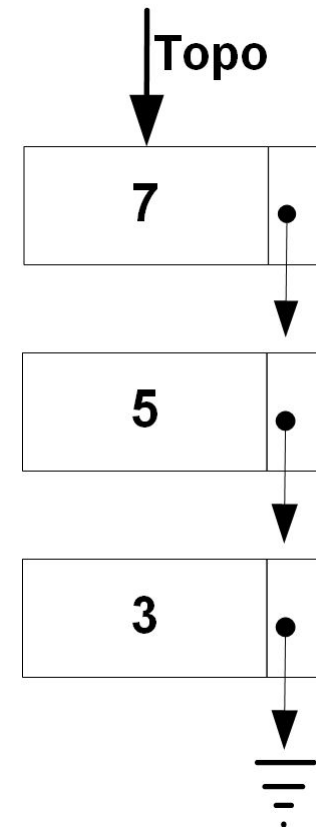
```

```

int remover() {
if (topo == NULL)
    errx(1, "Erro!");
    int elemento = topo->elemento;
    Celula *tmp = topo;
    topo = topo->prox;
    tmp->prox = NULL;
    free(tmp);      tmp = NULL;
    return elemento;
}

```

false



## Pilha Flexível

```

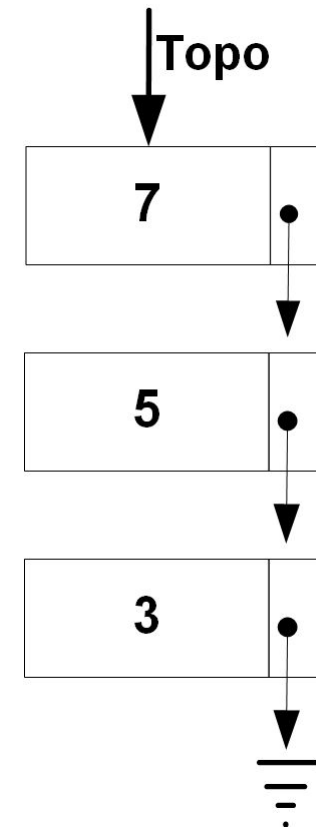
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover() {
    if (topo == NULL)
        errx(1, "Erro!");
    int elemento = topo->elemento;
    Celula *tmp = topo;
    topo = topo->prox;
    tmp->prox = NULL;
    free(tmp);          tmp = NULL;
    return elemento;
}

```



elemento 

7
---

## Pilha Flexível

```

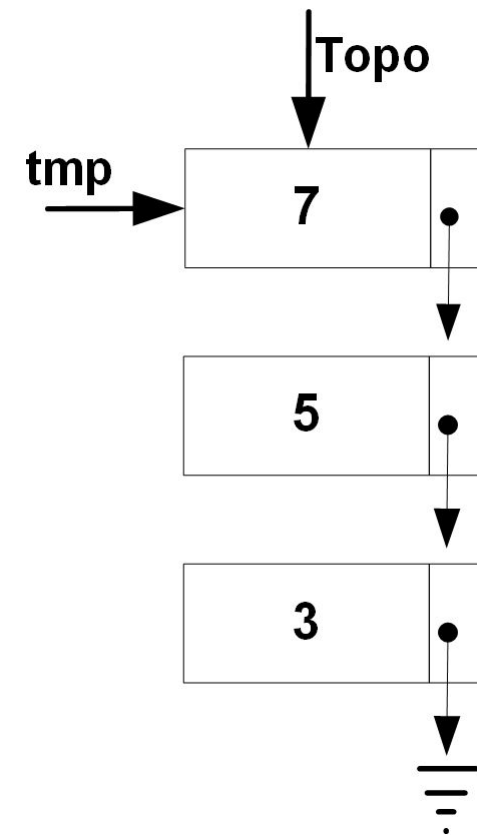
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover() {
    if (topo == NULL)
        errx(1, "Erro!");
    int elemento = topo->elemento;
    Celula *tmp = topo;
    topo = topo->prox;
    tmp->prox = NULL;
    free(tmp);          tmp = NULL;
    return elemento;
}

```



elemento 

7
---

## Pilha Flexível

```

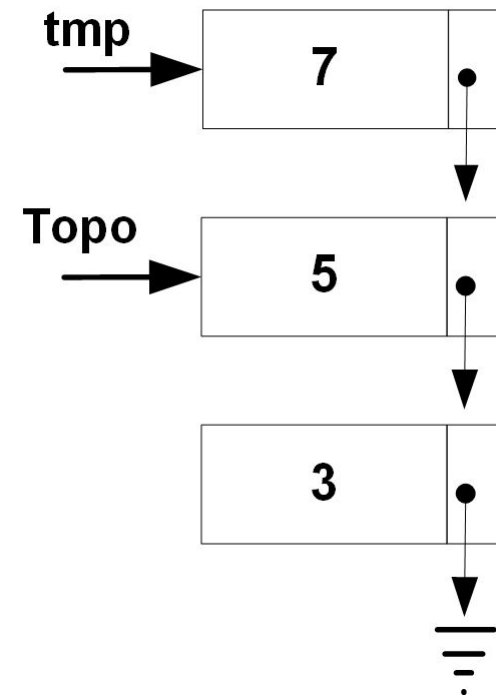
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover() {
    if (topo == NULL)
        errx(1, "Erro!");
    int elemento = topo->elemento;
    Celula *tmp = topo;
    topo = topo->prox;
    tmp->prox = NULL;
    free(tmp);      tmp = NULL;
    return elemento;
}

```



elemento 

7
---

## Pilha Flexível

```

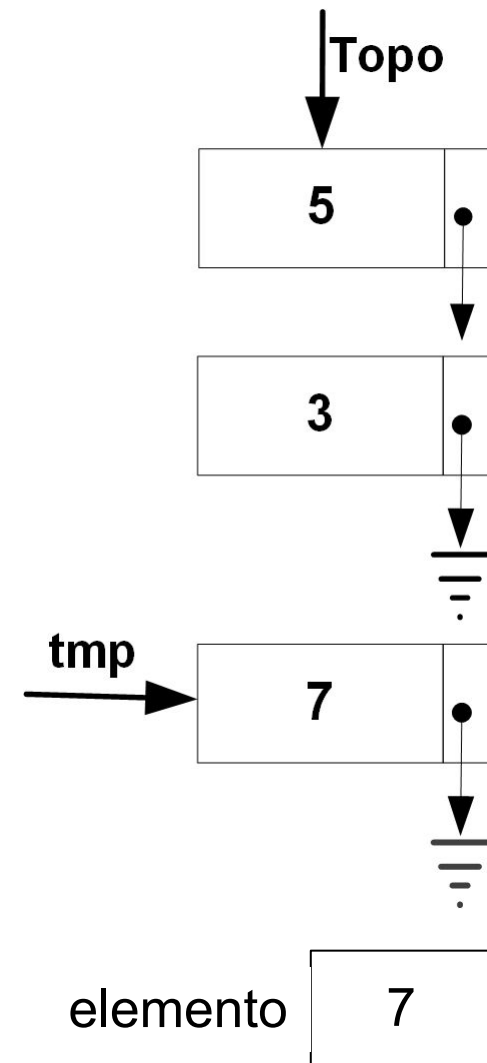
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover() {
    if (topo == NULL)
        errx(1, "Erro!");
    int elemento = topo->elemento;
    Celula *tmp = topo;
    topo = topo->prox;
    tmp->prox = NULL;
    free(tmp);          tmp = NULL;
    return elemento;
}

```



## Pilha Flexível

```

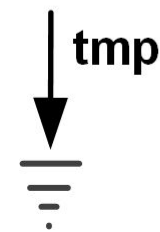
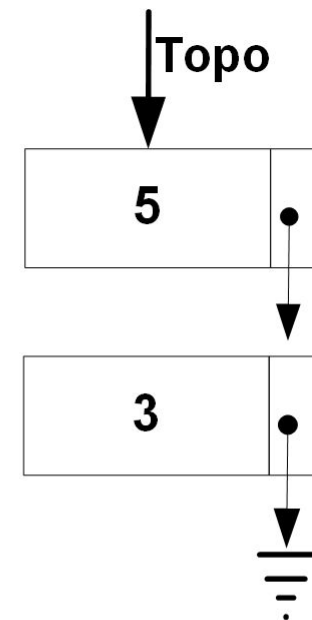
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover() {
    if (topo == NULL)
        errx(1, "Erro!");
    int elemento = topo->elemento;
    Celula *tmp = topo;
    topo = topo->prox;
    tmp->prox = NULL;
    free(tmp);      tmp = NULL;
    return elemento;
}

```



elemento 7

## Pilha Flexível

```

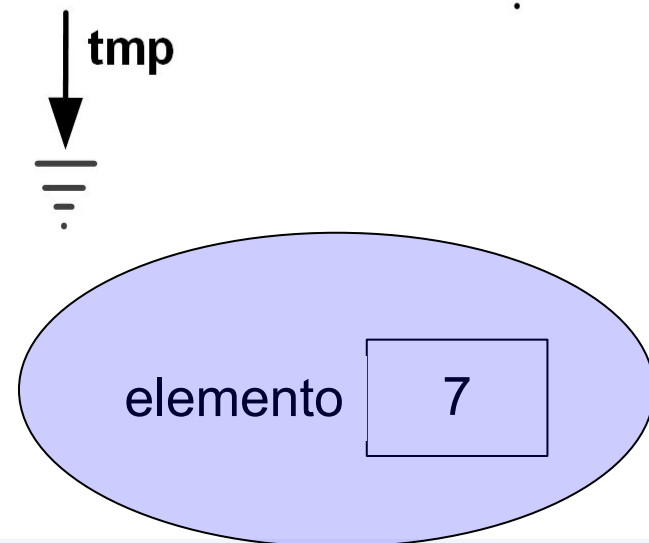
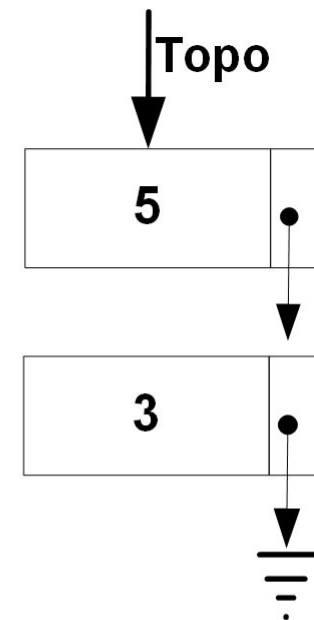
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

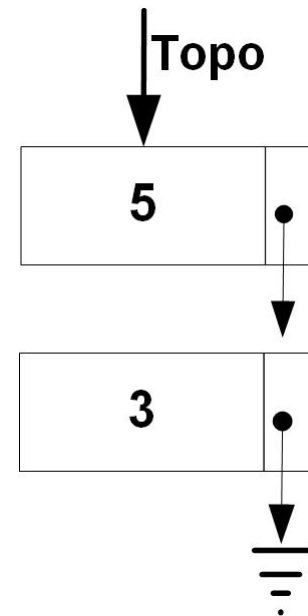
int remover() {
    if (topo == NULL)
        errx(1, "Erro!");
    int elemento = topo->elemento;
    Celula *tmp = topo;
    topo = topo->prox;
    tmp->prox = NULL;
    free(tmp);          tmp = NULL;
    return elemento;
}

```



## Pilha Flexível

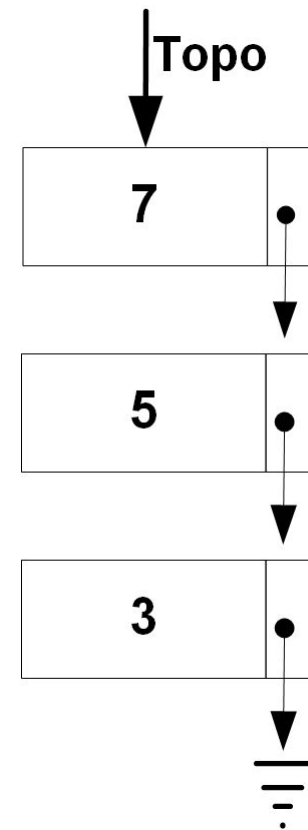
```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```





## Pilha Flexível

```
Celula *topo;  
void start() {  
    topo = NULL;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



## Pilha Flexível

```

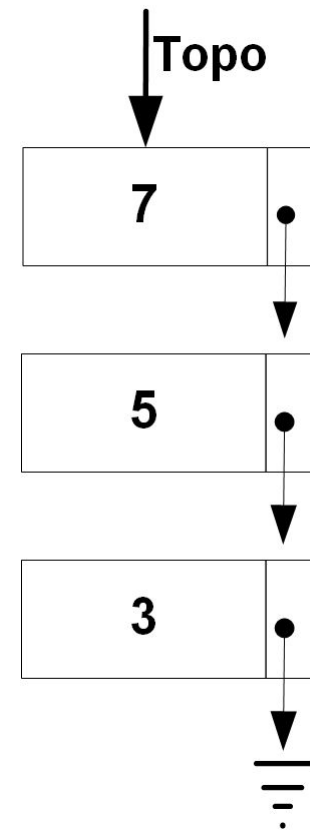
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

## Pilha Flexível

```

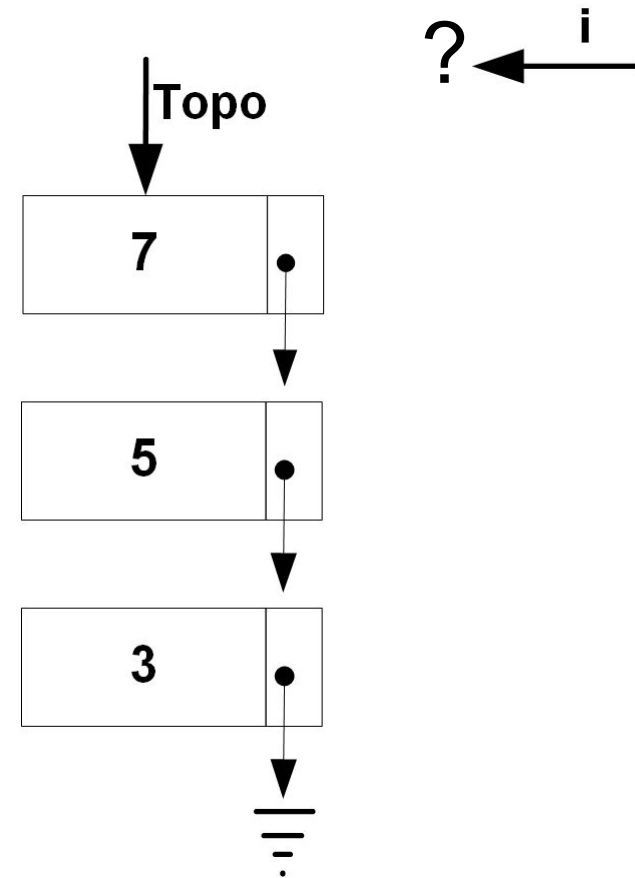
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

## Pilha Flexível

```

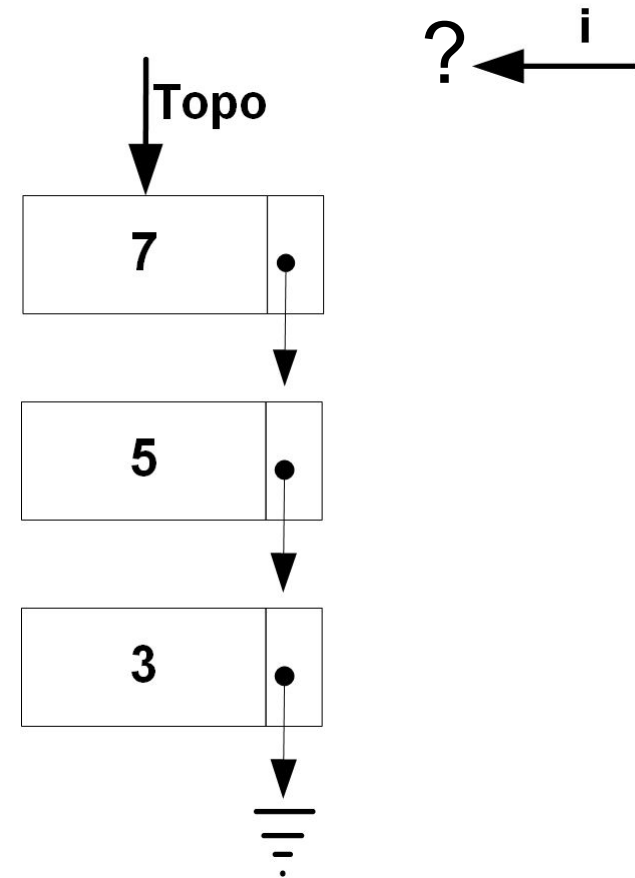
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

[

## Pilha Flexível

```

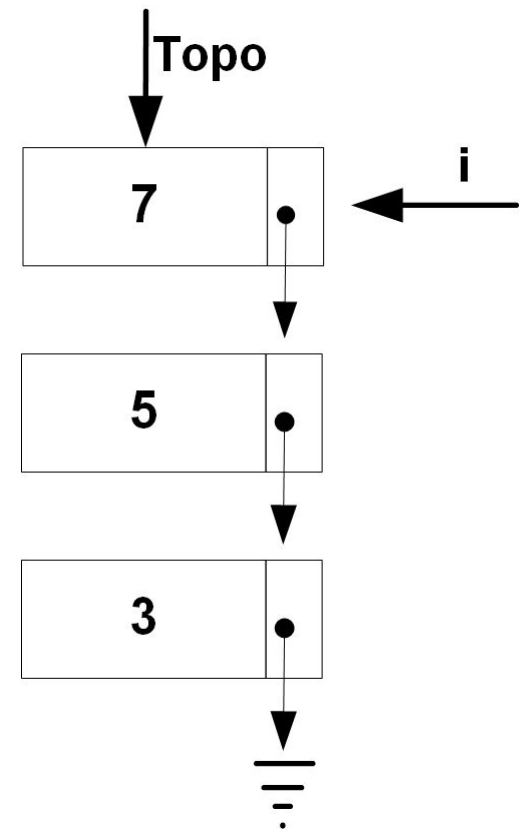
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

[

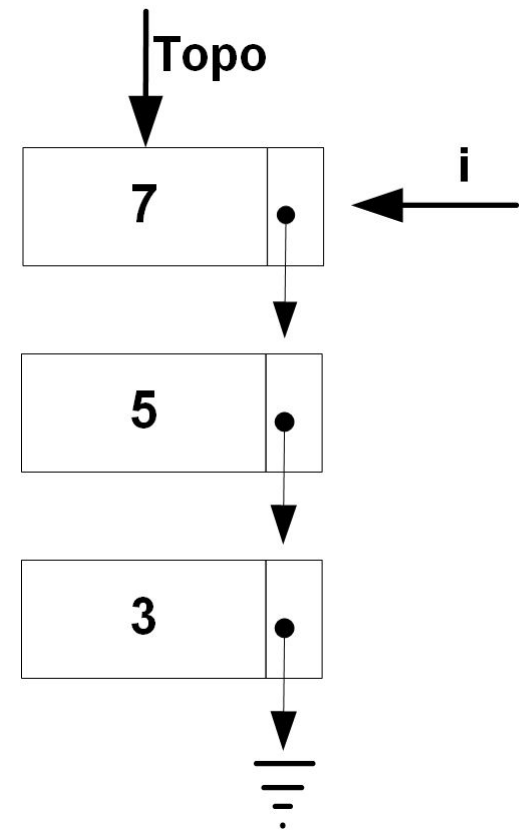
# Pilha Flexível

```

Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }
    
```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}
    
```



Saída  
na tela

[

## Pilha Flexível

```

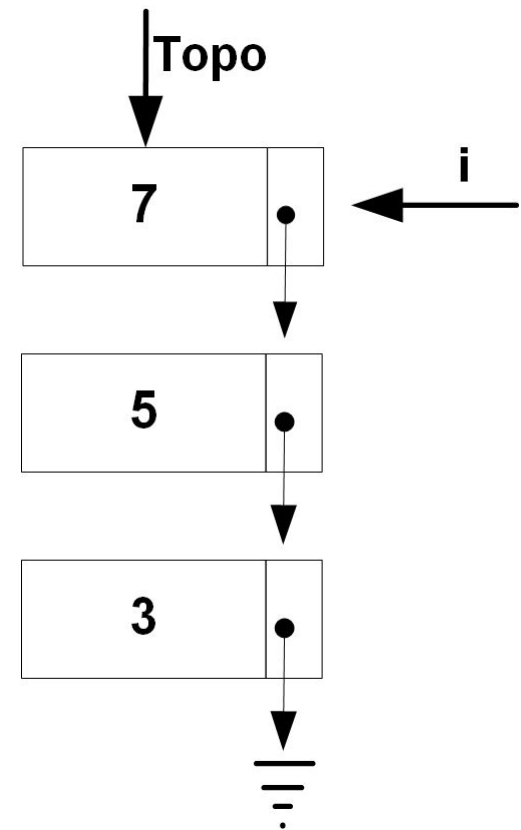
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

[ 7

## Pilha Flexível

```

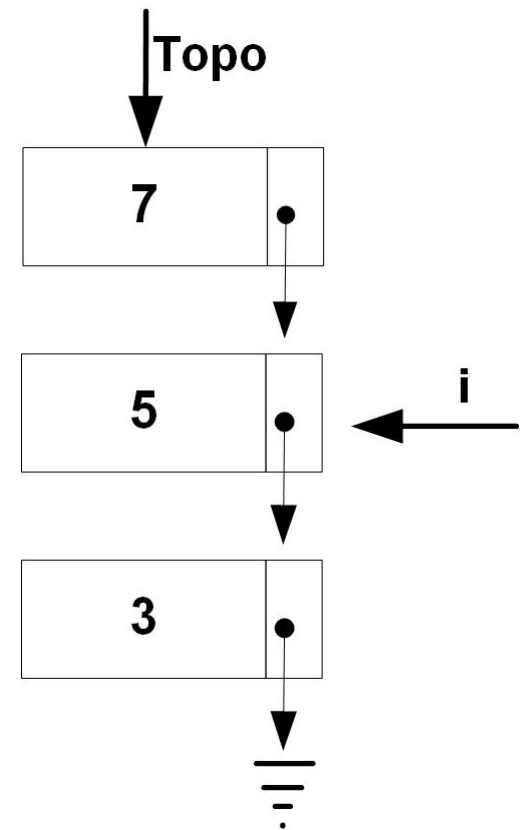
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

[ 7



## Pilha Flexível

```

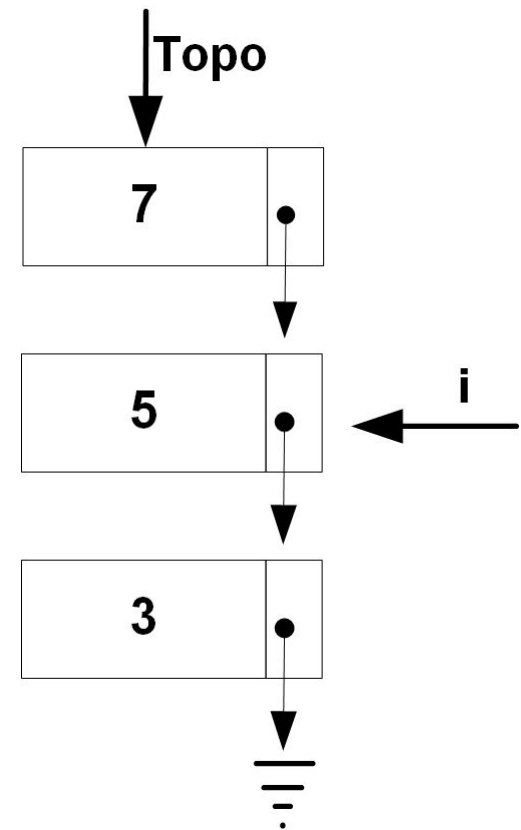
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

[ 7

## Pilha Flexível

```

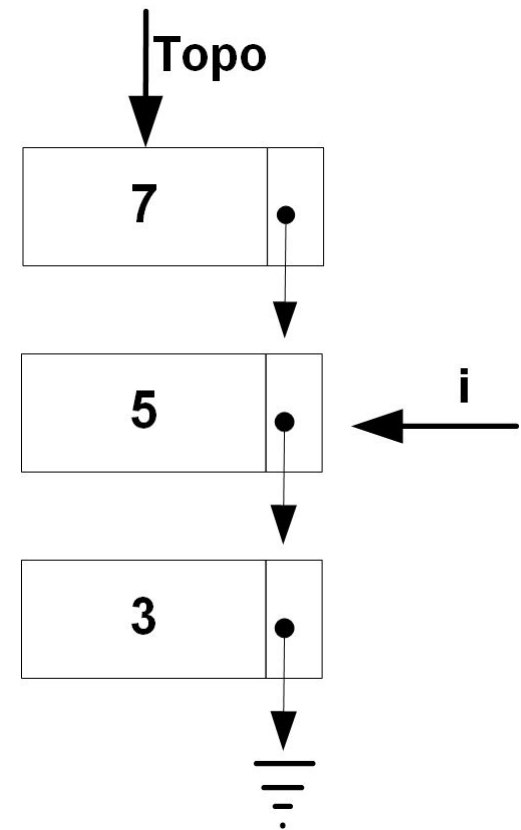
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

[ 7 5

## Pilha Flexível

```

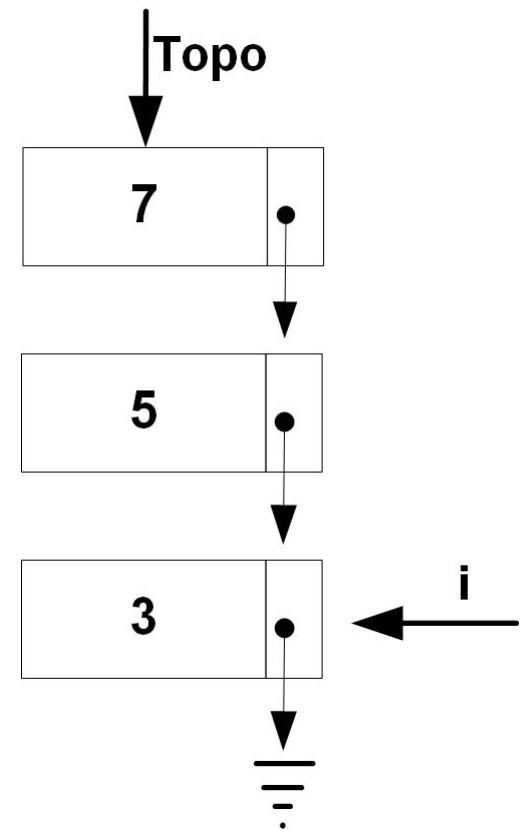
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

[ 7 5

## Pilha Flexível

```

Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

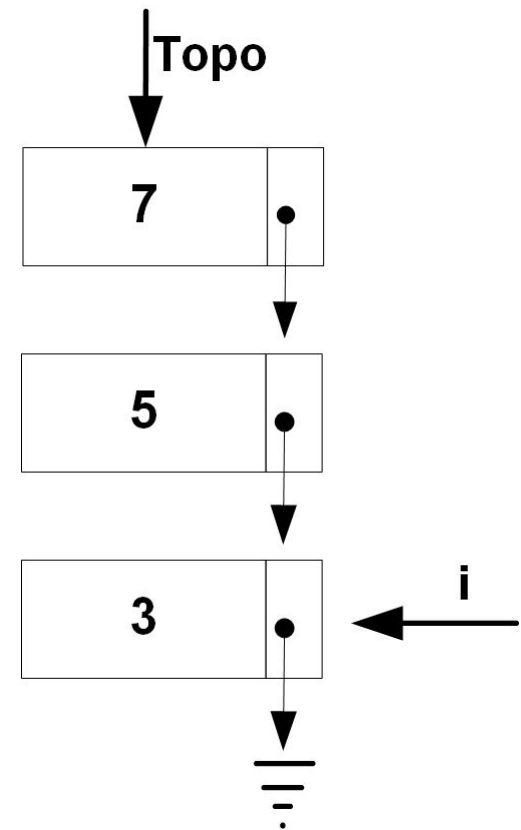
```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```

true



Saída  
na tela

[ 7 5

## Pilha Flexível

```

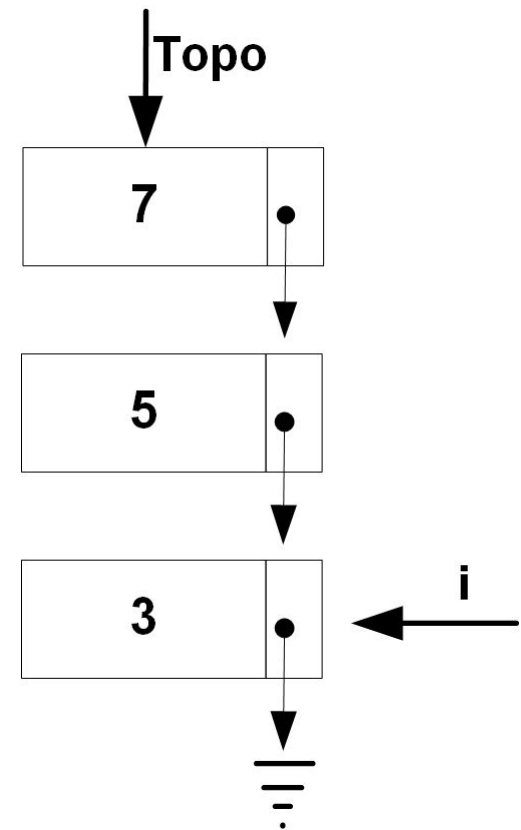
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

[ 7 5 3

## Pilha Flexível

```

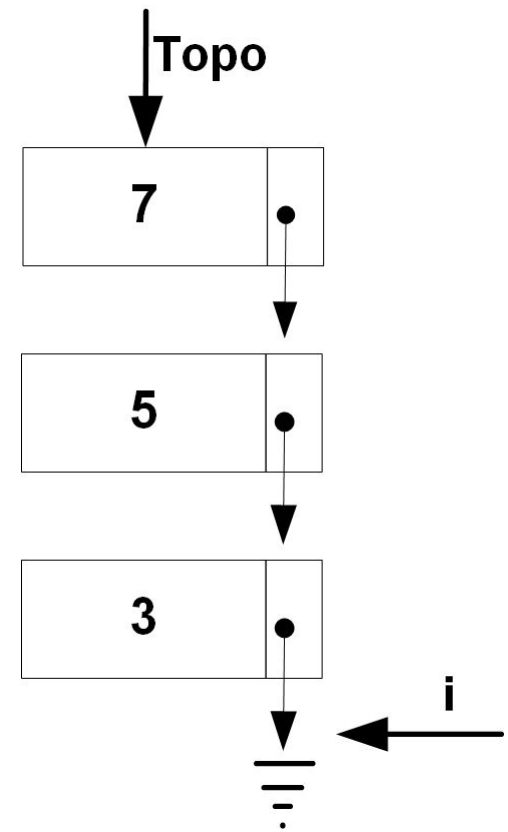
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

[ 7 5 3

## Pilha Flexível

```

Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

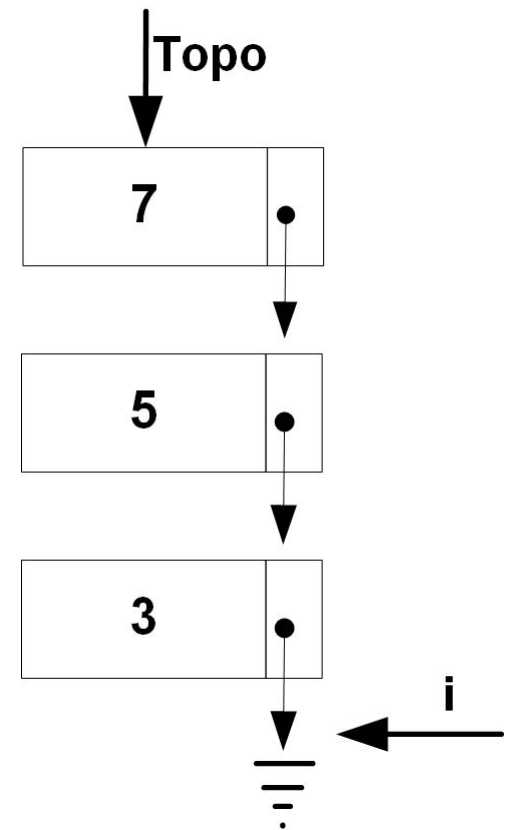
```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```

false



Saída  
na tela

[ 7 5 3

## Pilha Flexível

```

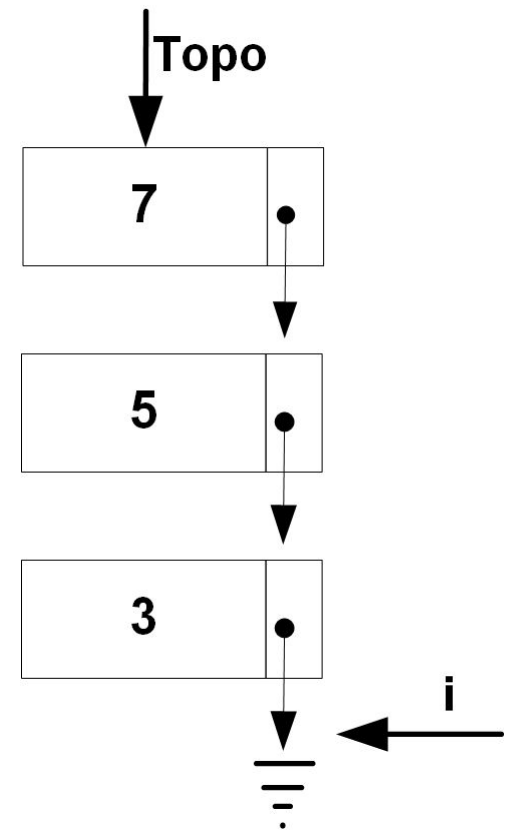
Celula *topo;
void start() {
    topo = NULL;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void mostrar() {
    Celula *i;
    printf("[ ");
    for (i = topo; i != NULL; i = i->prox){
        printf("%i ", i->elemento);
    }
    printf("]");
}

```



Saída  
na tela

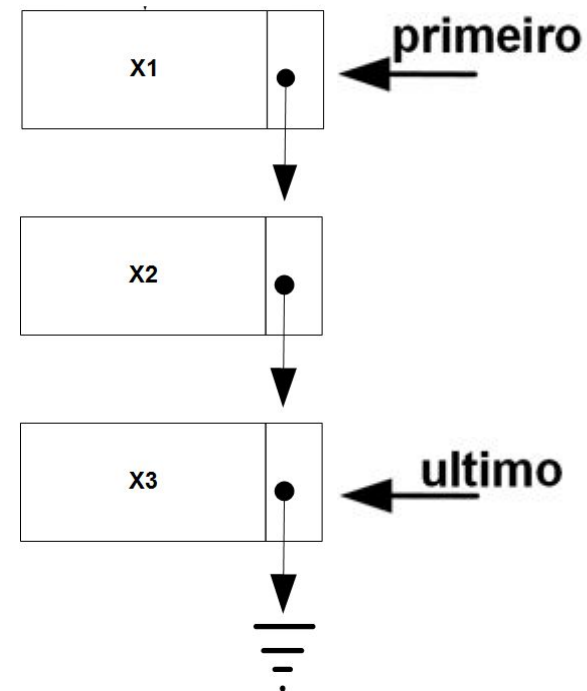
[ 7 5 3 ]



- Alocação dinâmica
- Pilha flexível
- **Fila flexível**
- Lista simples flexível (lista simplesmente encadeada)
- Lista dupla flexível (lista duplamente encadeada)

# Fila Flexível

- Código fonte:



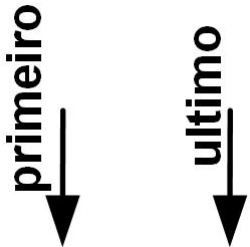
## Fila Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

## Fila Flexível

```
Celula *primeiro, *ultimo;
```

```
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



## Fila Flexível

```
Celula *primeiro, *ultimo;
```

```
void start () {
```

```
    primeiro = novaCelula(-1);
```

```
    ultimo = primeiro;
```

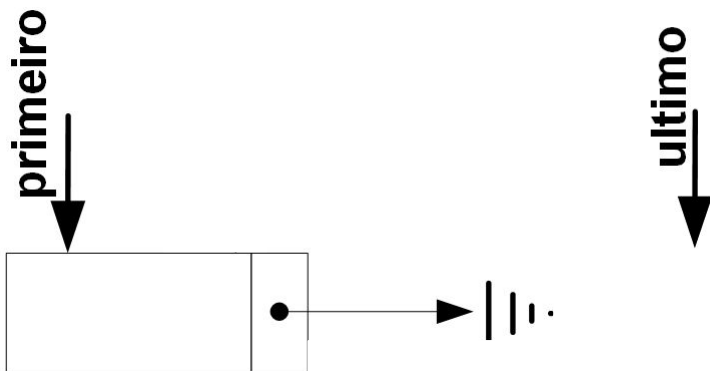
```
}
```

```
void inserir(int x) { ... }
```

```
int remover() { ... }
```

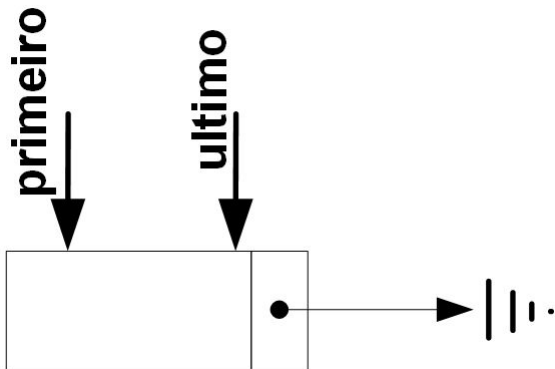
```
void mostrar() { ... }
```

A primeira célula da nossa fila é o nó cabeça, célula “café com leite” cuja função é eliminar um if no inserir



## Fila Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```



## Fila Flexível

```

Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}

```

```

void inserir(int x) { ... }

```

```

int remover() { ... }

```

```

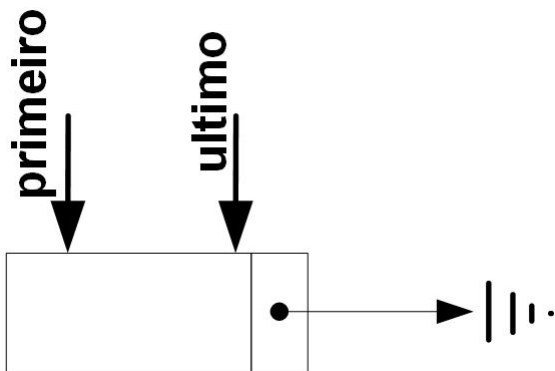
void mostrar() { ... }

```

```

void inserir(int x) { //Inserir(3)
    ultimo->prox = novaCelula(x);
    ultimo = ultimo->prox;
}

```



## Fila Flexível

```

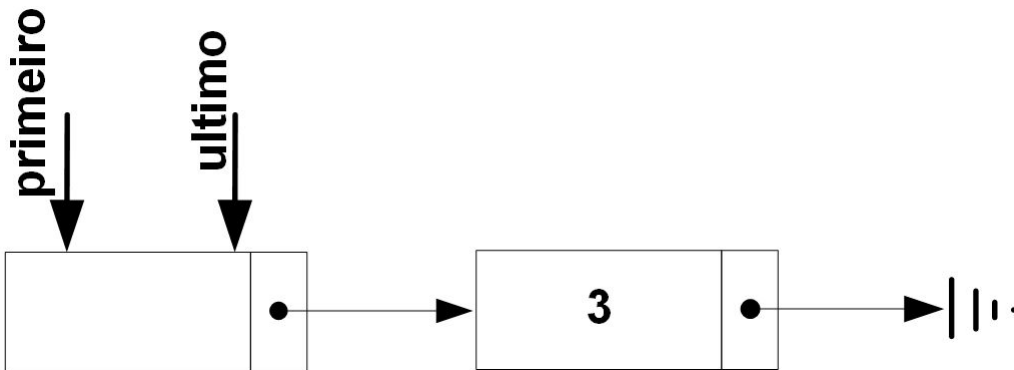
Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void inserir(int x) { //Inserir(3)
    ultimo->prox = novaCelula(x);
    ultimo = ultimo->prox;
}

```





## Fila Flexível

```

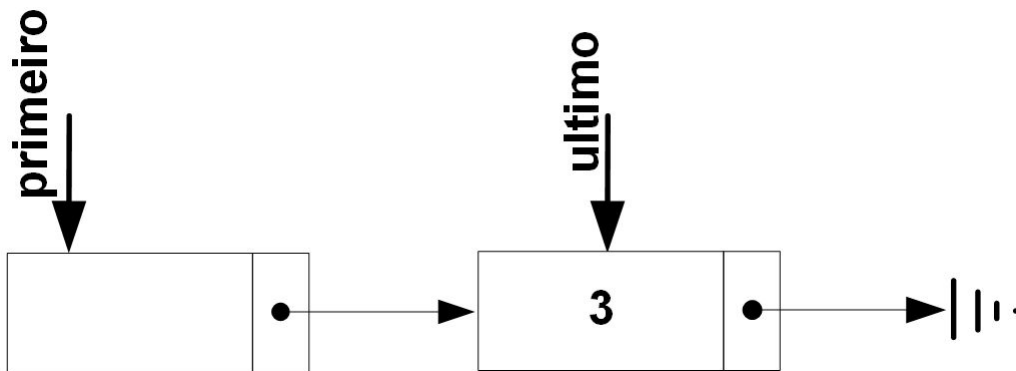
Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void inserir(int x) { //Inserir(3)
    ultimo->prox = novaCelula(x);
    ultimo = ultimo->prox;
}

```



## Fila Flexível

```

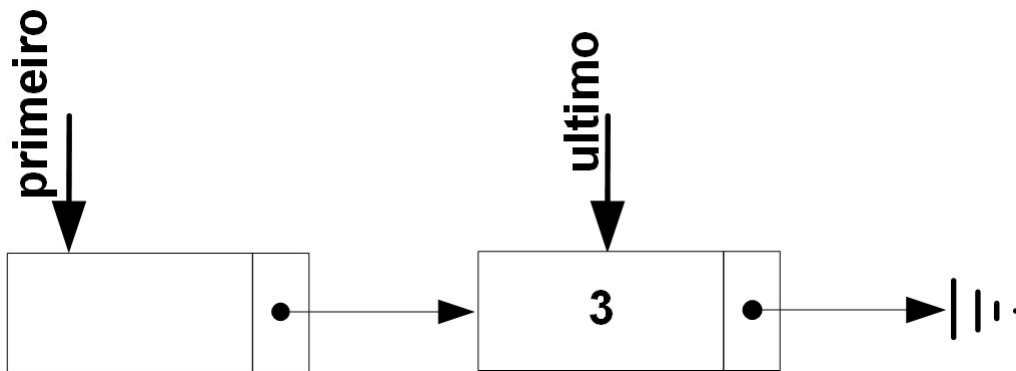
Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

void inserir(int x) { //Inserir(5)
    ultimo->prox = novaCelula(x);
    ultimo = ultimo->prox;
}

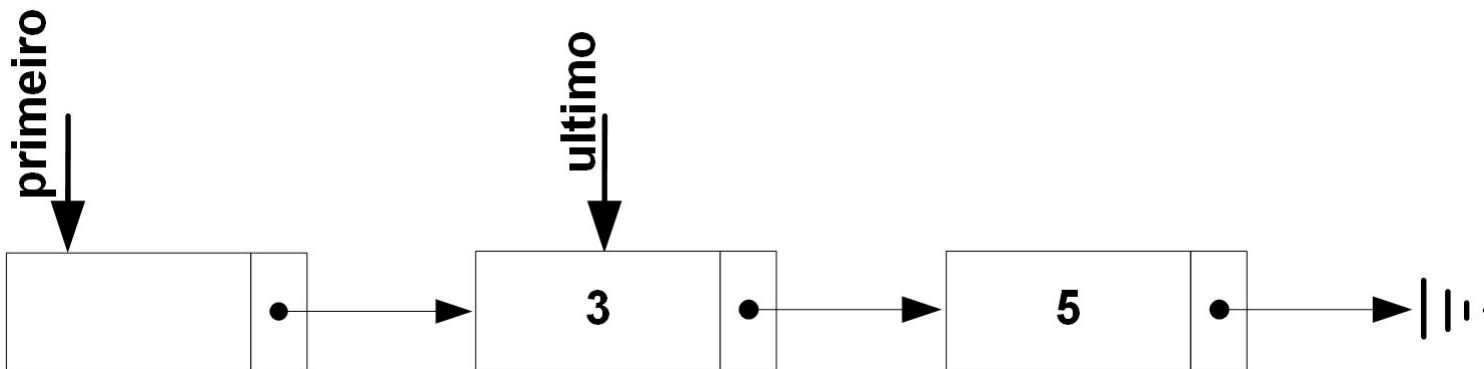
```



## Fila Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

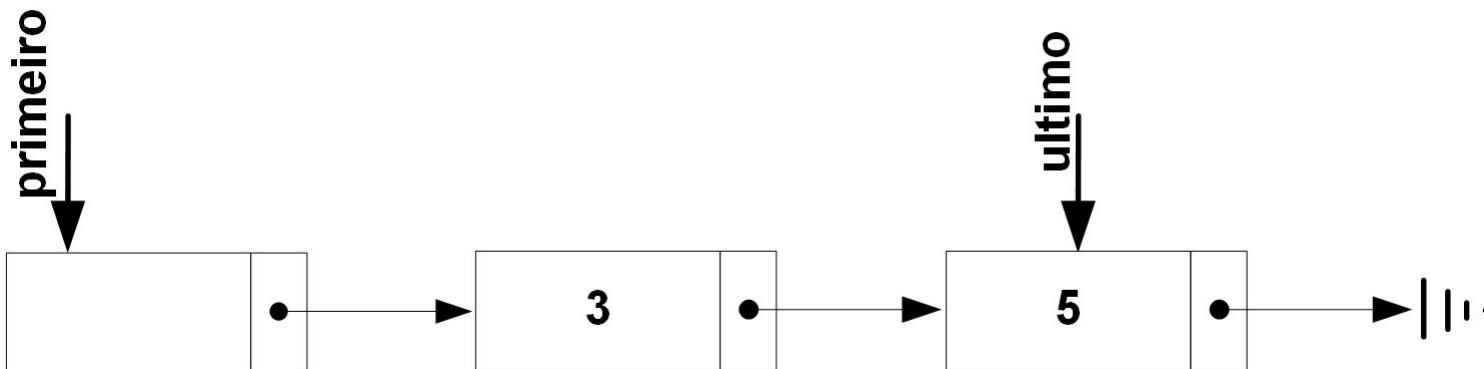
```
void inserir(int x) { //Inserir(5)  
    ultimo->prox = novaCelula(x);  
    ultimo = ultimo->prox;  
}
```



## Fila Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

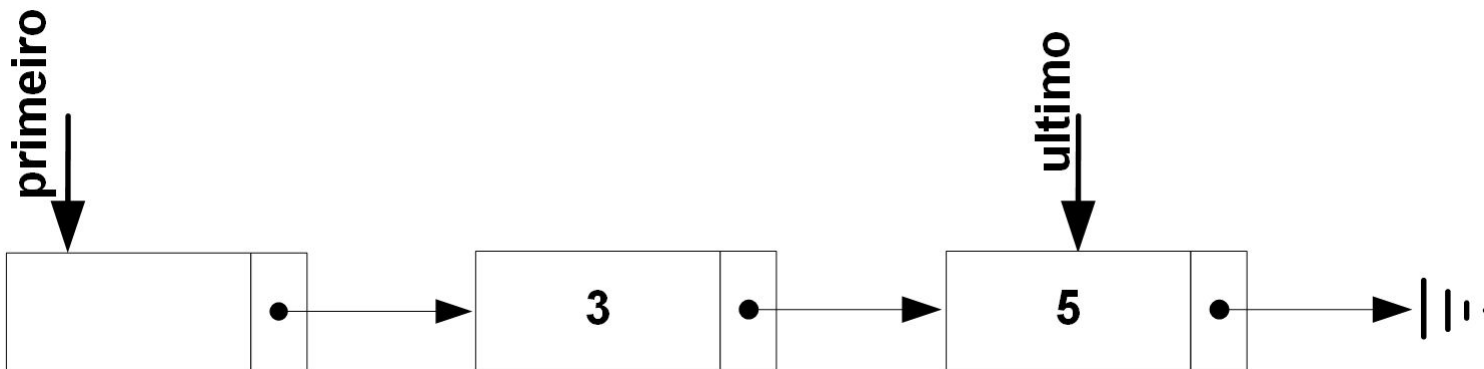
```
void inserir(int x) { //Inserir(5)  
    ultimo->prox = novaCelula(x);  
    ultimo = ultimo->prox;  
}
```



## Fila Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

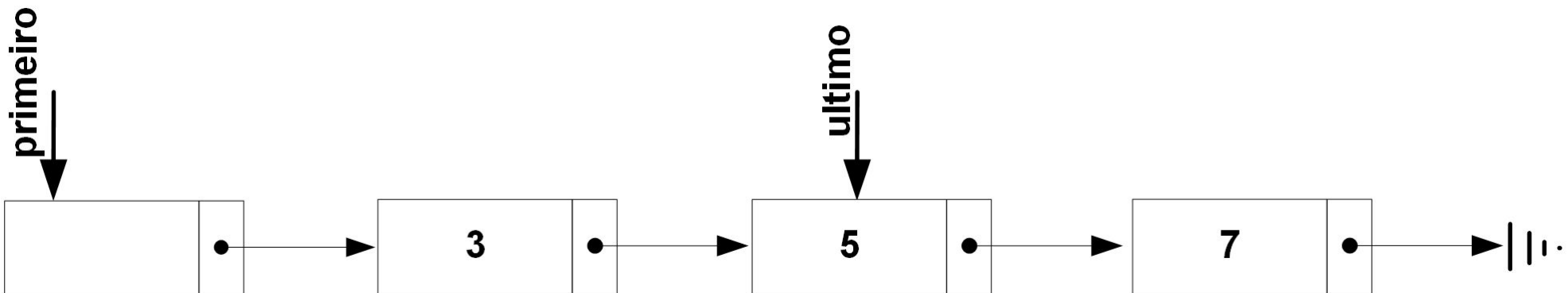
```
void inserir(int x) { //Inserir(7)  
    ultimo->prox = novaCelula(x);  
    ultimo = ultimo->prox;  
}
```



## Fila Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

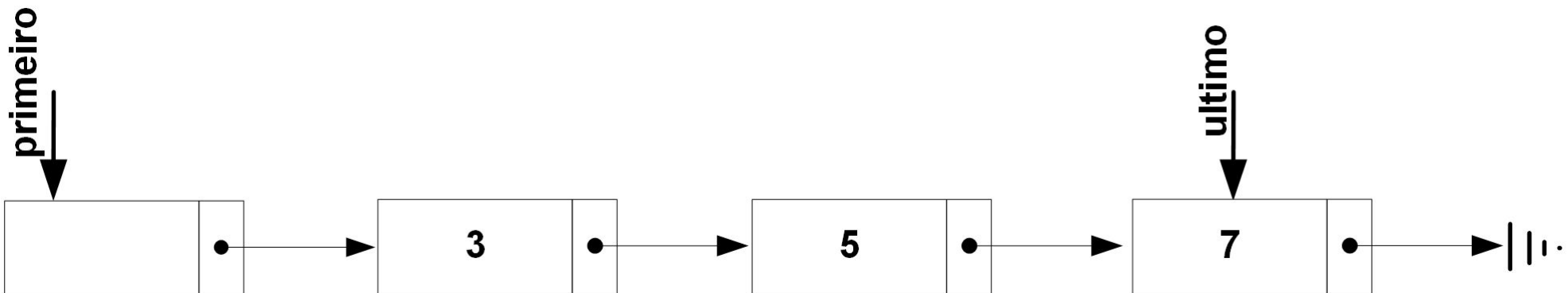
```
void inserir(int x) { //Inserir(7)  
    ultimo->prox = novaCelula(x);  
    ultimo = ultimo->prox;  
}
```



## Fila Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

```
void inserir(int x) { //Inserir(7)  
    ultimo->prox = novaCelula(x);  
    ultimo = ultimo->prox;  
}
```



## Fila Flexível

```

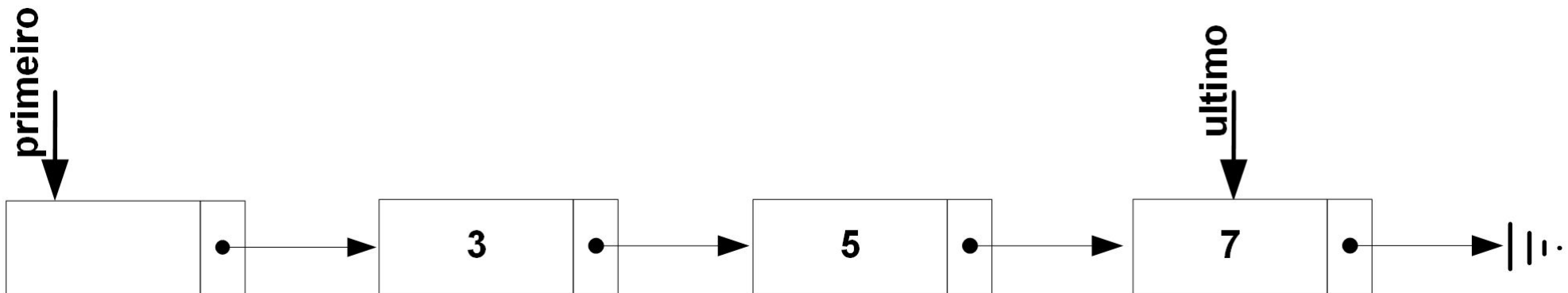
Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover(){
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *tmp = primeiro;
    primeiro = primeiro->prox;
    int elemento = primeiro->elemento;
    tmp->prox = NULL;
    free(tmp);      tmp = NULL;
    return elemento;
}

```





## Fila Flexível

```

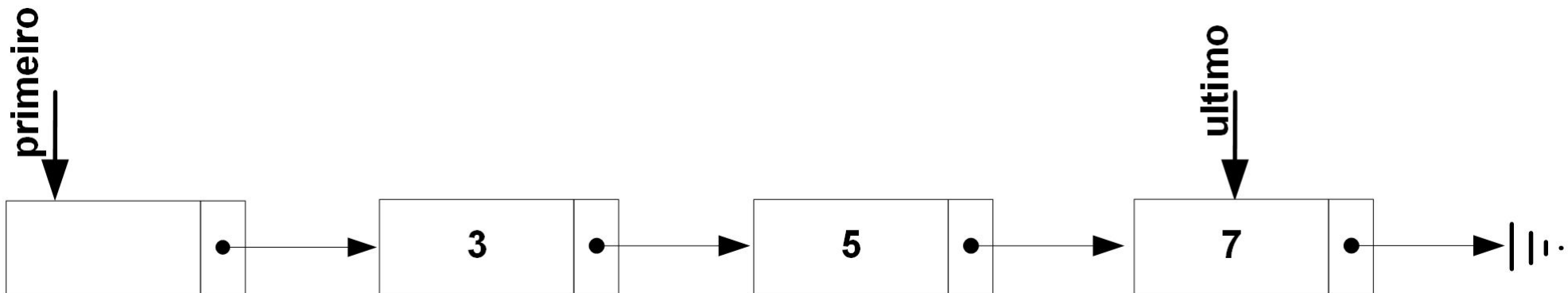
Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover(){
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *tmp = primeiro;
    primeiro = primeiro->prox;
    int elemento = primeiro->elemento;
    tmp->prox = NULL;
    free(tmp);    tmp = NULL;
    return elemento;
}

```



## Fila Flexível

```

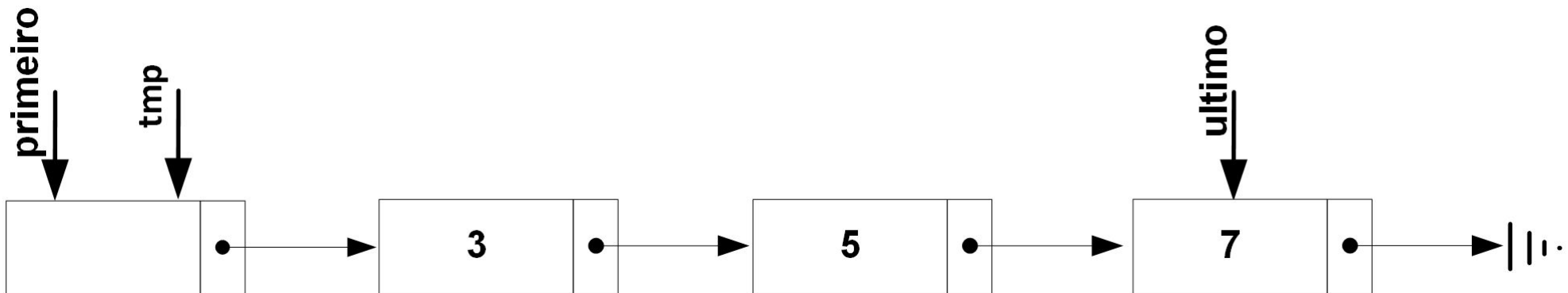
Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover(){
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *tmp = primeiro;
    primeiro = primeiro->prox;
    int elemento = primeiro->elemento;
    tmp->prox = NULL;
    free(tmp);      tmp = NULL;
    return elemento;
}

```



## Fila Flexível

```

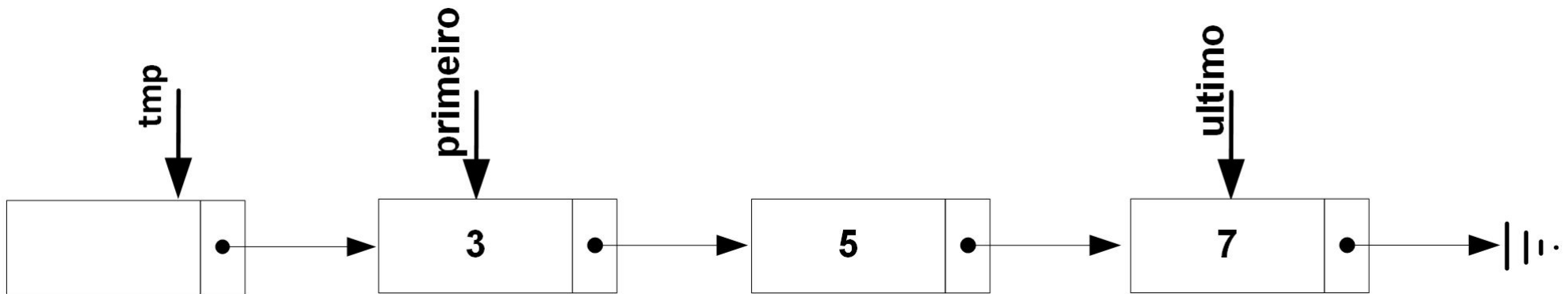
Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover(){
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *tmp = primeiro;
    primeiro = primeiro->prox;
    int elemento = primeiro->elemento;
    tmp->prox = NULL;
    free(tmp);      tmp = NULL;
    return elemento;
}

```



## Fila Flexível

```

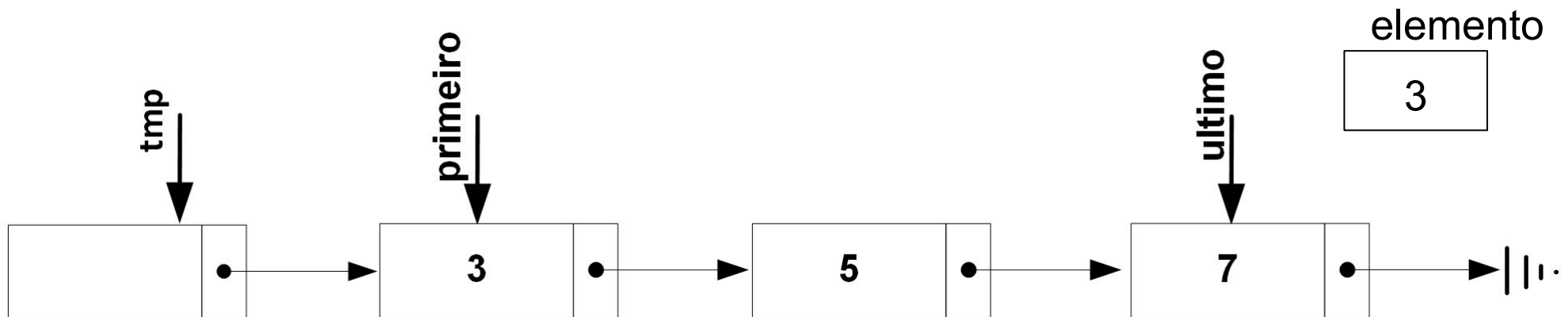
Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover(){
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *tmp = primeiro;
    primeiro = primeiro->prox;
    int elemento = primeiro->elemento;
    tmp->prox = NULL;
    free(tmp);      tmp = NULL;
    return elemento;
}

```



## Fila Flexível

```

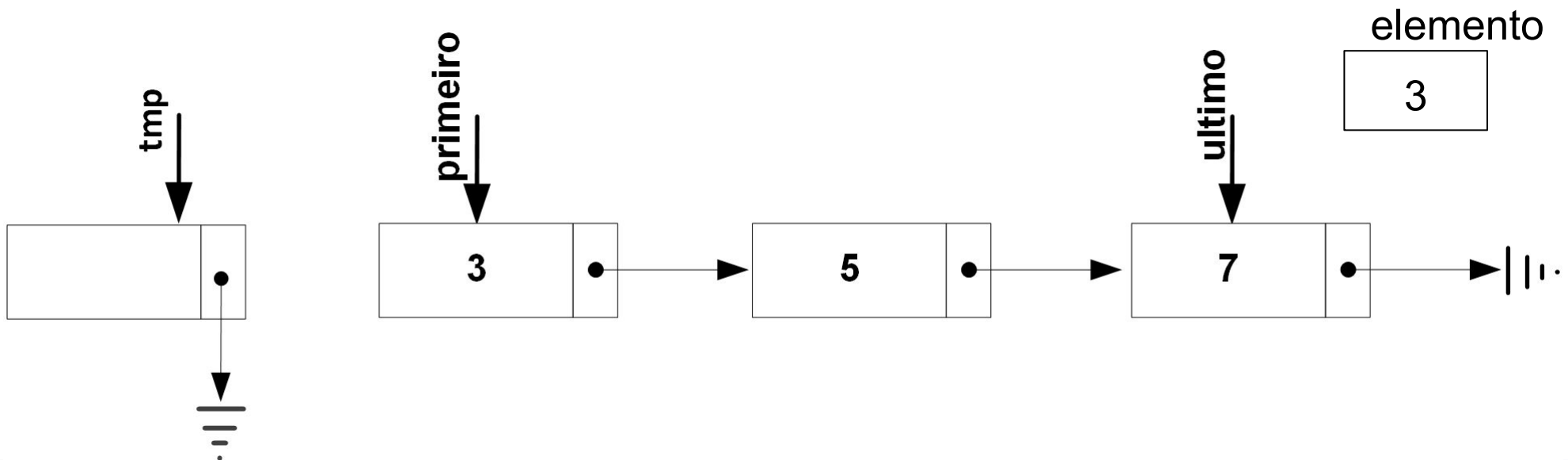
Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover(){
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *tmp = primeiro;
    primeiro = primeiro->prox;
    int elemento = primeiro->elemento;
    tmp->prox = NULL;
    free(tmp);      tmp = NULL;
    return elemento;
}

```



## Fila Flexível

```

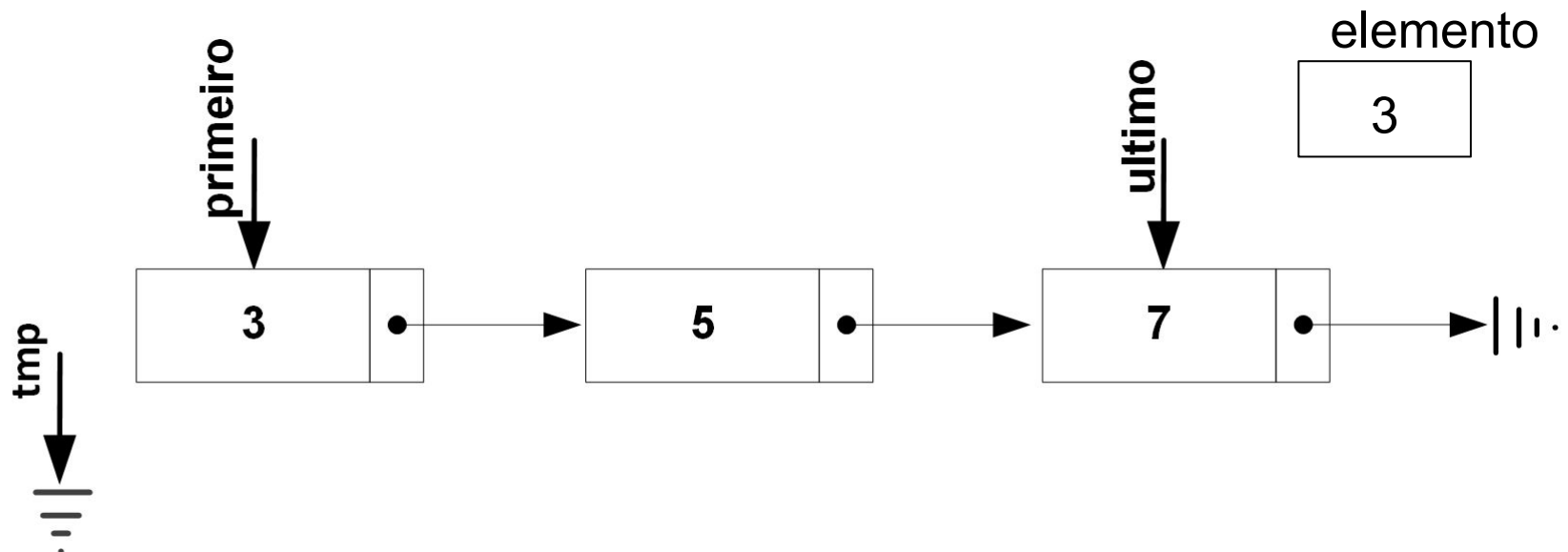
Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover(){
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *tmp = primeiro;
    primeiro = primeiro->prox;
    int elemento = primeiro->elemento;
    tmp->prox = NULL;
    free(tmp);      tmp = NULL;
    return elemento;
}

```



## Fila Flexível

```

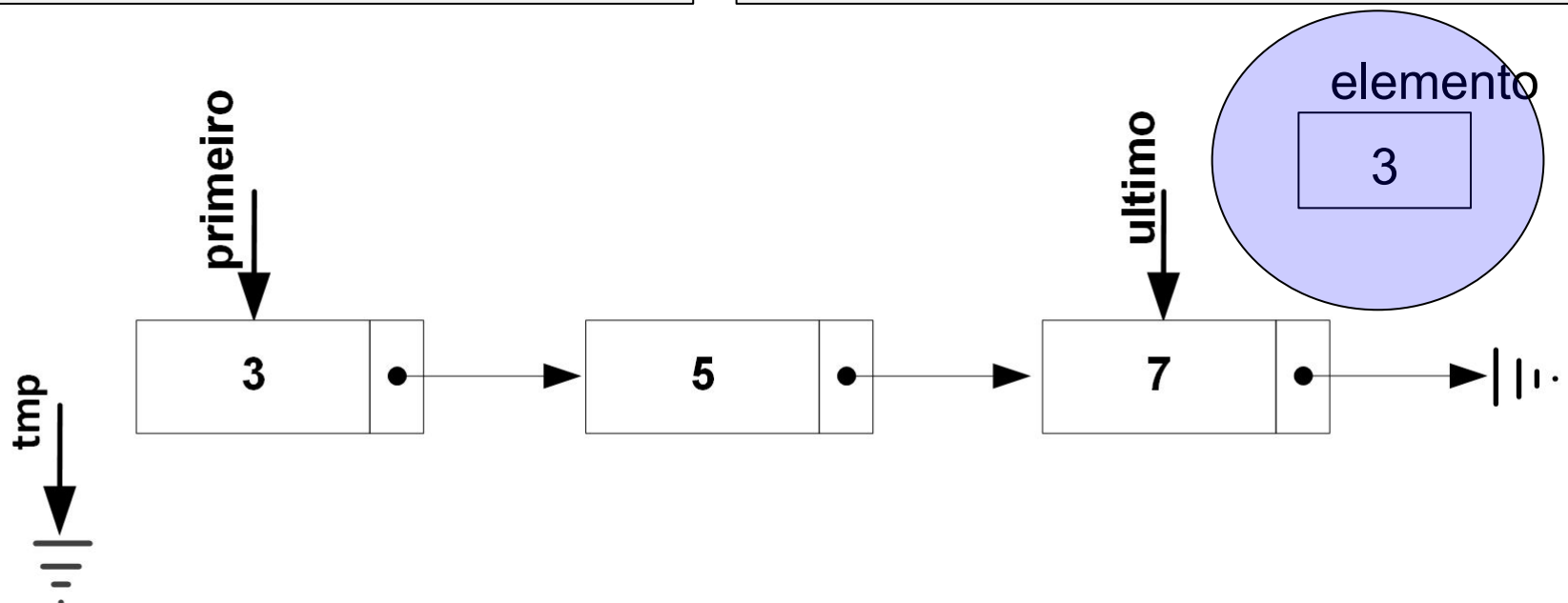
Celula *primeiro, *ultimo;
void start () {
    primeiro = novaCelula(-1);
    ultimo = primeiro;
}
void inserir(int x) { ... }
int remover() { ... }
void mostrar() { ... }

```

```

int remover(){
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *tmp = primeiro;
    primeiro = primeiro->prox;
    int elemento = primeiro->elemento;
    tmp->prox = NULL;
    free(tmp);      tmp = NULL;
    return elemento;
}

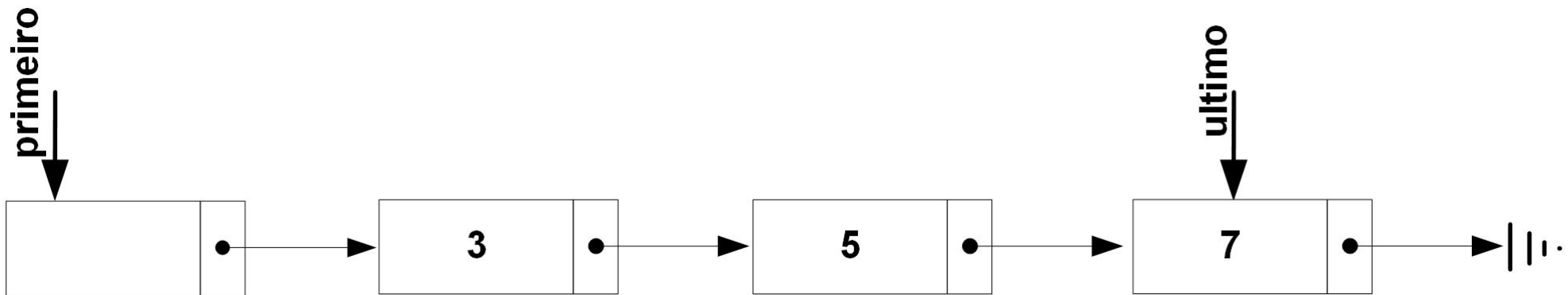
```



## Fila Flexível

- Exercício: O método remover apresentado remove fisicamente o nó cabeça e faz com que a célula do três seja a cabeça. Como alterar nosso remover para que ele remova fisicamente a célula do três ? **(FAZER AGORA)**

Dica: Execute seu método na fila abaixo



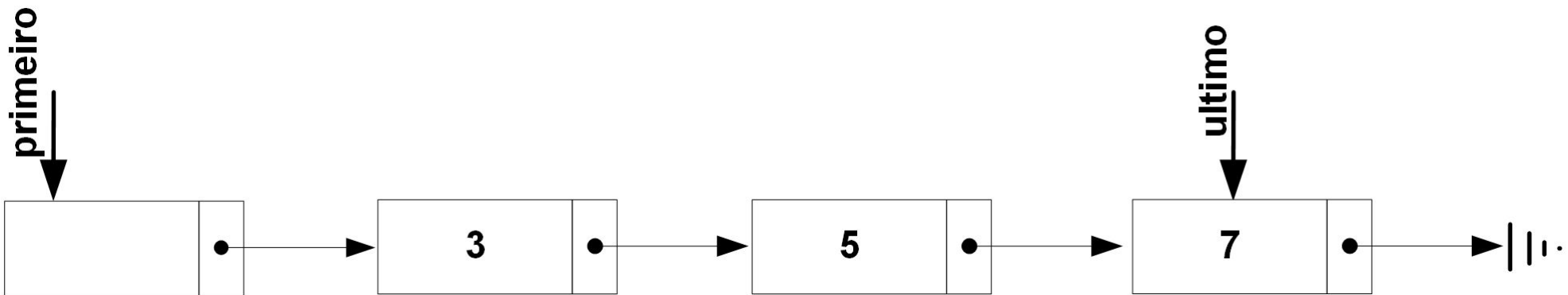


## Fila Flexível

- Exercício: O método remover apresentado e faz com que a célula do três seja a cabeça para que ele remova fisicamente a célula do

Dica: Execute seu método na fila abaixo

```
int remover(){  
    if (primeiro == ultimo)  
        errx(1, "Erro!");  
    Celula *tmp = primeiro->prox;  
    primeiro = primeiro->prox->prox;  
    int elemento = tmp->elemento;  
    tmp->prox = NULL;  
    free(tmp);    tmp = NULL;  
    return elemento;  
}
```



## Fila Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserir(int x) { ... }  
int remover() { ... }  
void mostrar() { ... }
```

```
void mostrar() {  
    // Exercício: Implemente este método  
}
```

# Fila Flexível

- Exercício: Seja nossa Fila, faça um método que retorne o maior elemento contido na mesma (**FAZER AGORA**)

## Fila Flexível

- Exercício: Seja nossa Fila, faça um método que retorne o maior elemento contido na mesma (**FAZER AGORA**)

```
int maior() {  
    int maior = - 1;  
    if (primeiro == ultimo) {  
        errx(1, "Erro!");  
    } else {  
        maior = primeiro->prox->elemento;  
        Celula *i = primeiro->prox->prox;  
        while (i != NULL){  
            if (i->elemento > maior) {  
                maior = i->elemento;  
            }  
            i = i->prox;  
        }  
        return maior;  
    }  
}
```

# Fila Flexível

- Exercício: Seja nossa Fila, faça um método para mostrar o terceiro elemento supondo que o mesmo existe (**FAZER AGORA**)

## Fila Flexível

- Exercício: Seja nossa Fila, faça um método para mostrar o terceiro elemento supondo que o mesmo existe (**FAZER AGORA**)

```
int mostrarTerceiroElemento(){  
    return (primeiro->prox->prox->prox->elemento);  
}
```

# Fila Flexível

- Exercício: Seja nossa Fila, faça um método que soma o conteúdo dos elementos contidos na mesma

## Fila Flexível

- Exercício: Seja nossa Fila, faça um método que soma o conteúdo dos elementos contidos na mesma

```
int somar() {  
    int elemento = 0;  
    for (Celula *i = primeiro->prox; i != NULL; i = i->prox) {  
        elemento += i->elemento;  
    }  
    return elemento;  
}
```



# Fila Flexível

- Exercício: Seja nossa Fila, faça um método que inverta a ordem dos seus elementos

## Fila Flexível

- Exercício: Seja nossa Fila, faça um método que inverta a ordem dos seus elementos

```
void inverter () {  
    Celula *fim = ultimo;  
    while (primeiro != fim){  
        Celula* nova = novaCelula (primeiro->prox->elemento);  
        nova->prox = fim->prox;  
        fim->prox = nova;  
        Celula tmp = primeiro->prox;  
        primeiro->prox = tmp->prox;        free(tmp);  
        nova = tmp = tmp.prox = null;  
        if (ultimo == fim) { ultimo = ultimo->prox; }  
    }  
    fim = NULL;  
}
```

# Fila Flexível

- Exercício: Seja nossa Fila, faça um método recursivo para contar o número de elementos pares AND múltiplos de cinco contidos na fila

## Fila Flexível

- Exercício: Seja nossa Fila, faça um método recursivo para contar o número de elementos pares AND múltiplos de cinco contidos na fila

```
int contar() { return contar(primeiro->prox); }  
int contar(Celula *i){  
    int elemento;  
    if (i == NULL){  
        elemento = 0;  
    } else if (i->elemento % 2 == 0 && i->elemento % 5 == 0){  
        elemento = 1 + contar(i->prox);  
    } else {  
        elemento = contar(i->prox);  
    }  
    return elemento;  
}
```

## Fila Flexível

- Exercício: Seja nossa Fila, mostre graficamente a execução do código abaixo supondo que a fila contém 5, 10, 15, 20 e 25, elementoectivamente

```
void metodoDoidao () {  
    Celula *fim = ultimo;  
    while (primeiro != fim) {  
        ultimo->prox = novaCelula (primeiro->prox->elemento);  
        Celula *tmp = primeiro;  
        primeiro = primeiro->prox;  
        tmp->prox = NULL;  
        free(tmp);    tmp = NULL;  
        ultimo = ultimo->prox;  
    }  
    fim = NULL;  
}
```

- Alocação dinâmica
- Pilha flexível
- Fila flexível
- **Lista simples flexível (lista simplesmente encadeada)**
- Lista dupla flexível (lista duplamente encadeada)

- Alocação dinâmica
- Pilha flexível
- Fila flexível
- Lista simples flexível (lista simplesmente encadeada)
- **Lista dupla flexível (lista duplamente encadeada)**

# Lista Simples Flexível

- Tem os atributos primeiro e início e os métodos abaixo:
  - Inserir no início
  - Inserir no fim
  - Inserir
  - Remover no início
  - Remover no fim
  - Remover



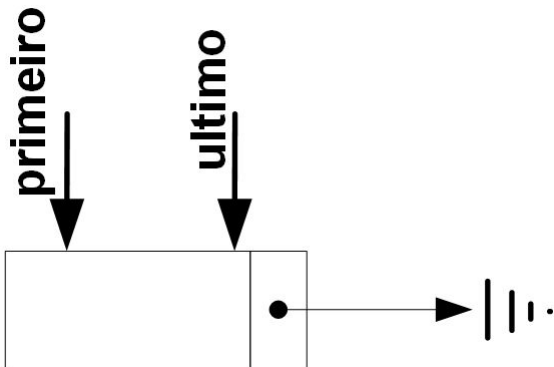
# Lista Simples Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```

# Lista Simples Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}
```

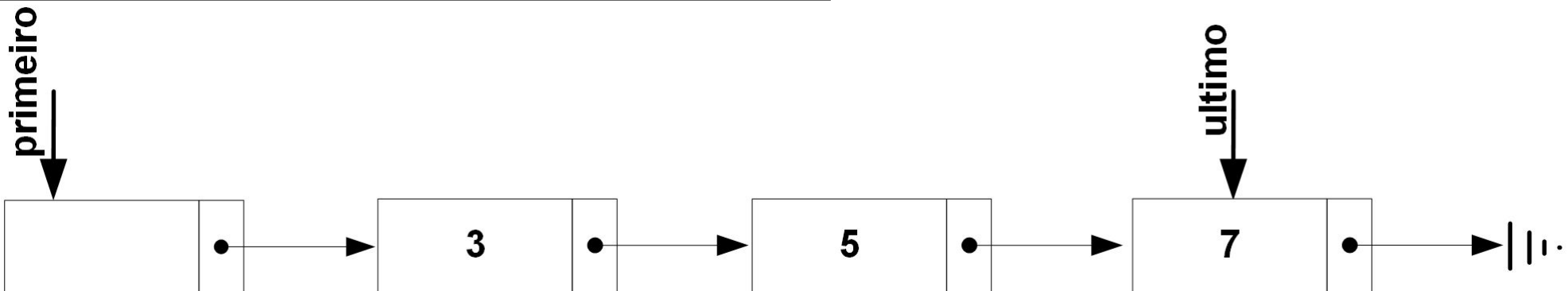
```
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```



# Lista Simples Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```

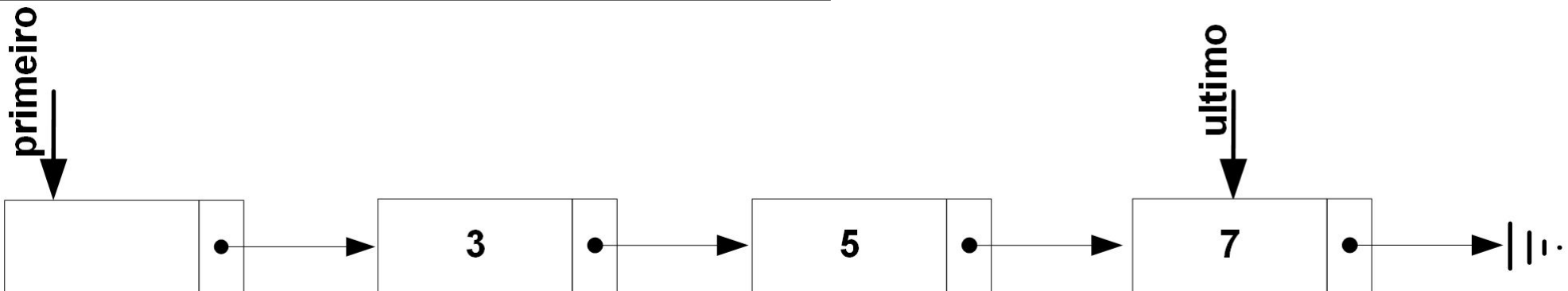
Iguais aos métodos da fila



# Lista Simples Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```

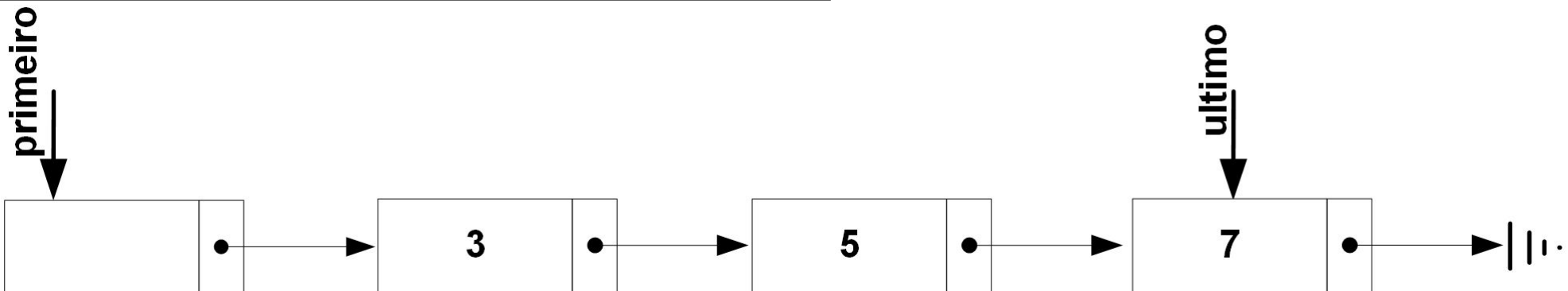
Igual aos da fila/pilha



## Lista Simples Flexível

```
Celula *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```

Assim, ...



## Lista Simples Flexível

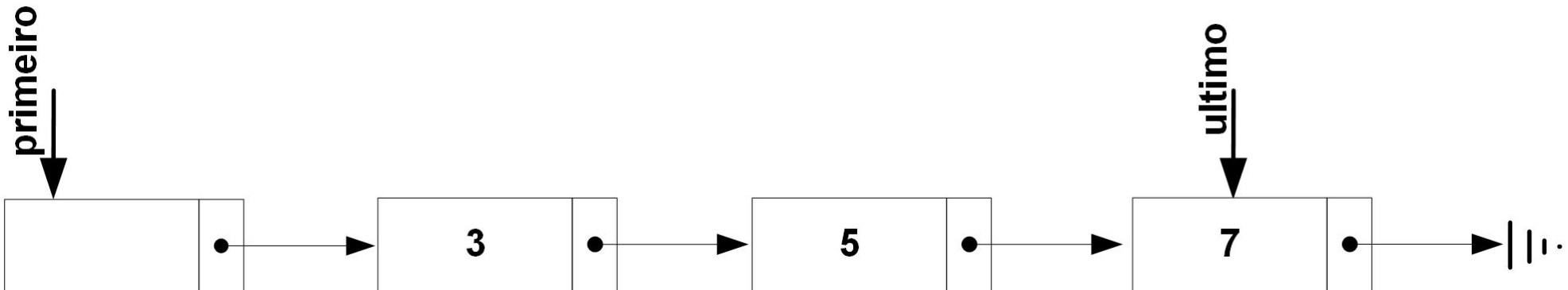
```
void inserirInicio(int x) { ... }
```

```
int removerFim() { ... }
```

```
void inserir(int x, int pos) { ... }
```

```
int remover(int pos) { ... }
```

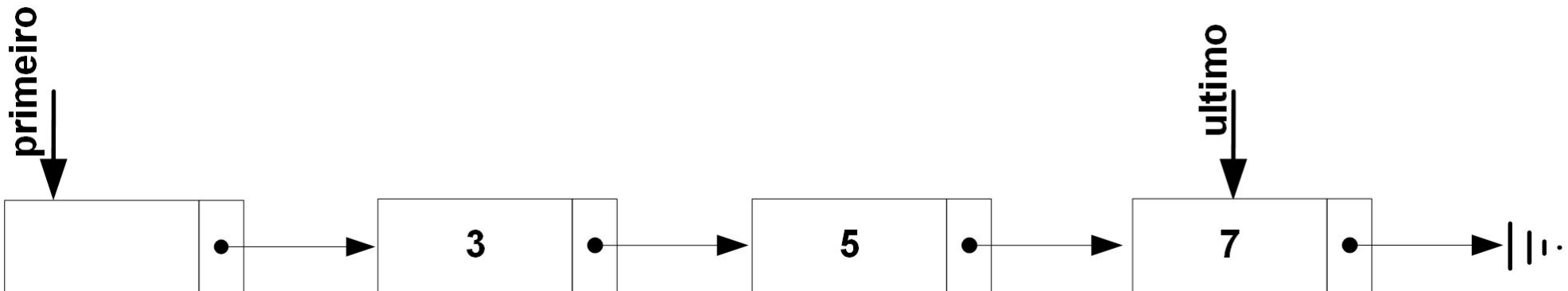
Assim, ...



# Lista Simples Flexível

```
...  
void inserirInicio(int x) { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }
```

Assim, ...



# Lista Simples Flexível

...

```
void inserirInicio(int x) { ... }
```

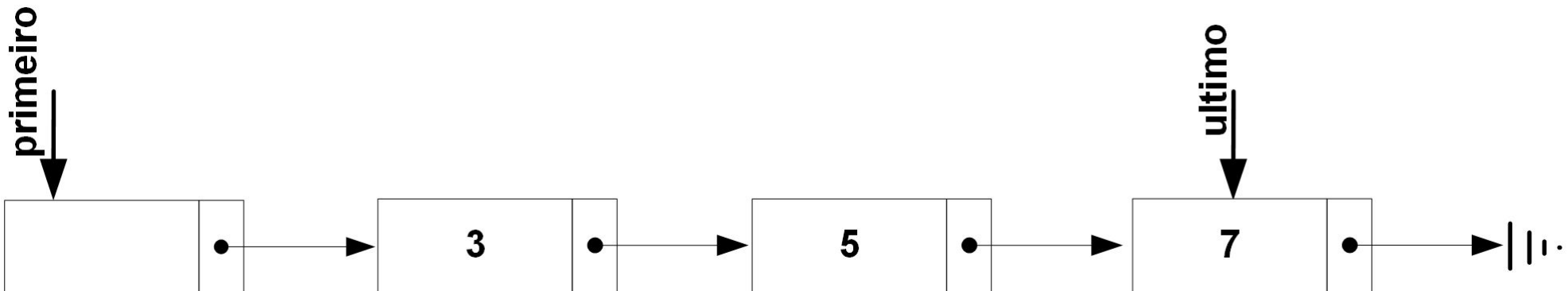
```
int removerFim() { ... }
```

```
void inserir(int x, int pos) { ... }
```

```
int remover(int pos) { ... }
```

```
//inserirInicio(1)
```

```
void inserirInicio(int x) {  
    Celula *tmp = novaCelula(x);  
    tmp->prox = primeiro->prox;  
    primeiro->prox = tmp;  
    if (primeiro == ultimo) {  
        ultimo = tmp;  
    }  
    tmp = NULL;  
}
```

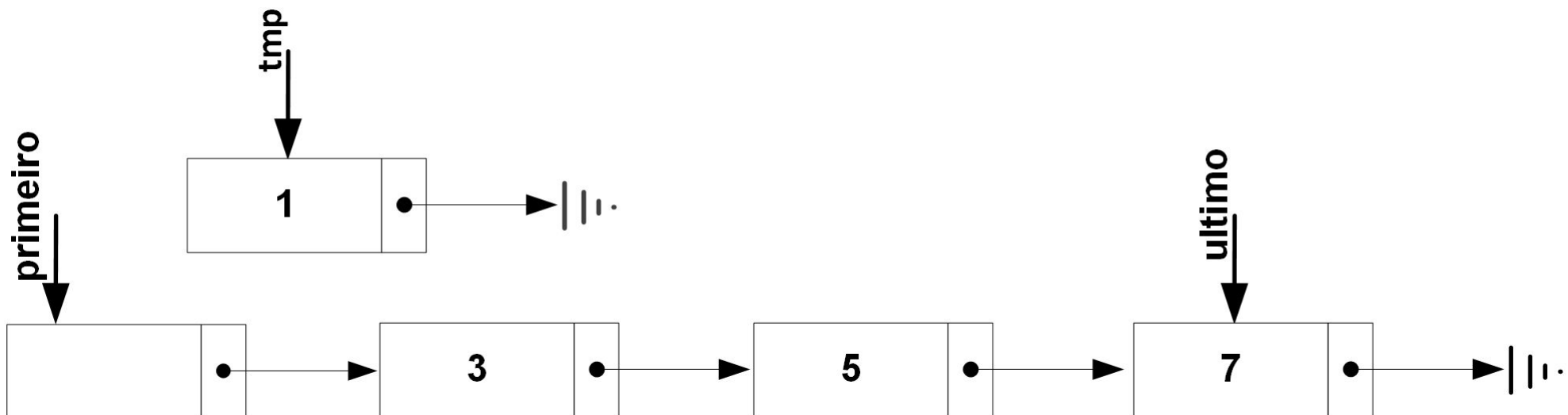




# Lista Simples Flexível

```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

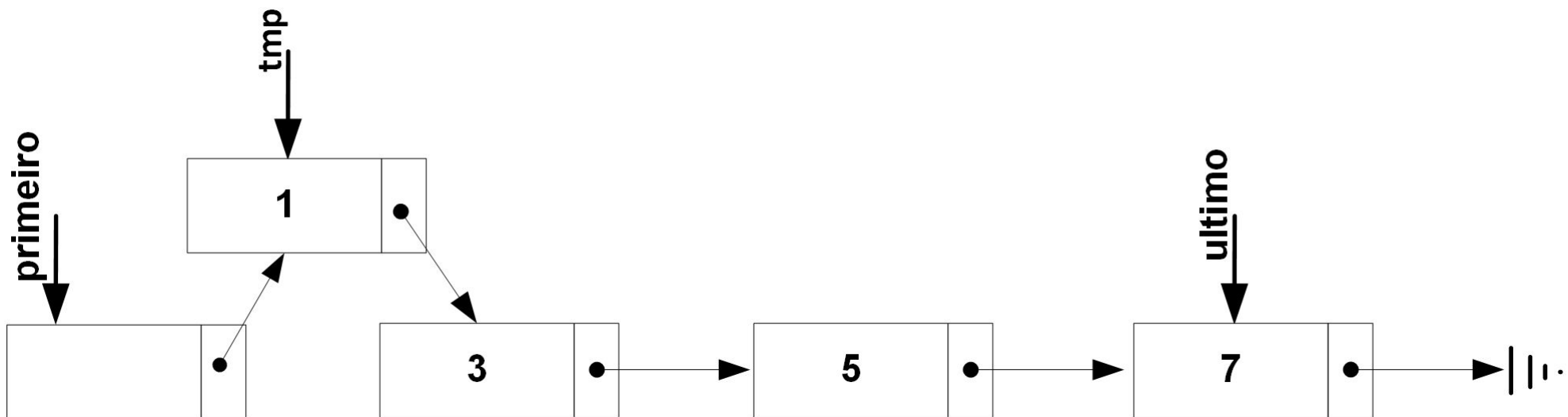
```
//inserirInicio(1)
void inserirInicio(int x) {
    Celula *tmp = novaCelula(x);
    tmp->prox = primeiro->prox;
    primeiro->prox = tmp;
    if (primeiro == ultimo) {
        ultimo = tmp;
    }
    tmp = NULL;
}
```



# Lista Simples Flexível

```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

```
//inserirInicio(1)
void inserirInicio(int x) {
    Celula *tmp = novaCelula(x);
    tmp->prox = primeiro->prox;
    primeiro->prox = tmp;
    if (primeiro == ultimo) {
        ultimo = tmp;
    }
    tmp = NULL;
}
```

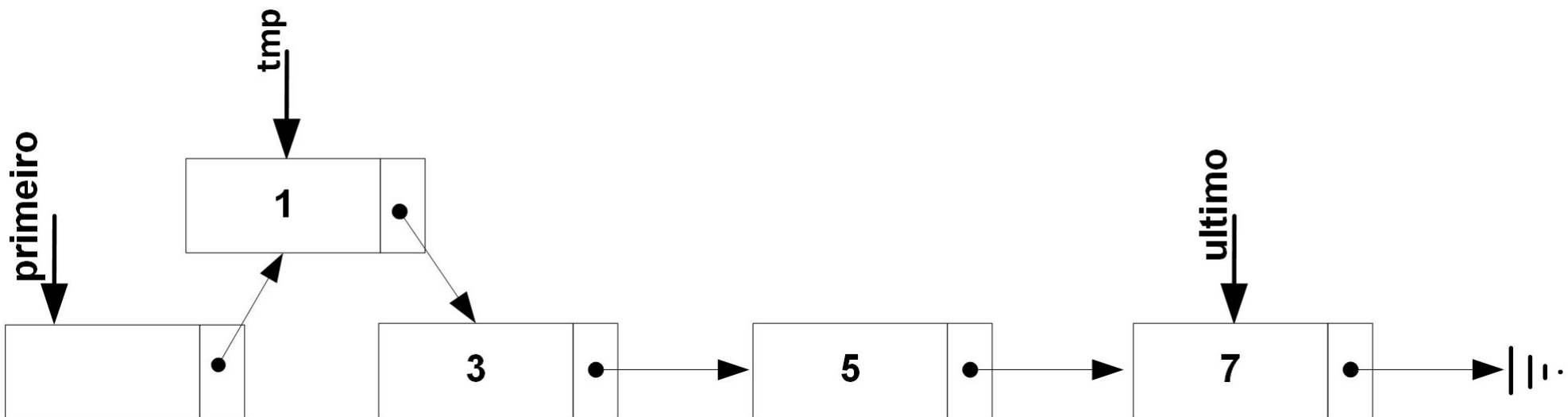


# Lista Simples Flexível

```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

```
//inserirInicio(1)
```

```
void inserirInicio(int x) {
    Celula *tmp = novaCelula(x);
    tmp->prox = primeiro->prox;
    primeiro->prox = tmp;
    if (primeiro == ultimo) {
        ultimo = tmp;
    }
    tmp = NULL;
}
```

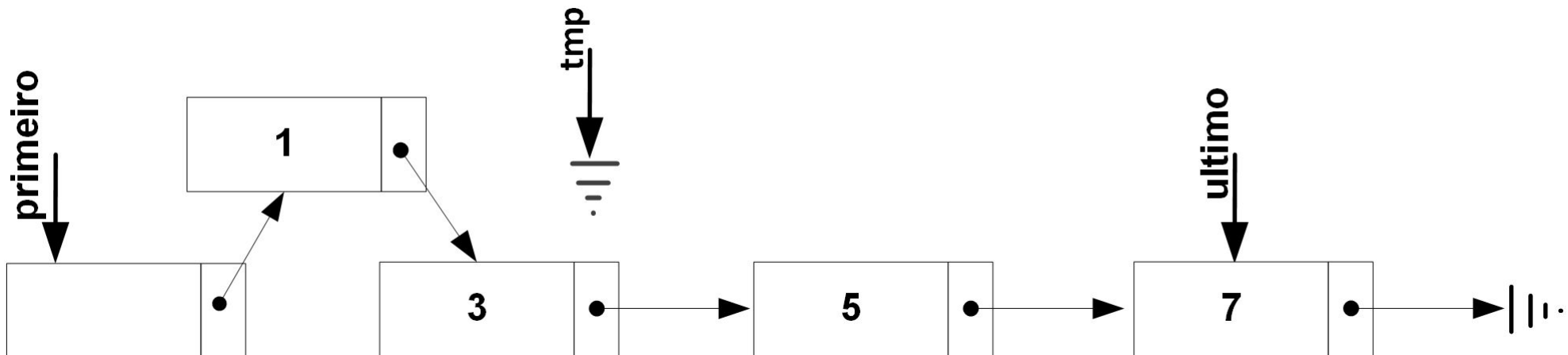


# Lista Simples Flexível

```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

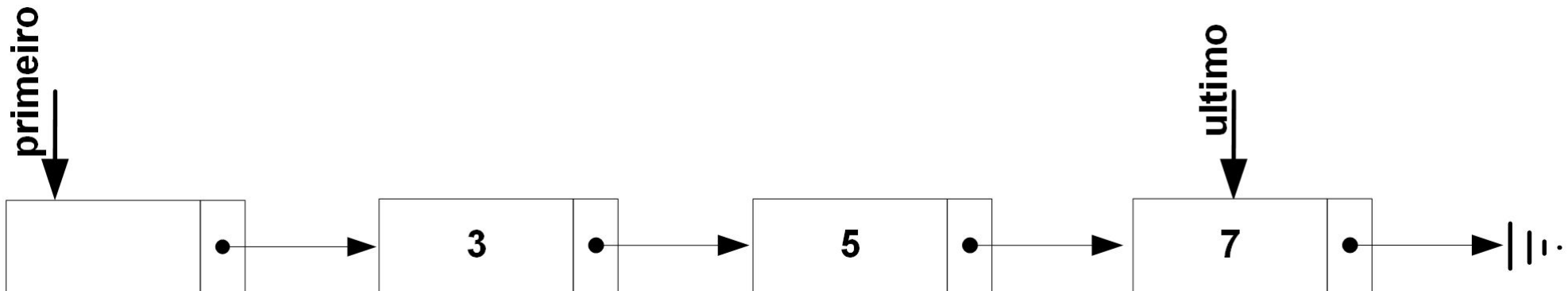
**//inserirInicio(1)**

```
void inserirInicio(int x) {
    Celula *tmp = novaCelula(x);
    tmp->prox = primeiro->prox;
    primeiro->prox = tmp;
    if (primeiro == ultimo) {
        ultimo = tmp;
    }
    tmp = NULL;
}
```



# Lista Simples Flexível

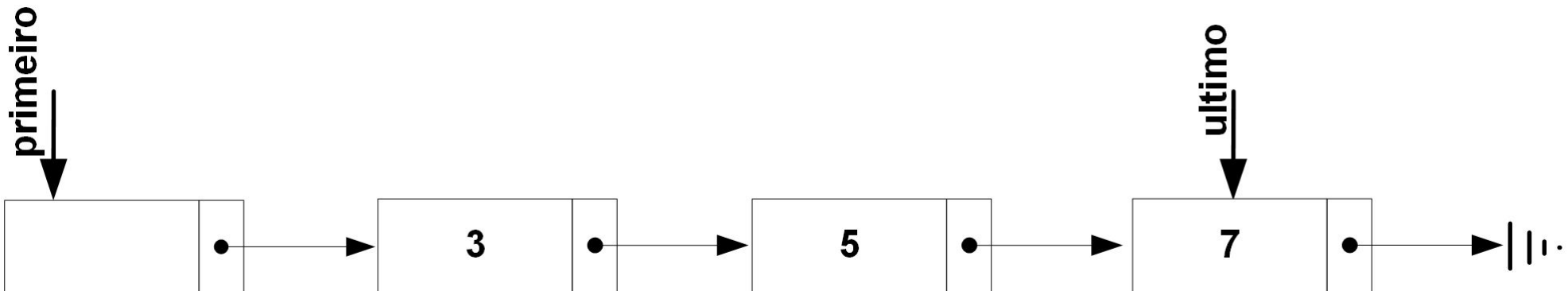
...

**void** inserirInicio(**int** x) { ... }**int** removerFim() { ... }**void** inserir(**int** x, **int** pos) { ... }**int** remover(**int** pos) { ... }

# Lista Simples Flexível

```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

```
int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;    free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}
```



# Lista Simples Flexível

...

```

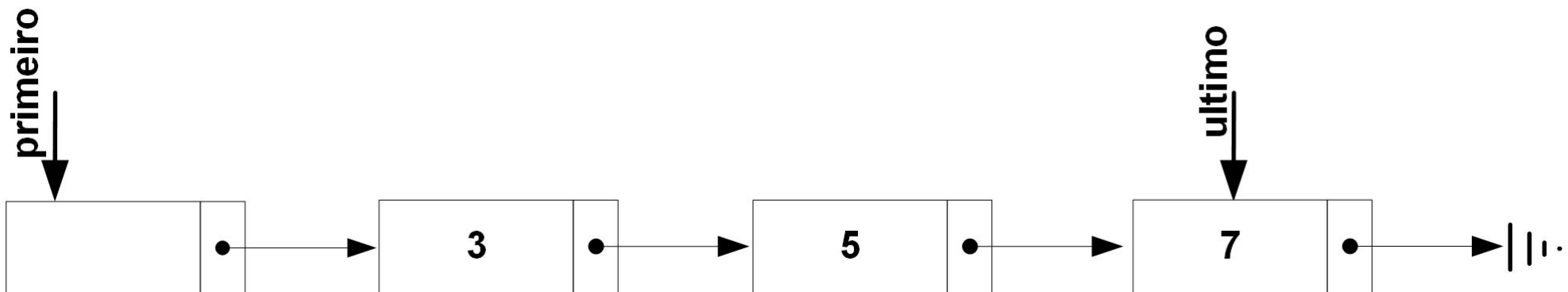
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }

```

```

int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;    free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}

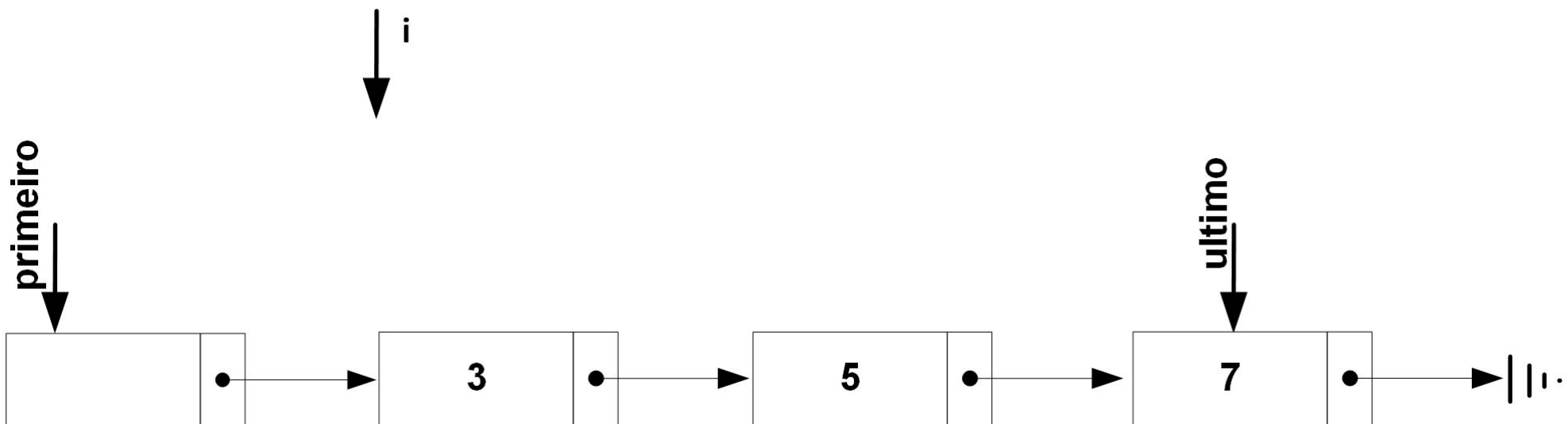
```



# Lista Simples Flexível

```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

```
int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;    free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}
```

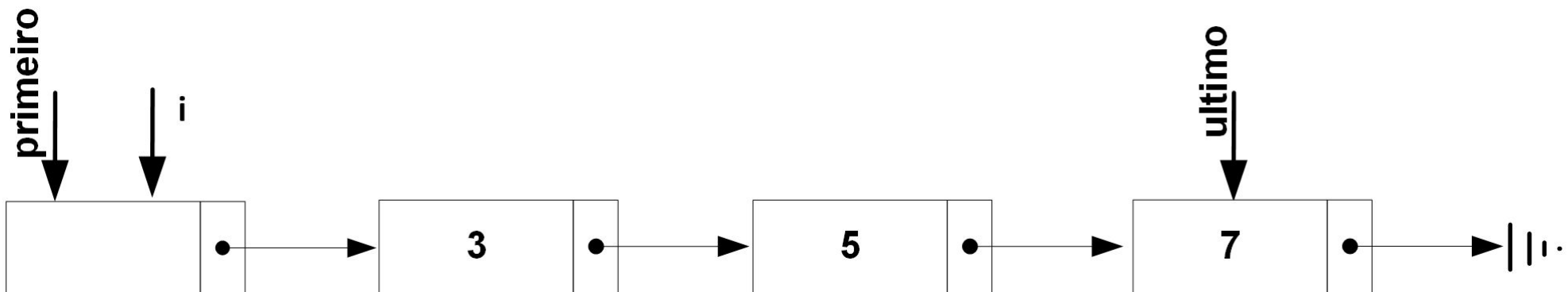




# Lista Simples Flexível

```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

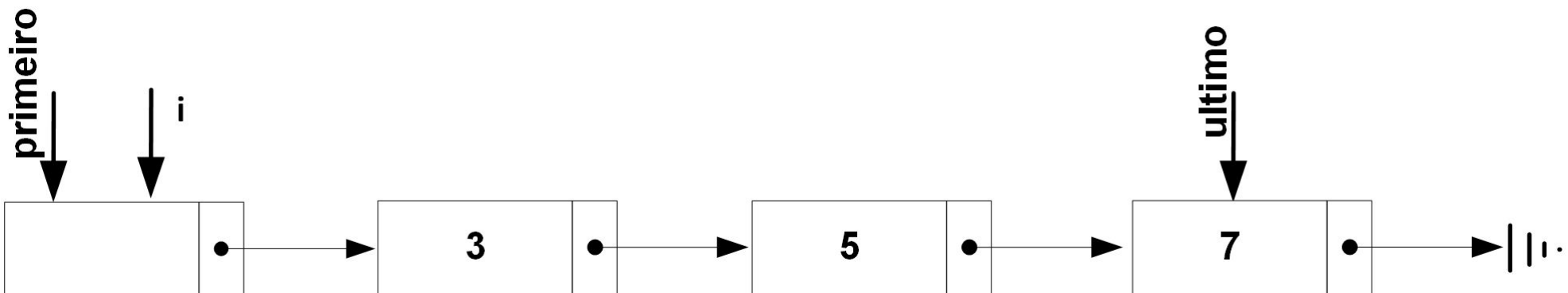
```
int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for (i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;    free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}
```



# Lista Simples Flexível

```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

```
int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;      free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;      true
}
```



# Lista Simples Flexível

...

```

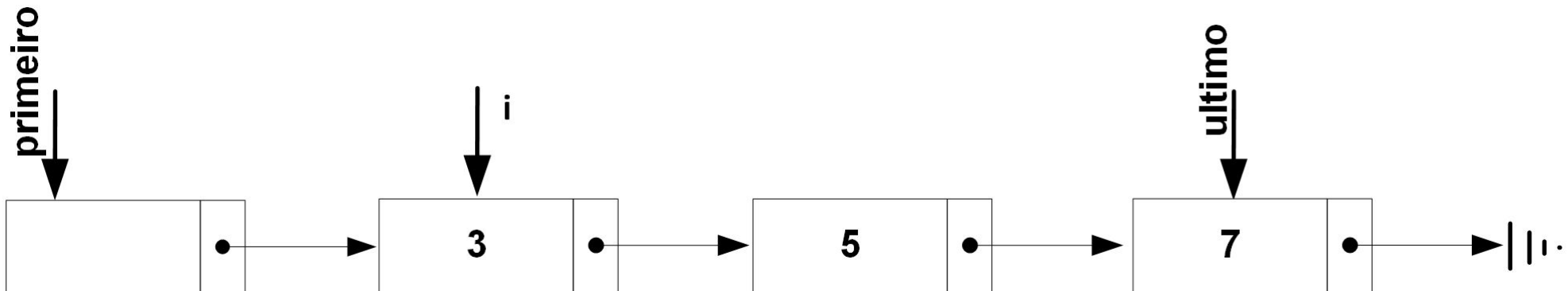
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }

```

```

int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;    free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}

```



# Lista Simples Flexível

...

```

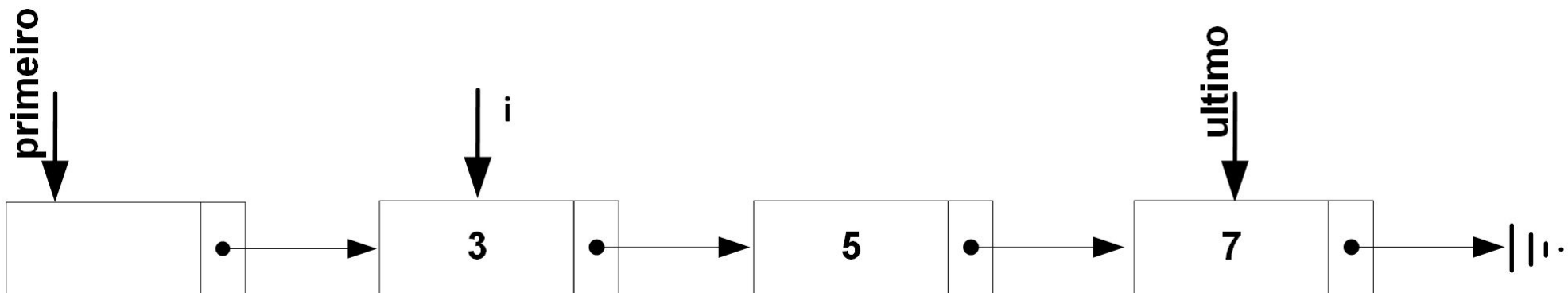
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }

```

```

int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;      free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}

```



# Lista Simples Flexível

...

```

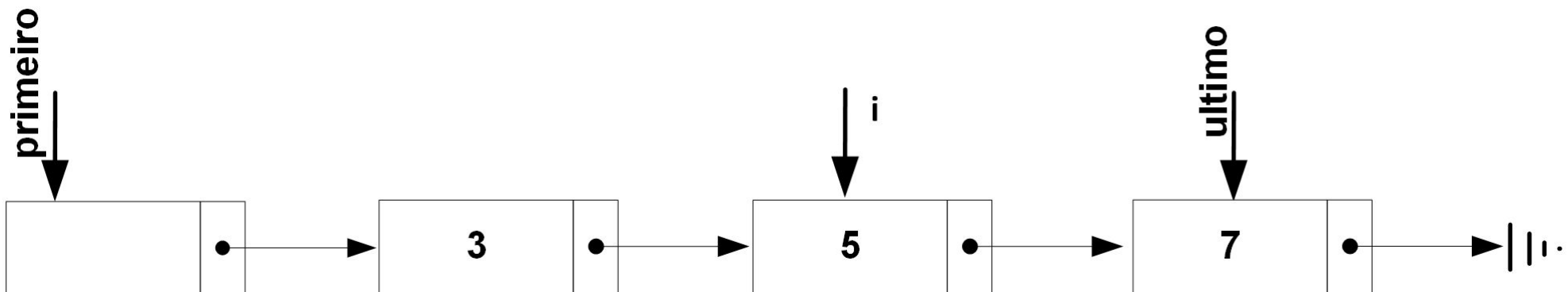
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }

```

```

int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;    free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}

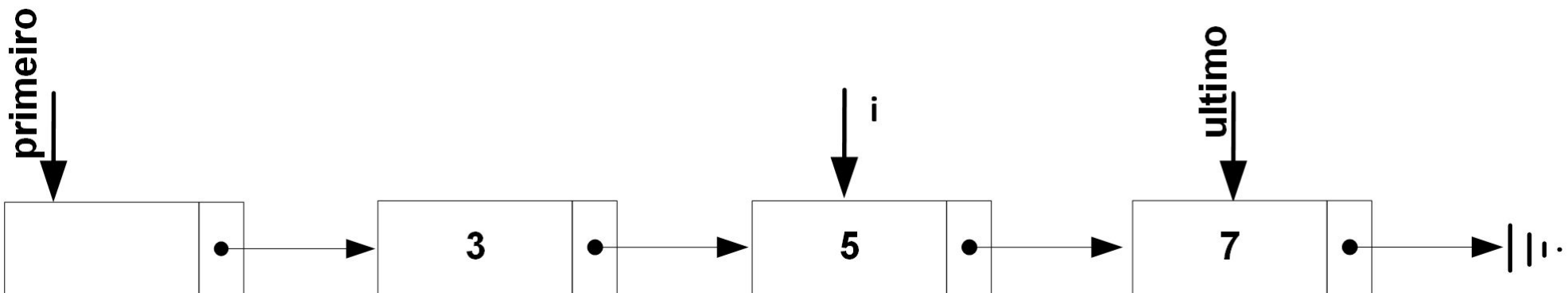
```



# Lista Simples Flexível

```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

```
int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;      free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}
```



# Lista Simples Flexível

...

```

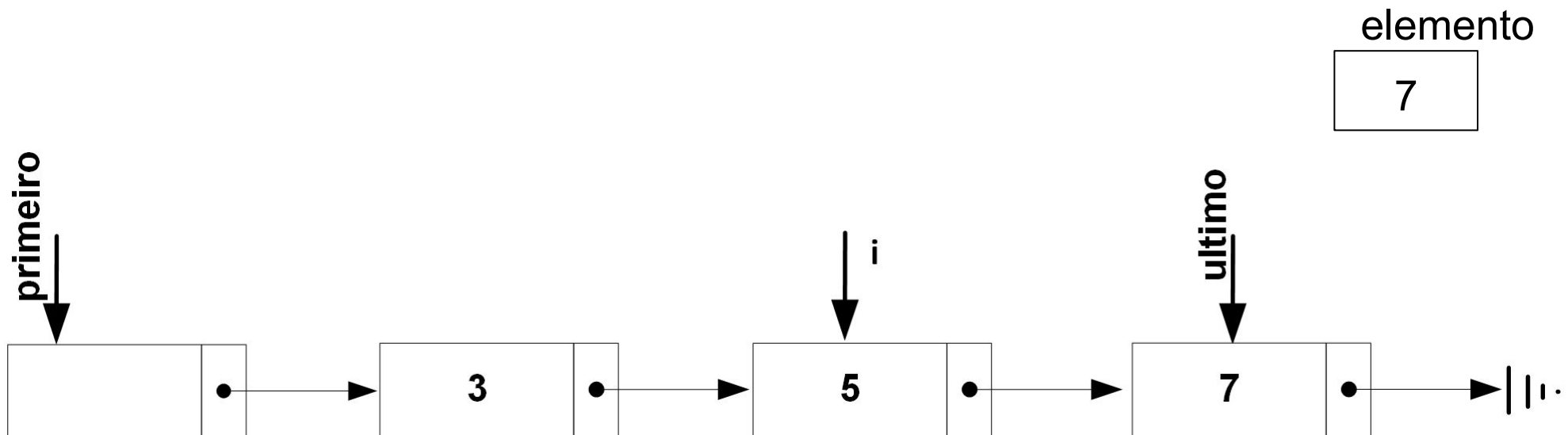
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }

```

```

int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;    free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}

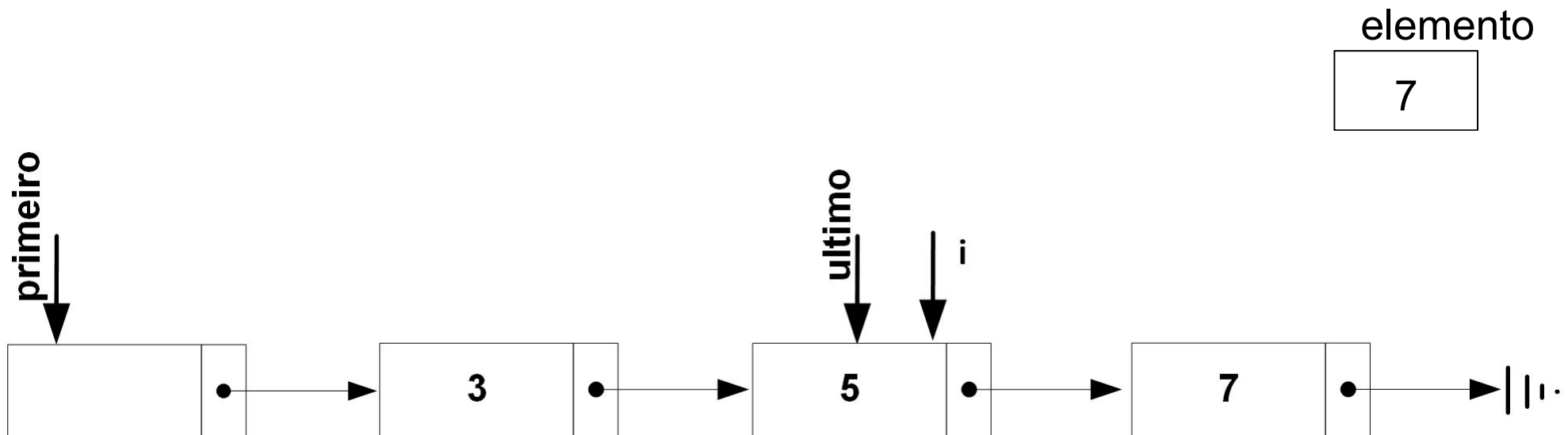
```



# Lista Simples Flexível

```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

```
int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;    free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}
```





# Lista Simples Flexível

...

```

void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }

```

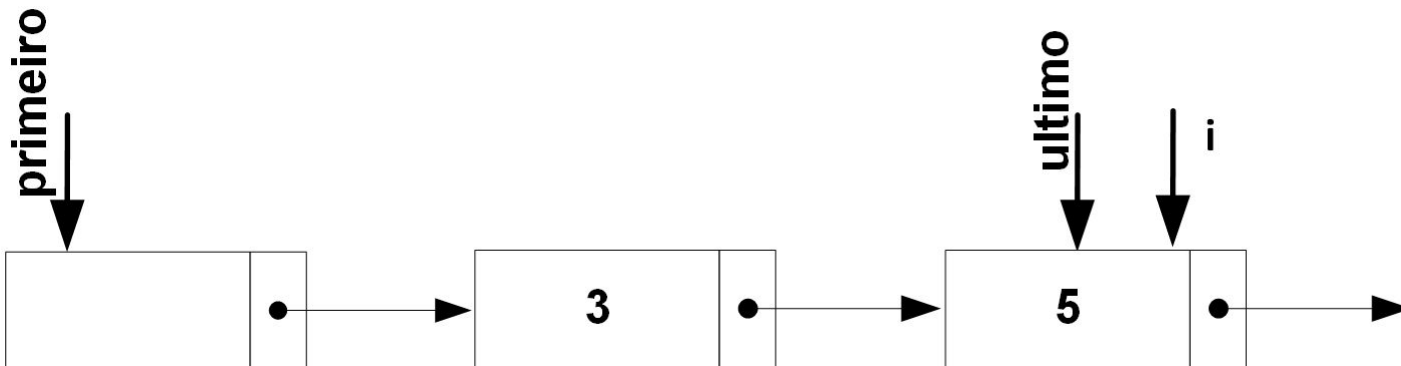
```

int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;    free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}

```

elemento

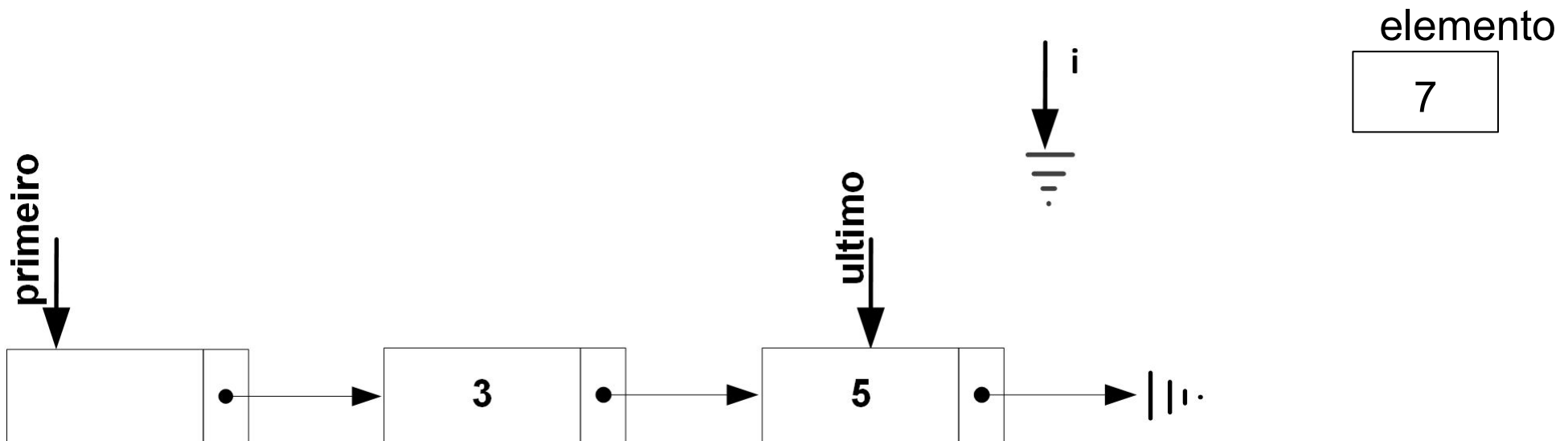
7



# Lista Simples Flexível

```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

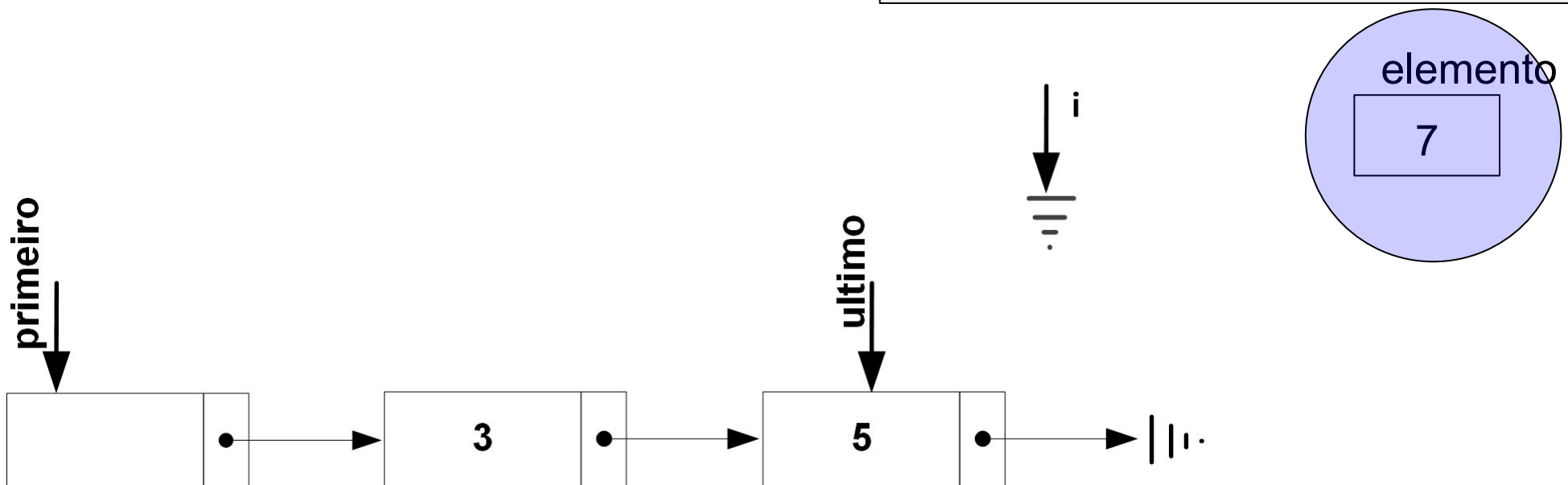
```
int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;      free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}
```



# Lista Simples Flexível

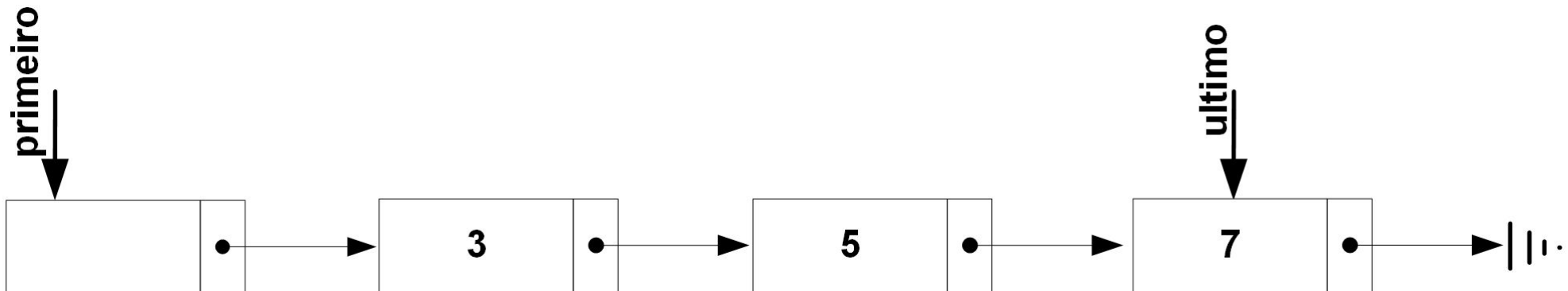
```
...
void inserirInicio(int x) { ... }
int removerFim() { ... }
void inserir(int x, int pos) { ... }
int remover(int pos) { ... }
```

```
int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i;    free(ultimo->prox);
    i = ultimo->prox = NULL;
    return elemento;
}
```



# Lista Simples Flexível

```
...  
void inserirInicio(int x) { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }
```

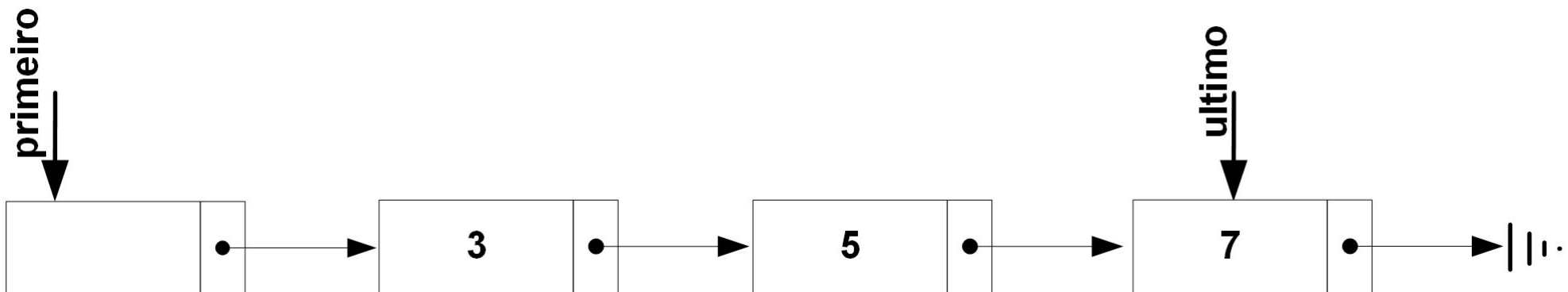


## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
  int tamanho = tamanho();
  if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
  } else if (pos == 0){      inserirInicio(x);
  } else if (pos == tamanho){ inserirFim(x);
  } else {
    Celula *i = primeiro;
    for(int j = 0; j < pos; j++, i = i->prox);
    Celula *tmp = novaCelula(x);
    tmp->prox = i->prox;
    i->prox = tmp;
    tmp = i = NULL;
  } }

```

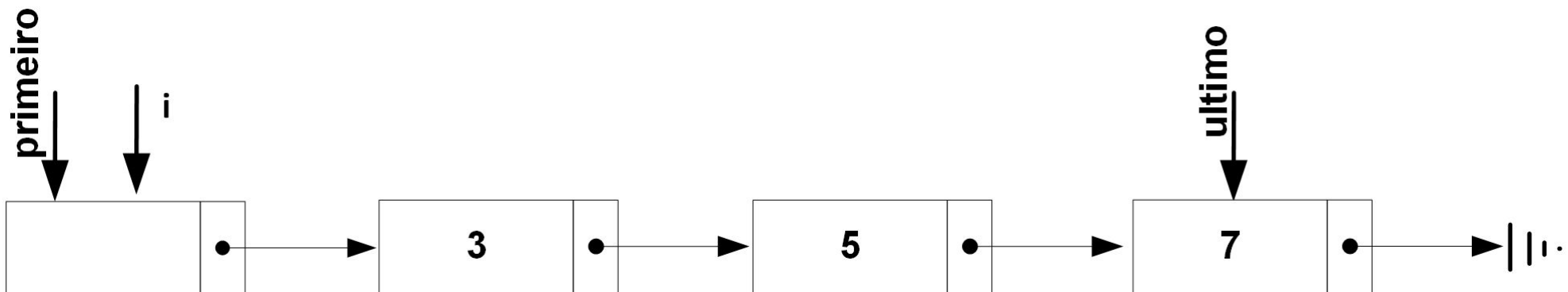


## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
  int tamanho = tamanho();
  if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
  } else if (pos == 0){      inserirInicio(x);
  } else if (pos == tamanho){ inserirFim(x);
  } else {
    Celula *i = primeiro;
    for(int j = 0; j < pos; j++, i = i->prox);
    Celula *tmp = novaCelula(x);
    tmp->prox = i->prox;
    i->prox = tmp;
    tmp = i = NULL;
  } }

```

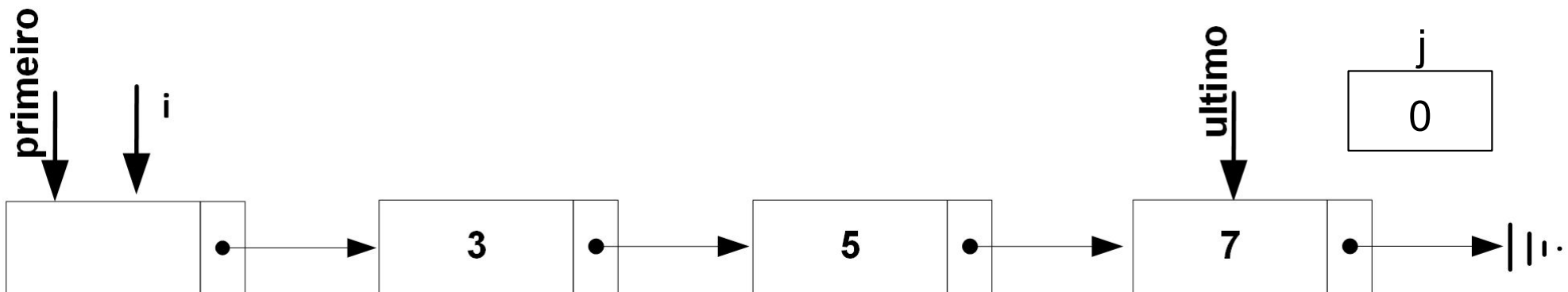


## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
    int tamanho = tamanho();
    if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
    } else if (pos == 0){      inserirInicio(x);
    } else if (pos == tamanho){ inserirFim(x);
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = novaCelula(x);
        tmp->prox = i->prox;
        i->prox = tmp;
        tmp = i = NULL;
    } }

```



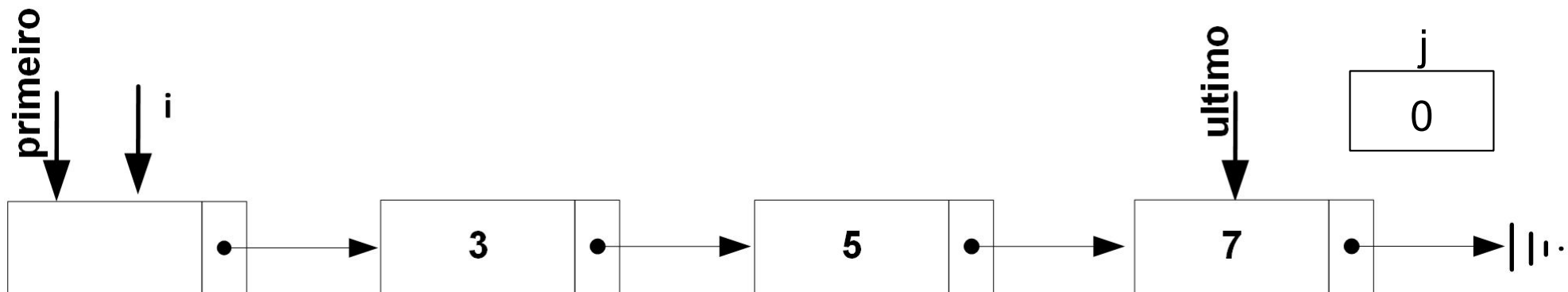
## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
  int tamanho = tamanho();
  if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
  } else if (pos == 0){      inserirInicio(x);
  } else if (pos == tamanho){ inserirFim(x);
  } else {
    Celula *i = primeiro;
    for(int j = 0; j < pos; j++, i = i->prox);
    Celula *tmp = novaCelula(x);
    tmp->prox = i->prox;
    i->prox = tmp;
    tmp = i = NULL;
  }
}

```

true



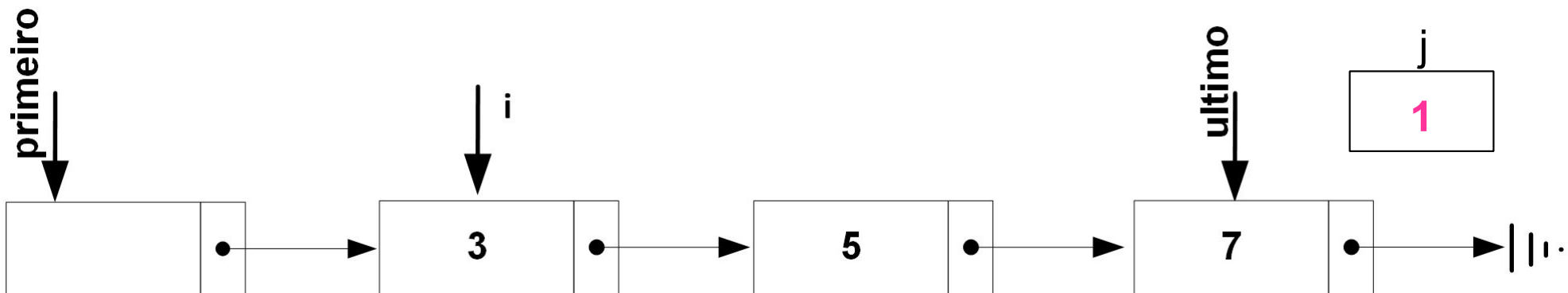


## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
    int tamanho = tamanho();
    if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
    } else if (pos == 0){      inserirInicio(x);
    } else if (pos == tamanho){ inserirFim(x);
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = novaCelula(x);
        tmp->prox = i->prox;
        i->prox = tmp;
        tmp = i = NULL;
    } }

```



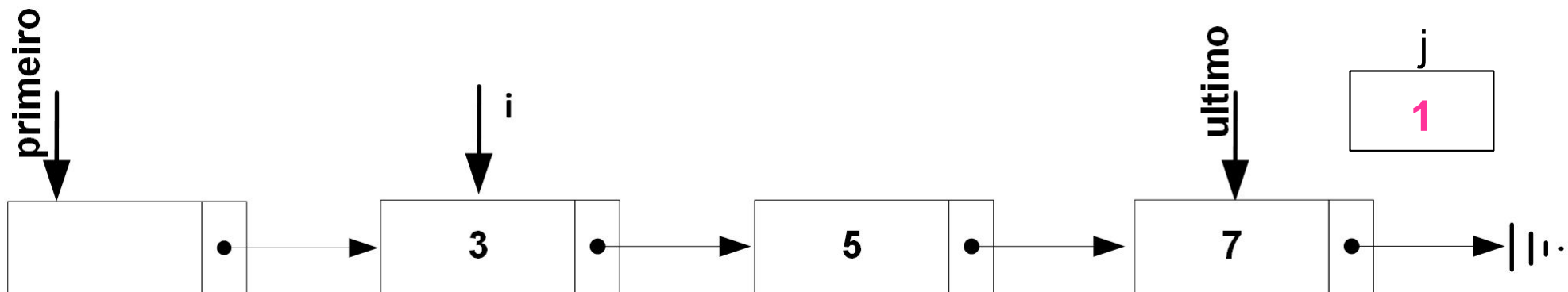
## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
    int tamanho = tamanho();
    if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
    } else if (pos == 0){                  inserirInicio(x);
    } else if (pos == tamanho){           inserirFim(x);
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = novaCelula(x);
        tmp->prox = i->prox;
        i->prox = tmp;
        tmp = i = NULL;
    }
}

```

true

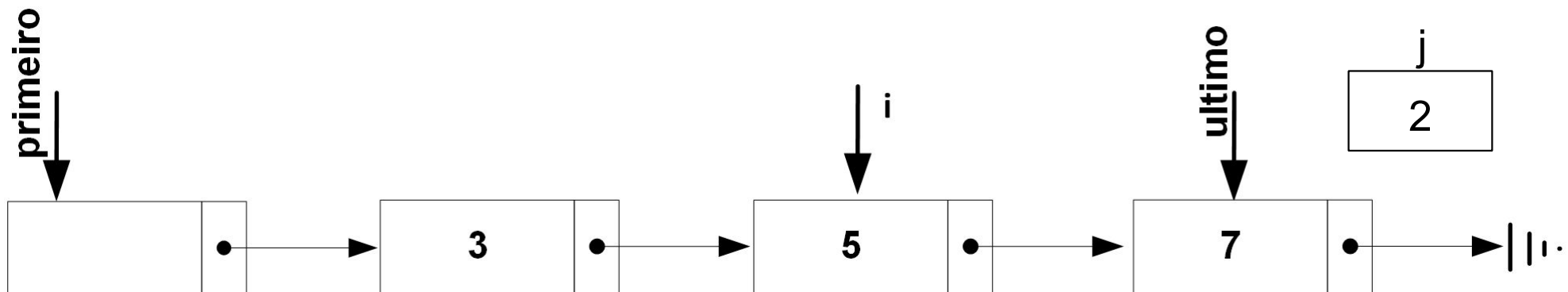


## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
  int tamanho = tamanho();
  if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
  } else if (pos == 0){      inserirInicio(x);
  } else if (pos == tamanho){ inserirFim(x);
  } else {
    Celula *i = primeiro;
    for(int j = 0; j < pos; j++, i = i->prox);
    Celula *tmp = novaCelula(x);
    tmp->prox = i->prox;
    i->prox = tmp;
    tmp = i = NULL;
  } }

```



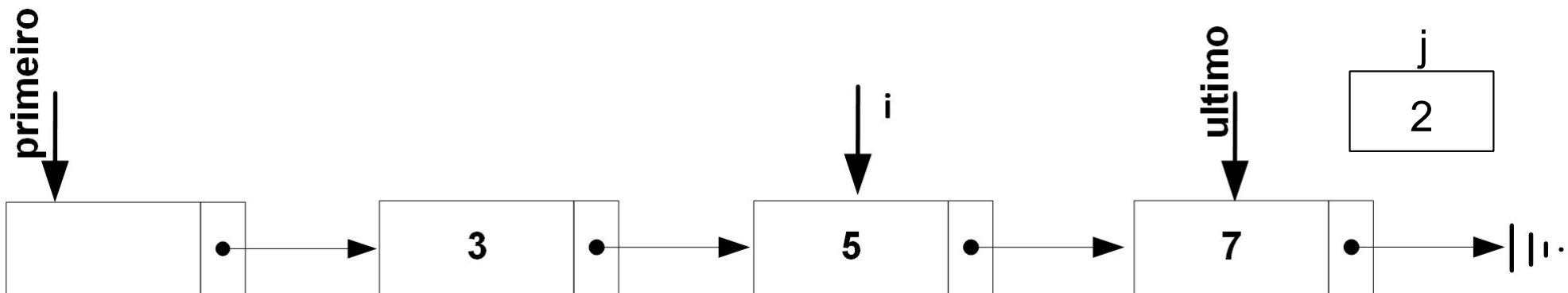
## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
  int tamanho = tamanho();
  if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
  } else if (pos == 0){      inserirInicio(x);
  } else if (pos == tamanho){ inserirFim(x);
  } else {
    Celula *i = primeiro;
    for(int j = 0; j < pos; j++, i = i->prox);
    Celula *tmp = novaCelula(x);
    tmp->prox = i->prox;
    i->prox = tmp;
    tmp = i = NULL;
  }
}

```

false

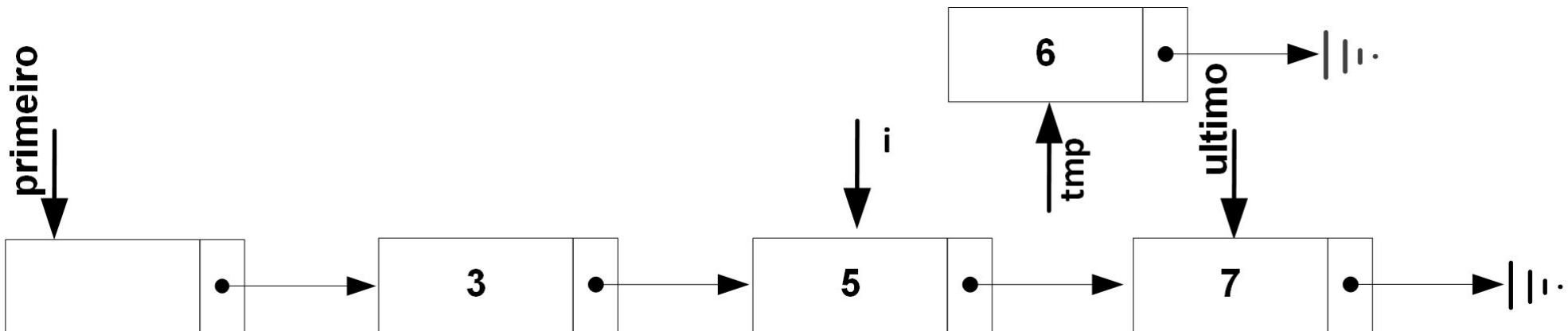


## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
    int tamanho = tamanho();
    if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
    } else if (pos == 0){      inserirInicio(x);
    } else if (pos == tamanho){ inserirFim(x);
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = novaCelula(x);
        tmp->prox = i->prox;
        i->prox = tmp;
        tmp = i = NULL;
    } }

```

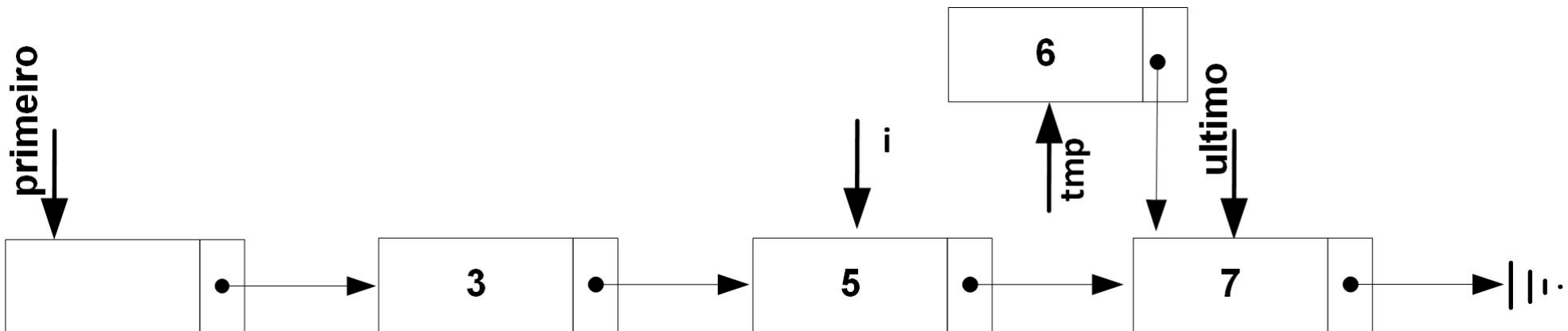


## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
    int tamanho = tamanho();
    if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
    } else if (pos == 0){      inserirInicio(x);
    } else if (pos == tamanho){ inserirFim(x);
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = novaCelula(x);
        tmp->prox = i->prox;
        i->prox = tmp;
        tmp = i = NULL;
    } }

```

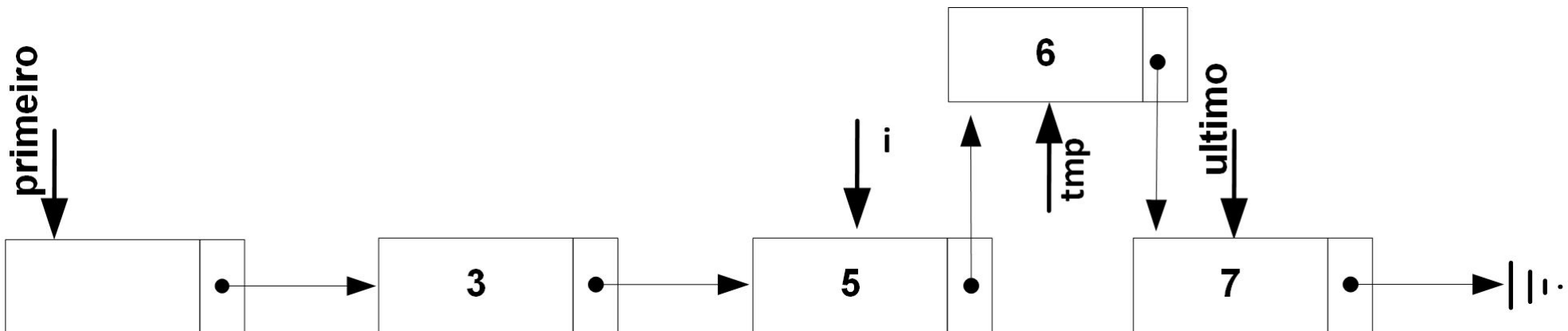


## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
  int tamanho = tamanho();
  if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
  } else if (pos == 0){      inserirInicio(x);
  } else if (pos == tamanho){ inserirFim(x);
  } else {
    Celula *i = primeiro;
    for(int j = 0; j < pos; j++, i = i->prox);
    Celula *tmp = novaCelula(x);
    tmp->prox = i->prox;
    i->prox = tmp;
    tmp = i = NULL;
  } }

```

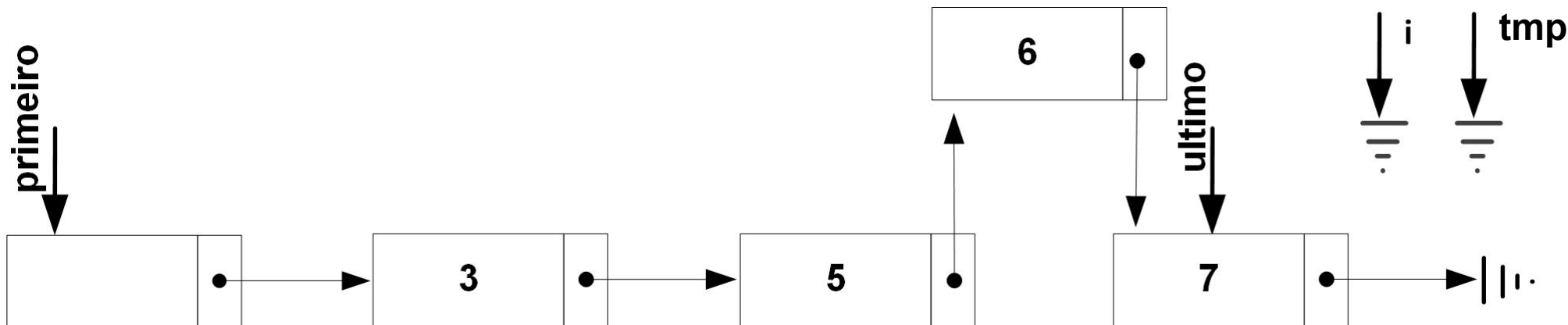


## Lista Simples Flexível

```

void inserir(int x, int pos) {           //Inserir(6, 2)
  int tamanho = tamanho();
  if (pos < 0 || pos > tamanho){ errx(1, "Erro!");
  } else if (pos == 0){      inserirInicio(x);
  } else if (pos == tamanho){ inserirFim(x);
  } else {
    Celula *i = primeiro;
    for(int j = 0; j < pos; j++, i = i->prox);
    Celula *tmp = novaCelula(x);
    tmp->prox = i->prox;
    i->prox = tmp;
    tmp = i = NULL;
  } }

```





# Lista Simples Flexível

...

**void** inserirInicio(**int** x) { ... }

**int** removerFim() { ... }

**void** inserir(**int** x, **int** pos) { ... }

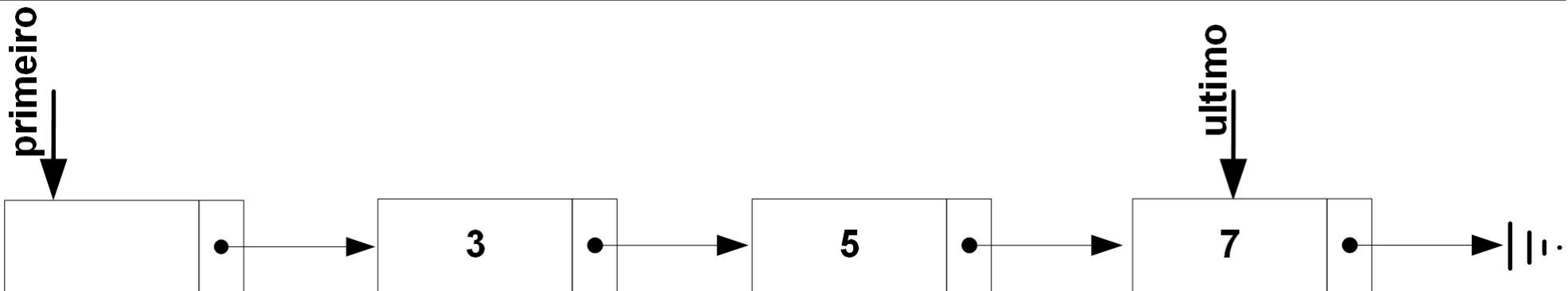
**int** remover(**int** pos) { ... }

## Lista Simples Flexível

```

int remover(int pos ) {                                //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;    i->prox = tmp->prox;
        tmp->prox = NULL;           free(tmp);           i = tmp = NULL;
    }
    return elemento;
}

```

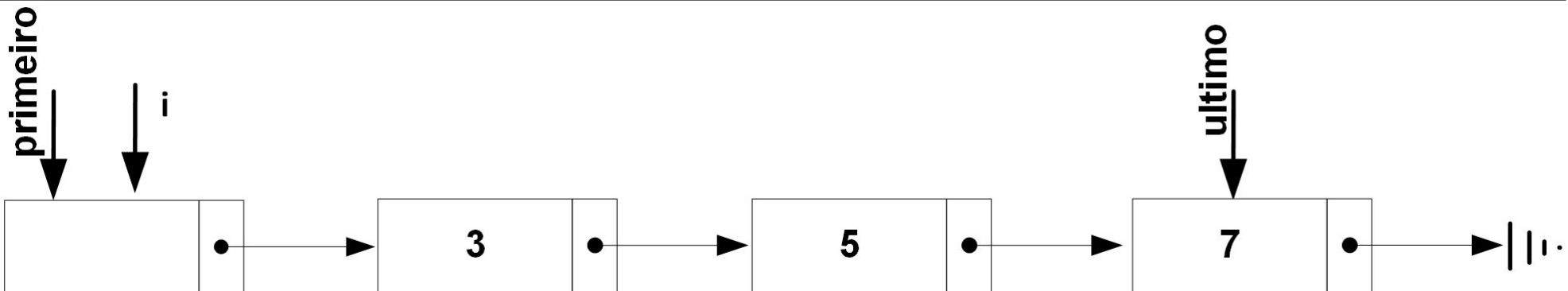


## Lista Simples Flexível

```

int remover(int pos ) {                               //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;    i->prox = tmp->prox;
        tmp->prox = NULL;           free(tmp);           i = tmp = NULL;
    }
    return elemento;
}

```

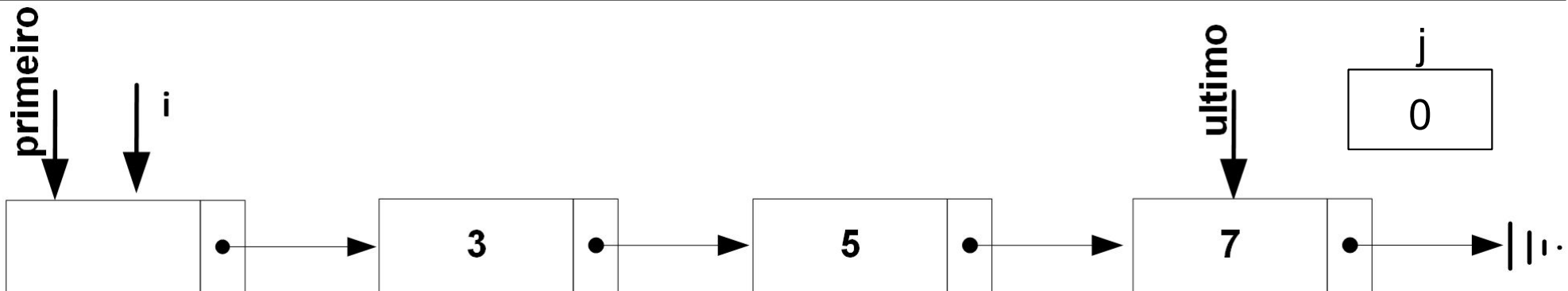


## Lista Simples Flexível

```

int remover(int pos ) {                                //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;    i->prox = tmp->prox;
        tmp->prox = NULL;           free(tmp);          i = tmp = NULL;
    }
    return elemento;
}

```

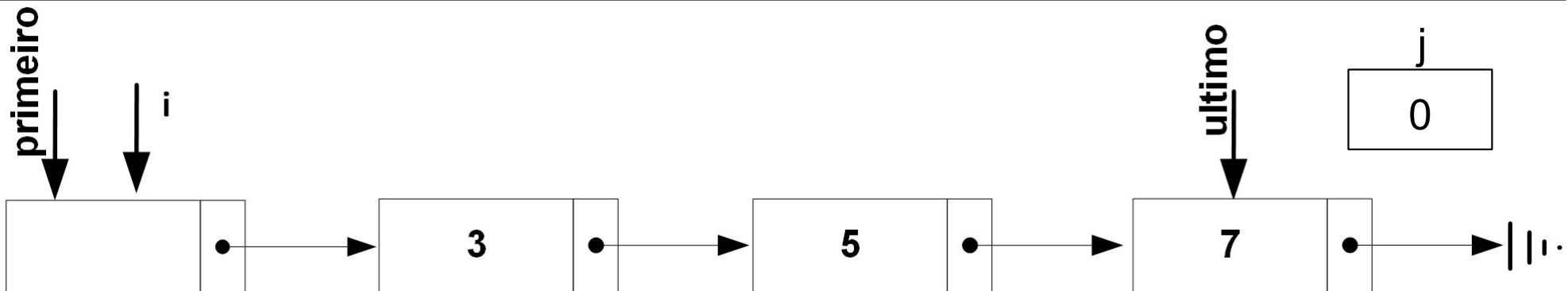


## Lista Simples Flexível

```

int remover(int pos ) {                               //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;    i->prox = tmp->prox;
        tmp->prox = NULL;           free(tmp);           i = tmp = NULL;
    }
    return elemento;
}

```

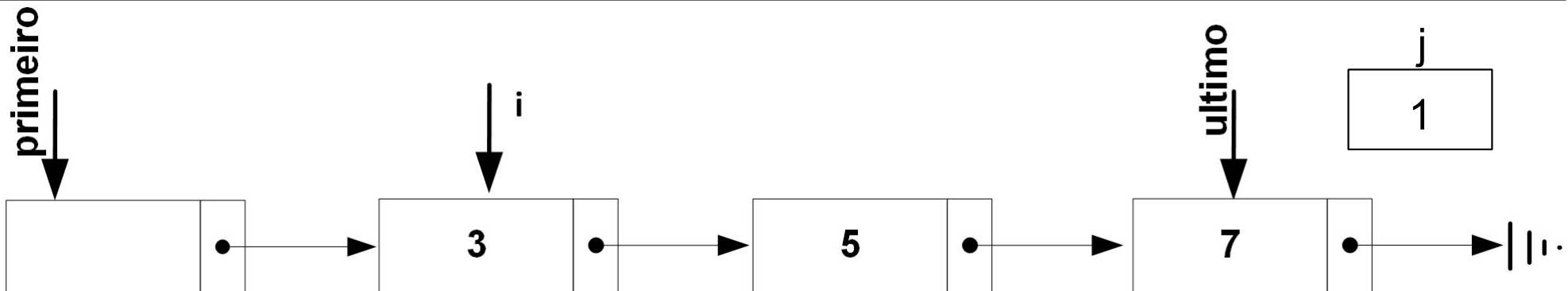


## Lista Simples Flexível

```

int remover(int pos ) {                               //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;   i->prox = tmp->prox;
        tmp->prox = NULL;          free(tmp);          i = tmp = NULL;
    }
    return elemento;
}

```

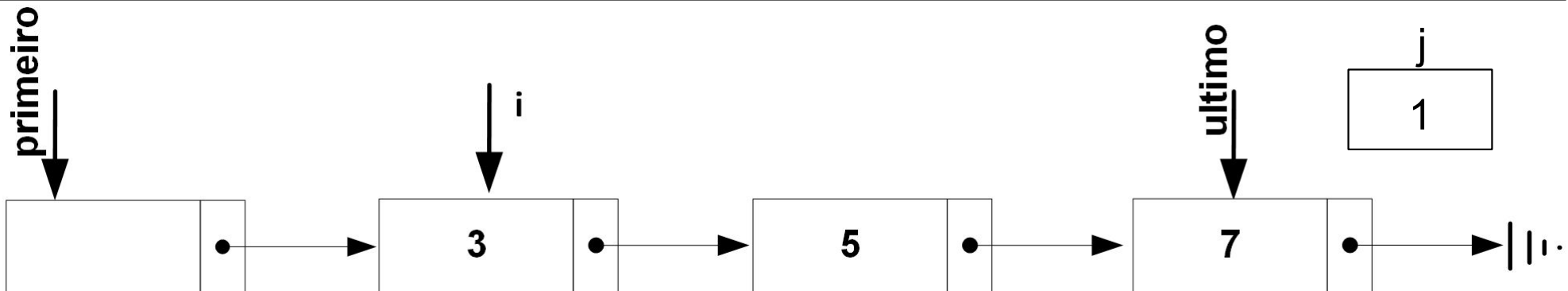


## Lista Simples Flexível

```

int remover(int pos ) {                               //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;    i->prox = tmp->prox;
        tmp->prox = NULL;           free(tmp);          i = tmp = NULL;
    }
    return elemento;
}

```

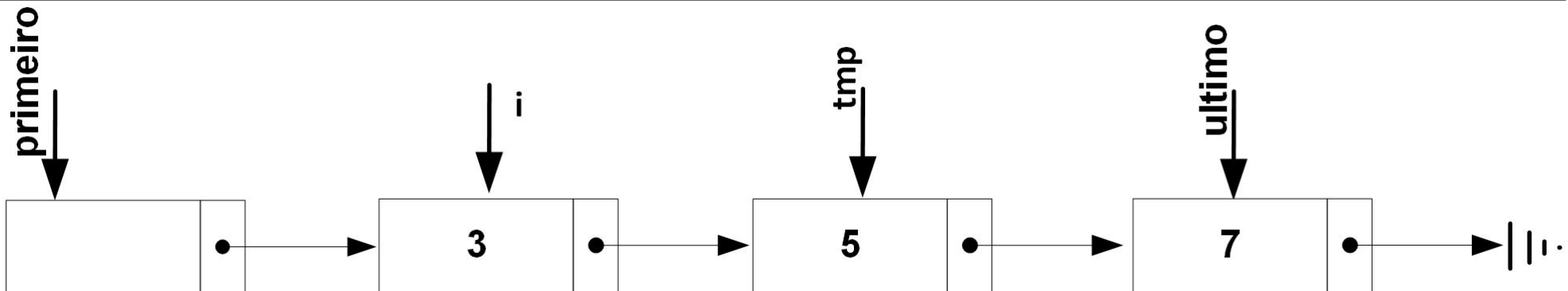


## Lista Simples Flexível

```

int remover(int pos ) {                               //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;   i->prox = tmp->prox;
        tmp->prox = NULL;          free(tmp);          i = tmp = NULL;
    }
    return elemento;
}

```





## Lista Simples Flexível

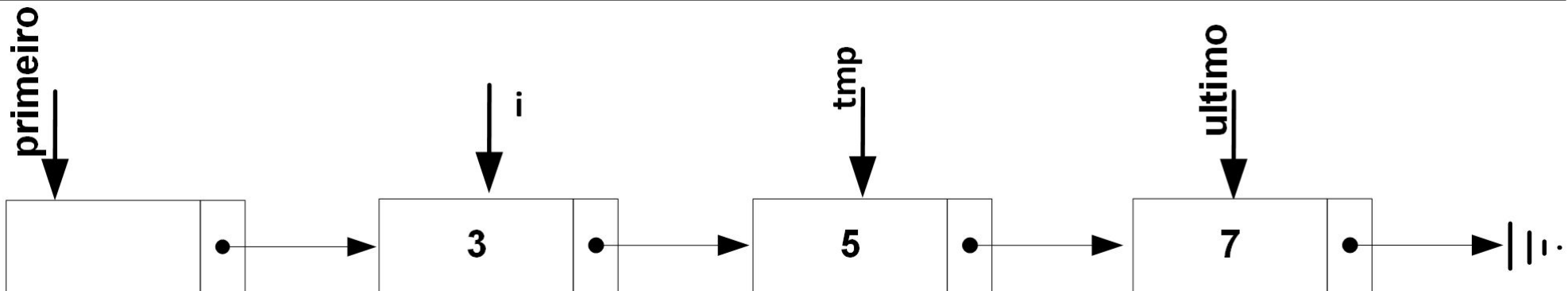
```

int remover(int pos ) {                               //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;
        i->prox = tmp->prox;
        tmp->prox = NULL;                             free(tmp);      i = tmp = NULL;
    }
    return elemento;
}

```

elemento

5



## Lista Simples Flexível

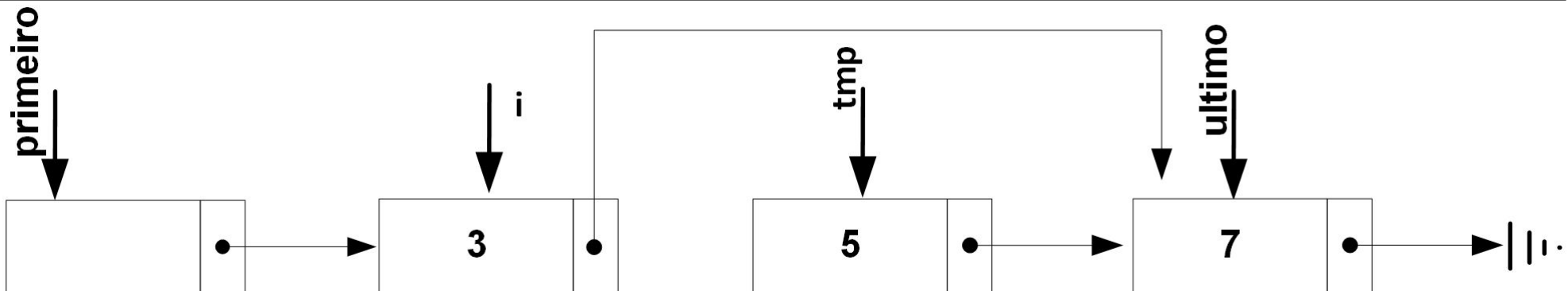
```

int remover(int pos ) {                                //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                       elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento; i->prox = tmp->prox;
        tmp->prox = NULL;                               free(tmp);           i = tmp = NULL;
    }
    return elemento;
}

```

elemento

5



## Lista Simples Flexível

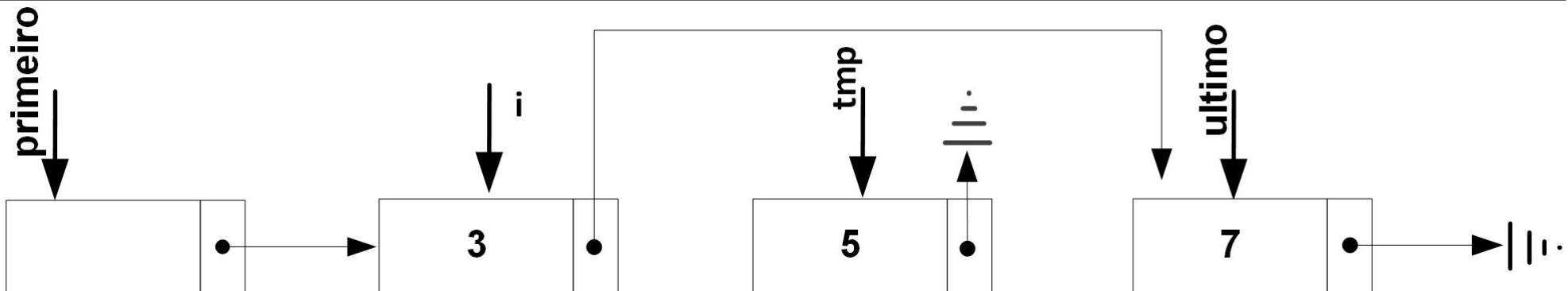
```

int remover(int pos ) {                               //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;   i->prox = tmp->prox;
        tmp->prox = NULL;          free(tmp);           i = tmp = NULL;
    }
    return elemento;
}

```

elemento

5



## Lista Simples Flexível

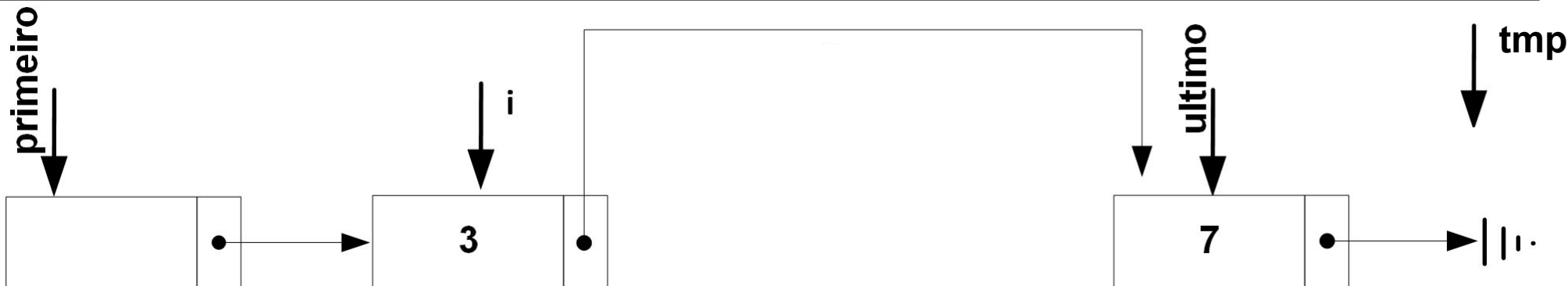
```

int remover(int pos ) {                               //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;   i->prox = tmp->prox;
        tmp->prox = NULL;          free(tmp);          i = tmp = NULL;
    }
    return elemento;
}

```

elemento

5



## Lista Simples Flexível

```

int remover(int pos ) {                               //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;   i->prox = tmp->prox;
        tmp->prox = NULL;          i = tmp = NULL;
        free(tmp);
    }
    return elemento;
}

```

elemento

5

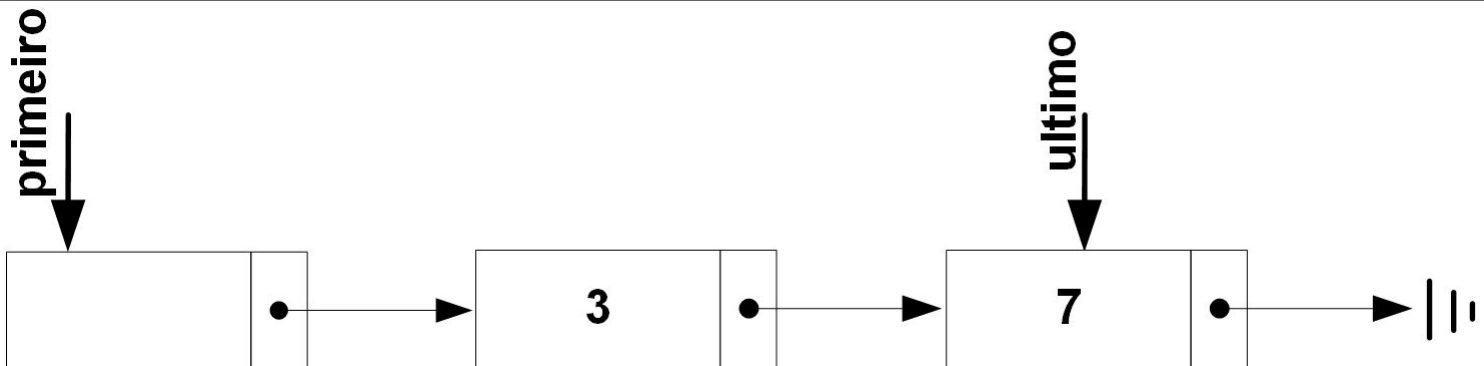
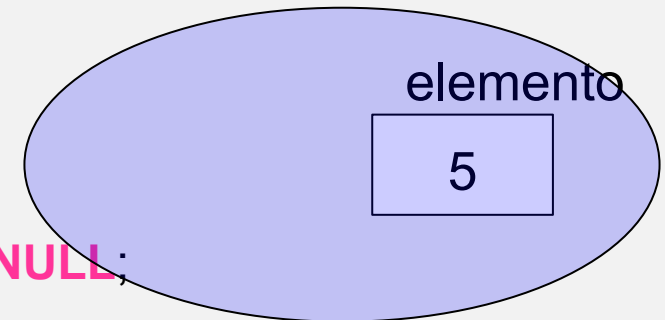


## Lista Simples Flexível

```

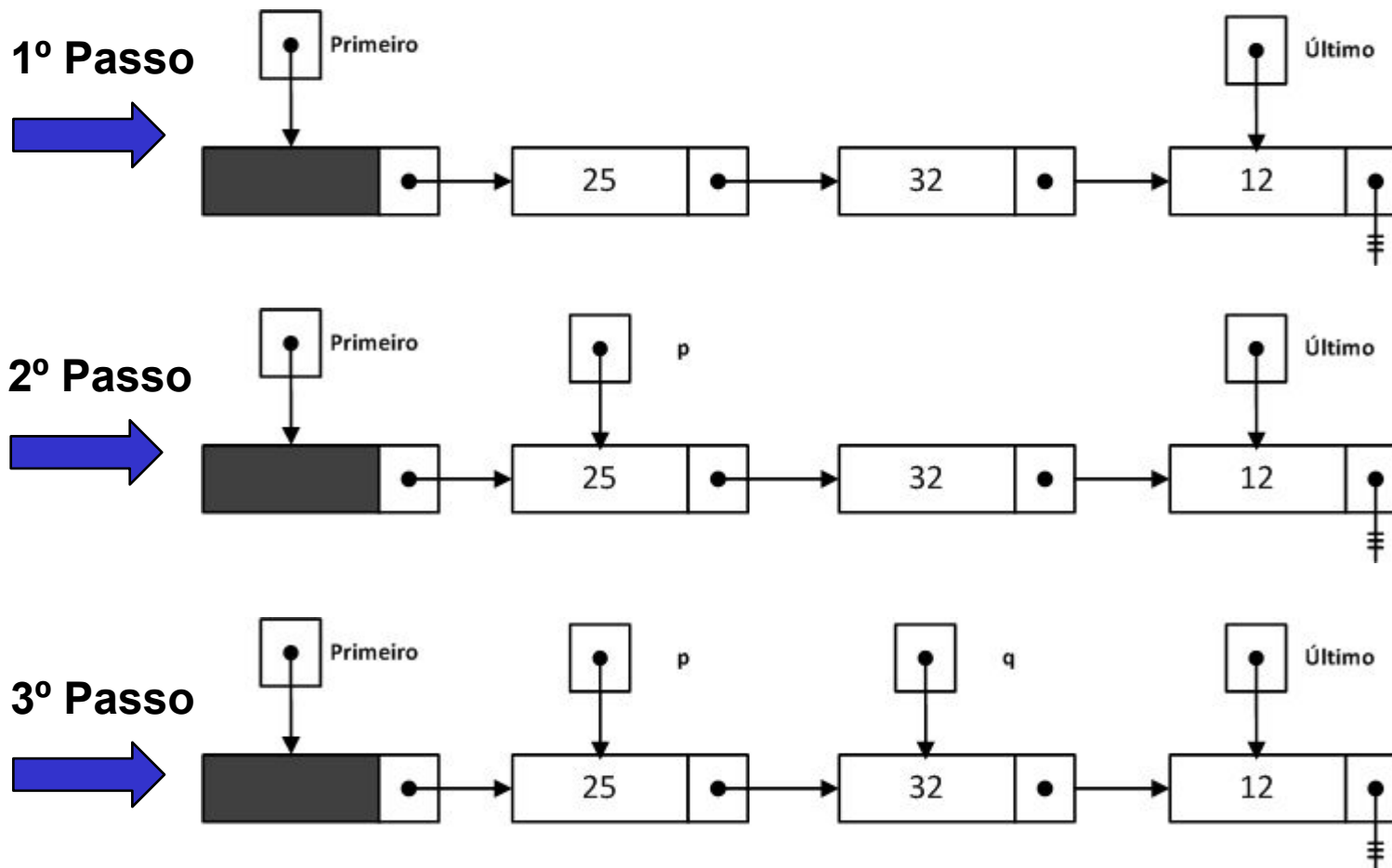
int remover(int pos ) {                                //remover(1)
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo || pos < 0 || pos >= tamanho){ errx(1, "Erro!");
    } else if (pos == 0) {                               elemento = removerInicio();
    } else if (pos == tamanho - 1){                     elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for(int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;    i->prox = tmp->prox;
        tmp->prox = NULL;           free(tmp);           i = tmp = NULL;
    }
    return elemento;
}

```



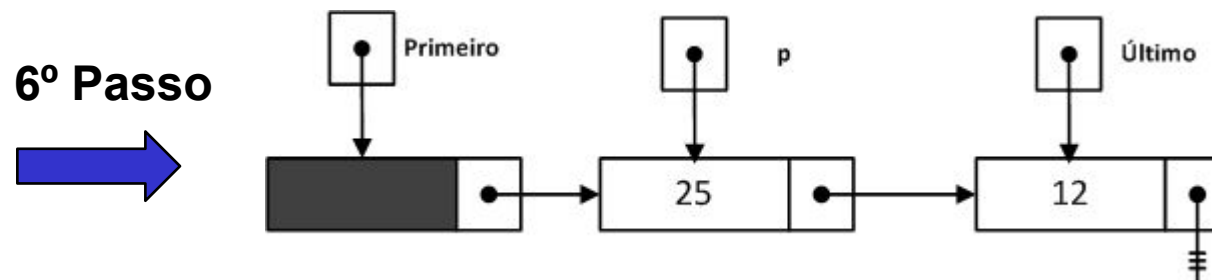
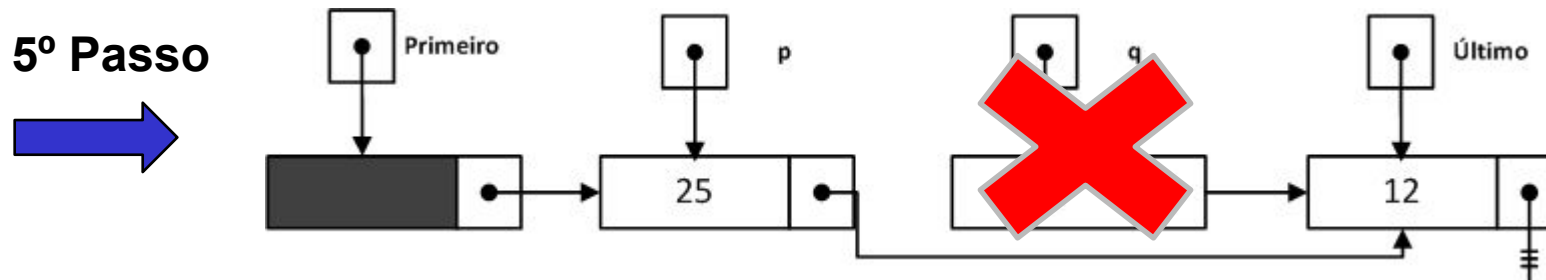
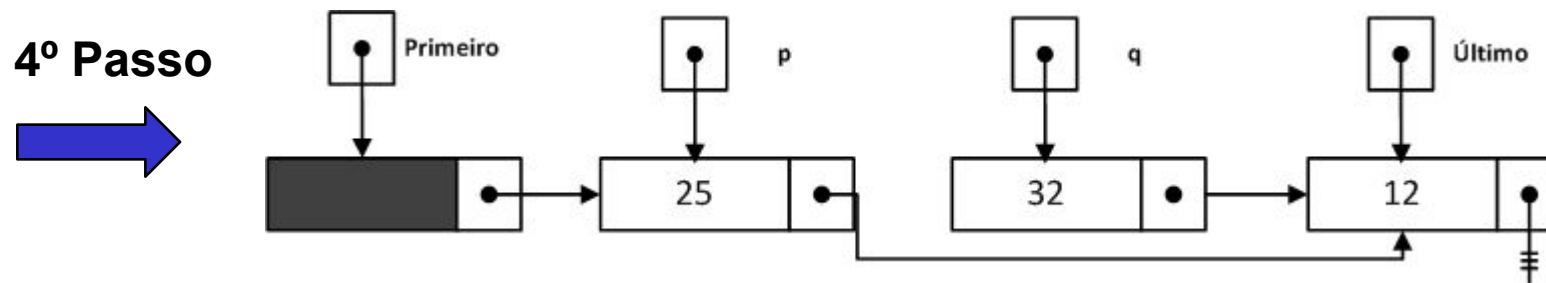
# Lista Simples Flexível

- Exercício: Seja nossa classe Lista, implemente um método que remove a segunda posição válida. Siga os passos da figura abaixo



## Lista Simples Flexível

- Exercício: Seja nossa classe Lista, implemente um método que remove a segunda posição válida. Siga os passos da figura abaixo





# Lista Dupla Flexível

## • Classe célula dupla

```
typedef struct CelulaDupla {  
    int elemento;  
    struct CelulaDupla *prox, *ant;  
} CelulaDupla;  
  
CelulaDupla *novaCelula(int elemento) {  
    CelulaDupla *nova = (CelulaDupla*) malloc(sizeof(CelulaDupla));  
    nova->elemento = elemento;  
    nova->prox = nova->ant = NULL;  
    return nova;  
}
```



# Lista Dupla Flexível

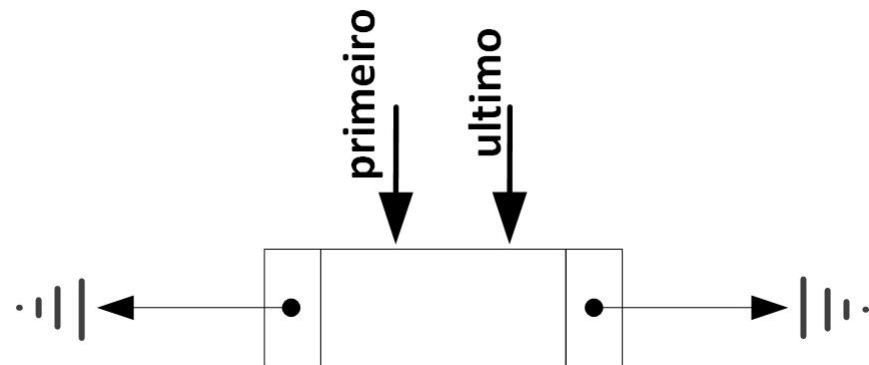
```
CelulaDupla *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```

Similar a Lista Simples,  
contudo, considerando o  
ponteiro ant

## Lista Dupla Flexível

```
CelulaDupla *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}
```

```
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```



## Lista Dupla Flexível

```
CelulaDupla *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}
```

```
void inserirInicio(int x) { ... }
```

```
void inserirFim(int x) { ... }
```

```
int removerInicio() { ... }
```

```
int removerFim() { ... }
```

```
void inserir(int x, int pos) { ... }
```

```
int remover(int pos) { ... }
```

```
void mostrar() { ... }
```

## Lista Dupla Flexível

## //LISTA DUPLA

```
void inserirInicio(int x) {  
    CelulaDupla *tmp = novaCelula(x);  
    tmp->ant = primeiro;  
    tmp->prox = primeiro->prox;  
    primeiro->prox = tmp;  
    if (primeiro == ultimo) {  
        ultimo = tmp;  
    } else {  
        tmp->prox->ant = tmp;  
    }  
    tmp = NULL;  
}
```

## //LISTA SIMPLES

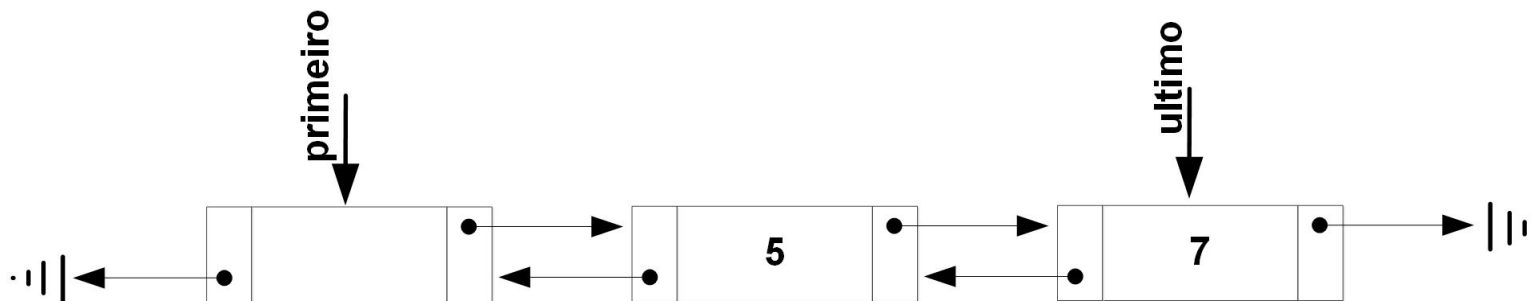
```
void inserirInicio(int x) {  
    Celula *tmp = novaCelula(x);  
  
    tmp->prox = primeiro->prox;  
    primeiro->prox = tmp;  
    if (primeiro == ultimo) {  
        ultimo = tmp;  
    }  
    tmp = NULL;  
}
```

# Lista Dupla Flexível

## //LISTA DUPLA

```
void inserirInicio(int x) {
    CelulaDupla *tmp = novaCelula(x);
    tmp->ant = primeiro;
    tmp->prox = primeiro->prox;
    primeiro->prox = tmp;
    if (primeiro == ultimo) {
        ultimo = tmp;
    } else {
        tmp->prox->ant = tmp;
    }
    tmp = NULL;
}
```

Supondo uma lista com os elementos 5 e 7, vamos inserir o 3 no início



## Lista Dupla Flexível

//Inserindo o 3 no início

void inserirInicio(int x) {

CelulaDupla \*tmp = novaCelula(x);

tmp-&gt;ant = primeiro;

tmp-&gt;prox = primeiro-&gt;prox;

primeiro-&gt;prox = tmp;

if (primeiro == ultimo) {

ultimo = tmp;

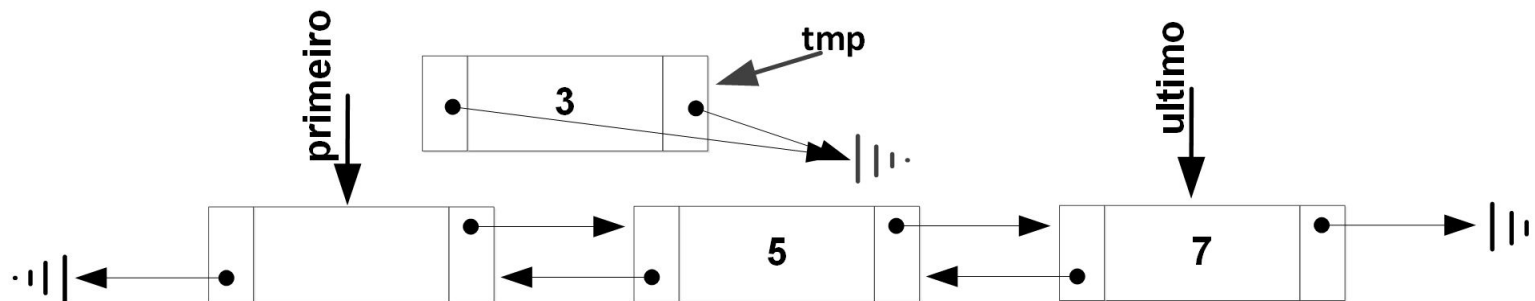
} else {

tmp-&gt;prox-&gt;ant = tmp;

}

tmp = NULL;

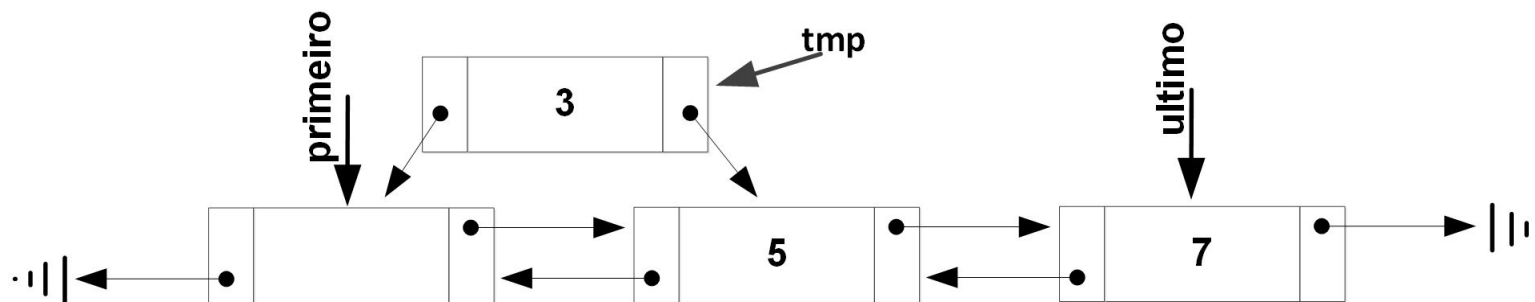
}



# Lista Dupla Flexível

**//Inserindo o 3 no início**

```
void inserirInicio(int x) {  
    CelulaDupla *tmp = novaCelula(x);  
    tmp->ant = primeiro;  
    tmp->prox = primeiro->prox;  
    primeiro->prox = tmp;  
    if (primeiro == ultimo) {  
        ultimo = tmp;  
    } else {  
        tmp->prox->ant = tmp;  
    }  
    tmp = NULL;  
}
```





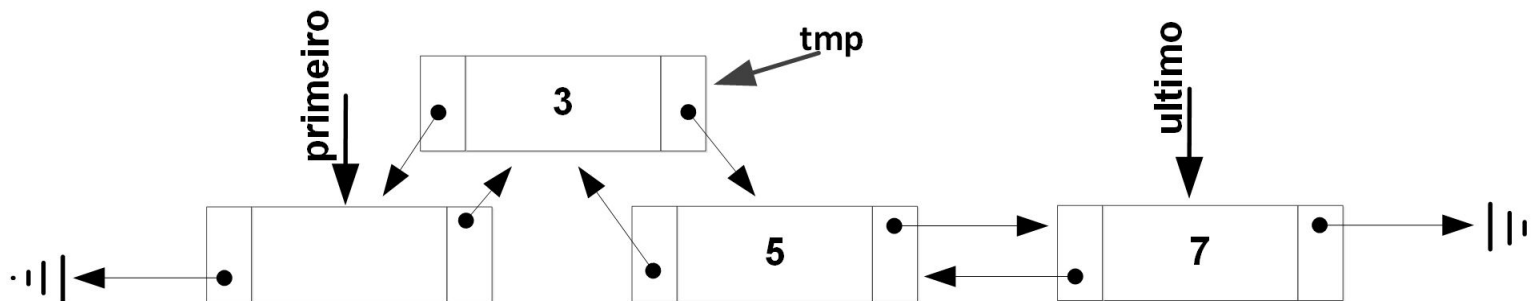
## Lista Dupla Flexível

**//Inserindo o 3 no início**

```

void inserirInicio(int x) {
    CelulaDupla *tmp = novaCelula(x);
    tmp->ant = primeiro;
    tmp->prox = primeiro->prox;
    primeiro->prox = tmp;
    if (primeiro == ultimo) {
        ultimo = tmp;
    } else {
        tmp->prox->ant = tmp;
    }
    tmp = NULL;
}

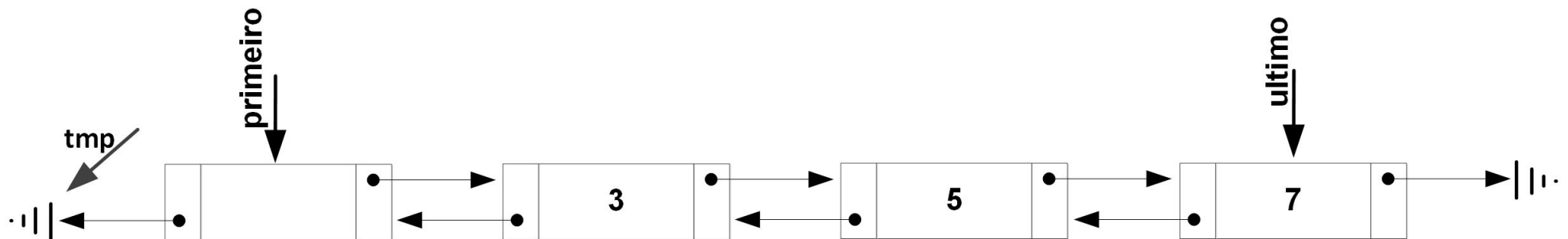
```



# Lista Dupla Flexível

**//Inserindo o 3 no início**

```
void inserirInicio(int x) {
    CelulaDupla *tmp = novaCelula(x);
    tmp->ant = primeiro;
    tmp->prox = primeiro->prox;
    primeiro->prox = tmp;
    if (primeiro == ultimo) {
        ultimo = tmp;
    } else {
        tmp->prox->ant = tmp;
    }
    tmp = NULL;
}
```



## Lista Dupla Flexível

```
CelulaDupla *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```

# Lista Dupla Flexível

## //LISTA DUPLA

```
void inserirFim(int x) {  
    ultimo->prox = novaCelula(x);  
    ultimo->prox->ant = ultimo;  
    ultimo = ultimo->prox;  
}
```

## //LISTA SIMPLES

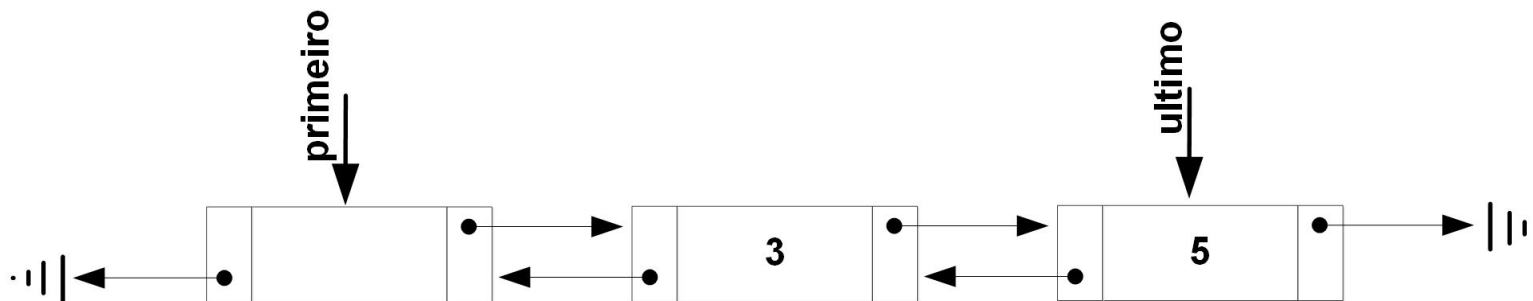
```
void inserirFim(int x) {  
    ultimo->prox = novaCelula(x);  
  
    ultimo = ultimo->prox;  
}
```

# Lista Dupla Flexível

**//LISTA DUPLA**

```
void inserirFim(int x) {  
    ultimo->prox = novaCelula(x);  
    ultimo->prox->ant = ultimo;  
    ultimo = ultimo->prox;  
}
```

Supondo uma lista com os elementos 3 e 5, vamos inserir o 7 no fim



# Lista Dupla Flexível

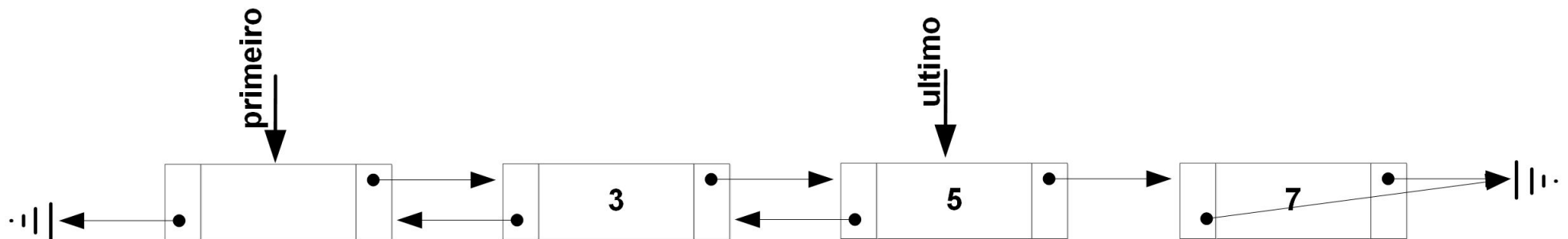
**//LISTA DUPLA****void** inserirFim(**int** x) {

ultimo-&gt;prox = novaCelula(x);

ultimo-&gt;prox-&gt;ant = ultimo;

ultimo = ultimo-&gt;prox;

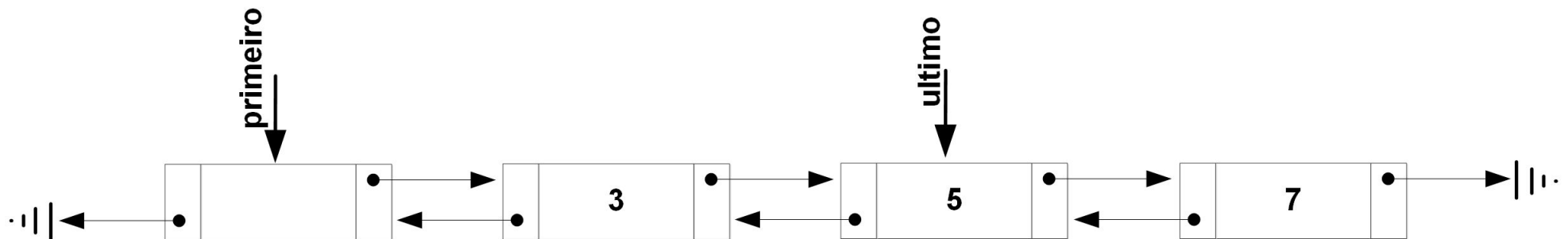
}



# Lista Dupla Flexível

**//LISTA DUPLA**

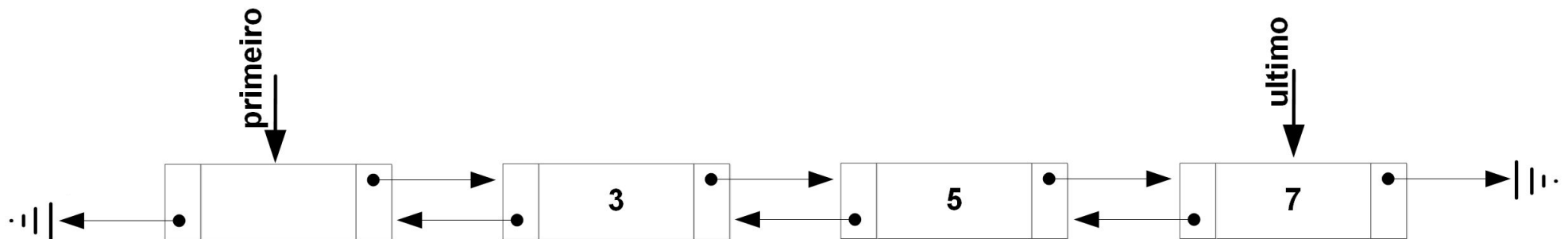
```
void inserirFim(int x) {  
    ultimo->prox = novaCelula(x);  
    ultimo->prox->ant = ultimo;  
    ultimo = ultimo->prox;  
}
```



# Lista Dupla Flexível

**//LISTA DUPLA**

```
void inserirFim(int x) {  
    ultimo->prox = novaCelula(x);  
    ultimo->prox->ant = ultimo;  
    ultimo = ultimo->prox;  
}
```





## Lista Dupla Flexível

```
CelulaDupla *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```

## Lista Dupla Flexível

## //LISTA DUPLA

```
int removerInicio() {  
    if (primeiro == ultimo)  
        errx(1, "Erro!");  
    CelulaDupla *tmp = primeiro;  
    primeiro = primeiro->prox;  
    int elemento = primeiro->elemento;  
    tmp->prox = primeiro->ant = NULL;  
    free(tmp);    tmp = NULL;  
    return elemento;  
}
```

## //LISTA SIMPLES

```
int removerInicio() {  
    if (primeiro == ultimo)  
        errx(1, "Erro!");  
    Celula *tmp = primeiro;  
    primeiro = primeiro->prox;  
    int elemento = primeiro->elemento;  
    tmp->prox = NULL;  
    free(tmp);    tmp = NULL;  
    return elemento;  
}
```

# Lista Dupla Flexível

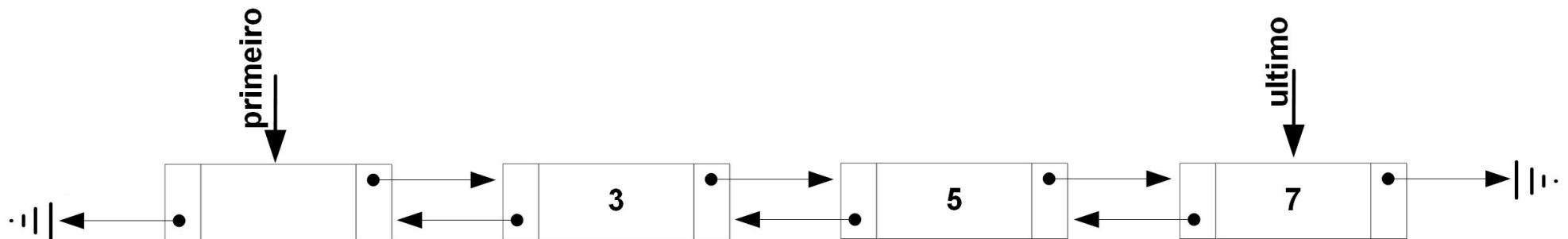
## //LISTA DUPLA

```

int removerInicio() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    CelulaDupla *tmp = primeiro;
    primeiro = primeiro->prox;
    int elemento = primeiro->elemento;
    tmp->prox = primeiro->ant = NULL;
    free(tmp);    tmp = NULL;
    return elemento;
}

```

**Exercício:** Supondo uma lista com os elementos 3, 5 e 7, execute o remover no início



## Lista Dupla Flexível

```
CelulaDupla *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```

# Lista Dupla Flexível

## //LISTA DUPLA

```
int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");

    int elemento = ultimo->elemento;
    ultimo = ultimo->ant;
    ultimo->prox->ant = NULL;
    free(ultimo->prox);
    ultimo->prox = NULL;
    return elemento;
}
```

## //LISTA SIMPLES

```
int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");
    Celula *i;
    for(i = primeiro; i->prox != ultimo; i = i->prox);
    int elemento = ultimo->elemento;
    ultimo = i; free(ultimo->prox);

    i = ultimo->prox = NULL;
    return elemento;
}
```

# Lista Dupla Flexível

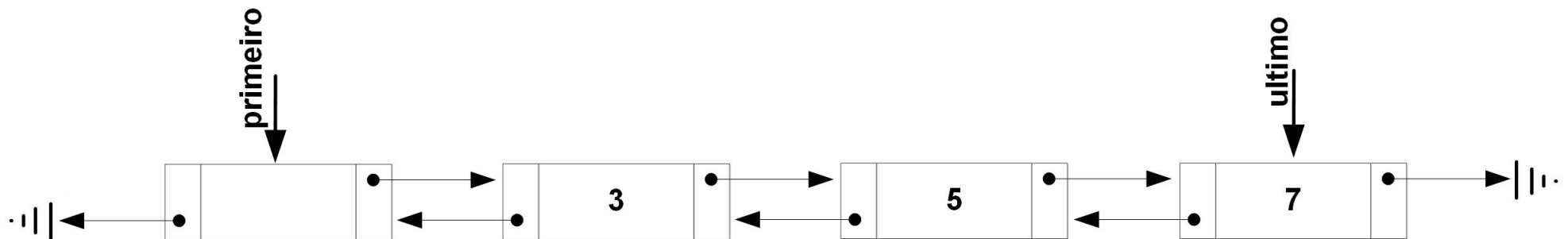
## //LISTA DUPLA

```

int removerFim() {
    if (primeiro == ultimo)
        errx(1, "Erro!");

    int elemento = ultimo->elemento;
    ultimo = ultimo->ant;
    ultimo->prox->ant = NULL;
    free(ultimo->prox);
    ultimo->prox = NULL;
    return elemento;
}
  
```

**Exercício:** Supondo uma lista com os elementos 3, 5 e 7, execute o remover no fim



## Lista Dupla Flexível

```
CelulaDupla *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```

## Lista Dupla Flexível

## //LISTA DUPLA

```

void inserir(int x, int pos) {
    int tamanho = tamanho();
    if (pos < 0 || pos > tamanho){
        errx(1, "Erro!");
    } else if (pos == 0){ inserirInicio(x);
    } else if (pos == tamanho){ inserirFim(x);
    } else {
        CelulaDupla i = primeiro;
        for (int j = 0; j < pos; j++, i = i->prox);

        CelulaDupla *tmp = novaCelula(x);
        tmp->ant = i;
        tmp->prox = i->prox;
        tmp->ant->prox = tmp->prox->ant =
tmp;
        tmp = i = NULL;
    }
}

```

## //LISTA SIMPLES

```

void inserir(int x, int pos) {
    int tamanho = tamanho();
    if (pos < 0 || pos > tamanho){
        errx(1, "Erro!");
    } else if (pos == 0){ inserirInicio(x);
    } else if (pos == tamanho){ inserirFim(x);
    } else {
        Celula *i = primeiro;
        for (int j = 0; j < pos; j++, i = i->prox);

        Celula *tmp = novaCelula(x);

        tmp->prox = i->prox;
        i->prox = tmp;
        tmp = i = NULL;
    }
}

```



## Lista Dupla Flexível

## //LISTA DUPLA

```

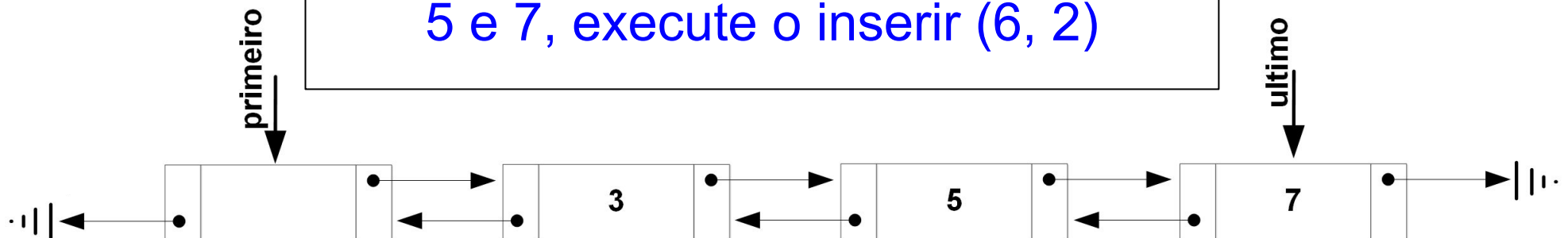
void inserir(int x, int pos) {
    int tamanho = tamanho();
    if (pos < 0 || pos > tamanho){
    } else if (pos == 0){
    } else if (pos == tamanho){
    } else {
        CelulaDupla i = primeiro;

        CelulaDupla *tmp = novaCelula(x);
        tmp->ant = i;
        tmp->ant->prox = tmp->prox->ant = tmp;
    }
}

```

errx(1, "Erro!");  
 inserirInicio(x);  
 inserirFim(x);  
 for (int j = 0; j < pos; j++, i = i->prox);  
 tmp->prox = i->prox;  
 tmp = i = NULL;

**Exercício:** Supondo a lista com o 3, 5 e 7, execute o inserir (6, 2)



## Lista Dupla Flexível

```
CelulaDupla *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```

## Lista Dupla Flexível

## //LISTA DUPLA

```

int remover(int pos) {
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo){
        errx(1, "Erro!");
    } else if (pos < 0 || pos >= tamanho){
        errx(1, "Erro!");
    } else if (pos == 0){
        elemento = removerInicio();
    } else if (pos == tamanho - 1){
        elemento = removerFim();
    } else {
        CelulaDupla i = primeiro->prox;
        for (int j = 0; j < pos; j++, i = i->prox);
        i->ant->prox = i->prox;
        i->prox->ant = i->ant;
        elemento = i->elemento;
        i->prox = i->ant = NULL;
        free(i);
        i = NULL;
    }
    return elemento;
}

```

## //LISTA SIMPLES

```

int remover(int pos) {
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo){
        errx(1, "Erro!");
    } else if (pos < 0 || pos >= tamanho){
        errx(1, "Erro!");
    } else if (pos == 0){
        elemento = removerInicio();
    } else if (pos == tamanho - 1){
        elemento = removerFim();
    } else {
        Celula *i = primeiro;
        for (int j = 0; j < pos; j++, i = i->prox);
        Celula *tmp = i->prox;
        elemento = tmp->elemento;
        i->prox = tmp->prox;
        tmp->prox = NULL;
        free(tmp);
        i = tmp = NULL;
    }
    return elemento;
}

```

## Lista Dupla Flexível

## //LISTA DUPLA

```

int remover(int pos) {
    int elemento, tamanho = tamanho();
    if (primeiro == ultimo){
    } else if (pos < 0 || pos >= tamanho){
    } else if (pos == 0){
    } else if (pos == tamanho - 1){
    } else {
        CelulaDupla i = primeiro->prox;
        i->ant->prox = i->prox;
        elemento = i->elemento;
    }
    return elemento;
}

```

```
errx(1, "Erro!");
```

```
errx(1, "Erro!");
```

```
elemento = removerInicio();
```

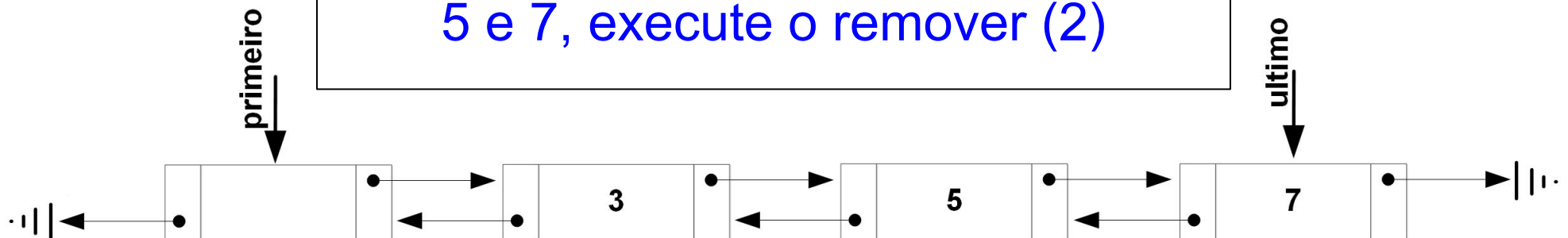
```
elemento = removerFim();
```

```
for (int j = 0; j < pos; j++, i = i->prox);
```

```
i->prox->ant = i->ant; free(i);
```

```
i = i->prox = i->ant = NULL;
```

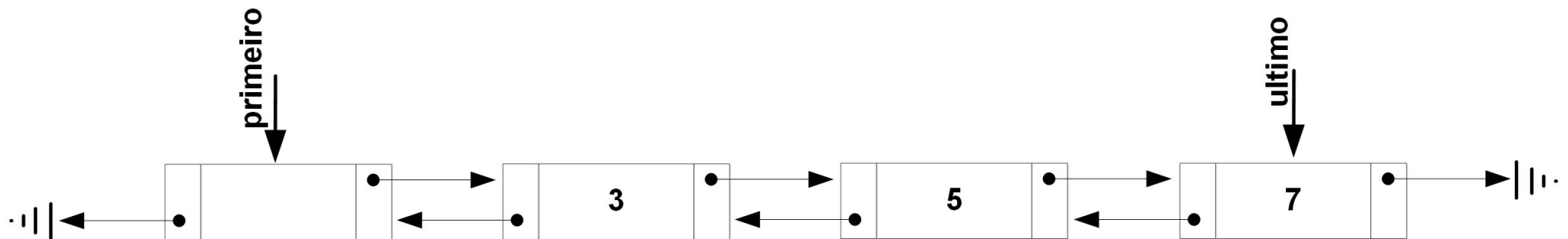
**Exercício:** Supondo a lista com o 3, 5 e 7, execute o remover (2)



# Lista Dupla Flexível

```
CelulaDupla *primeiro, *ultimo;  
void start () {  
    primeiro = novaCelula(-1);  
    ultimo = primeiro;  
}  
void inserirInicio(int x) { ... }  
void inserirFim(int x) { ... }  
int removerInicio() { ... }  
int removerFim() { ... }  
void inserir(int x, int pos) { ... }  
int remover(int pos) { ... }  
void mostrar() { ... }
```

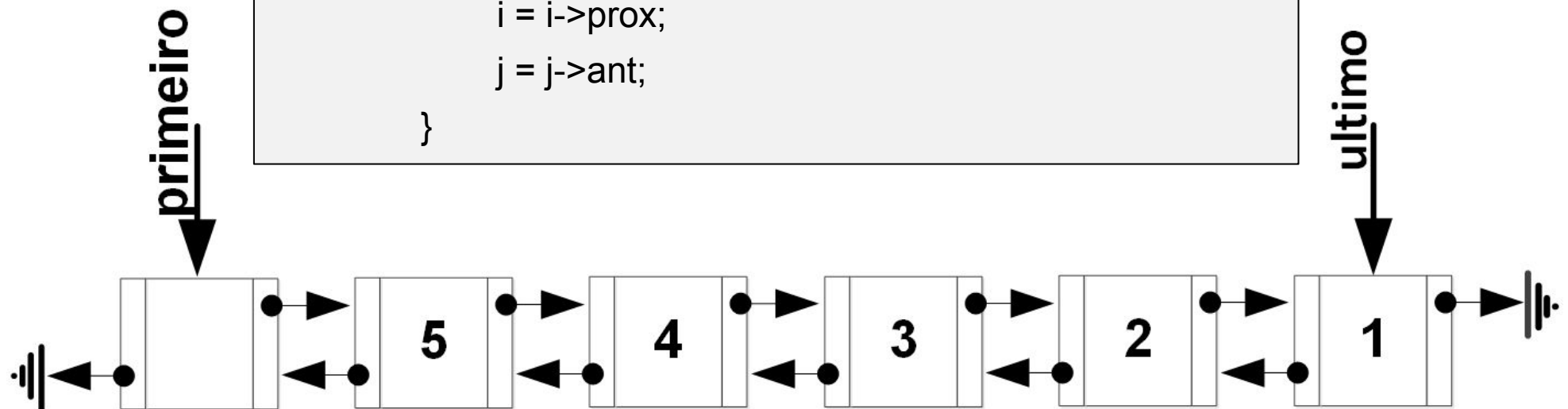
**Exercício:** Implemente o mostrar e o execute para uma lista com os elementos 3, 5 e 7



## Exercício

- Exercício: Faça um método que inverta a ordem dos elementos da lista dupla. No exemplo abaixo, após a inversão, os elementos ficarão na ordem crescente

```
void inverte(){  
    Celula *i = primeiro->prox;   Celula *j = ultimo;  
    while (i != j && j->prox != i){  
        int tmp = i->elemento;  
        i->elemento = j->elemento;  
        j->elemento = tmp;  
        i = i->prox;  
        j = j->ant;  
    }  
}
```

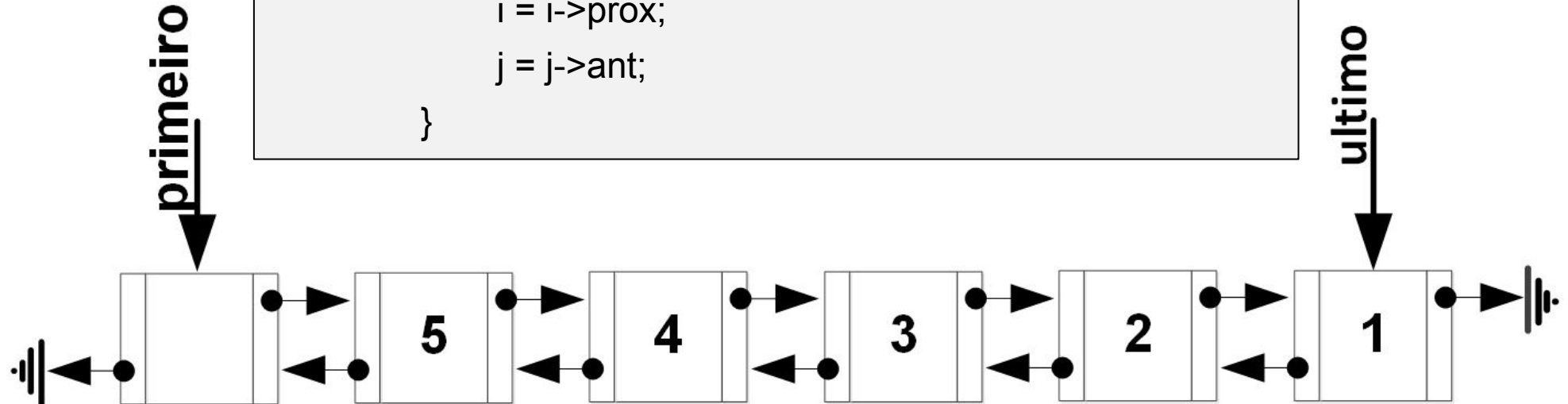


## Exercício

- Exercício: Faça um método que inverta a ordem dos elementos da lista dupla. No exemplo abaixo, após a inversão, os elementos ficarão na ordem crescente

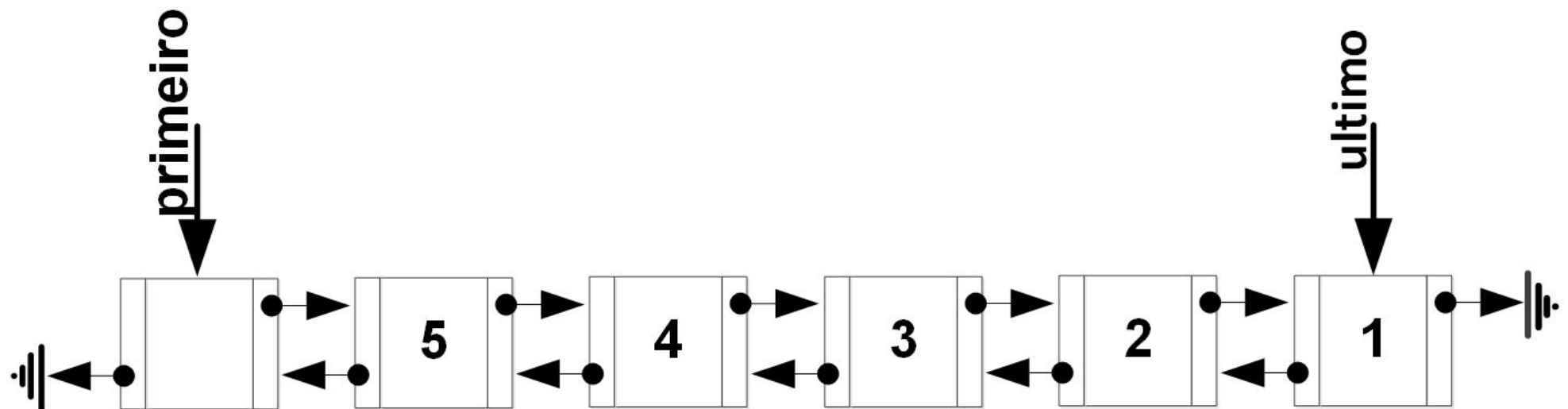
```
void inverte(){  
    Celula *i = primeiro->prox;   Celula *j = ultimo;  
    while (i != j && j->prox != i){  
        int tmp = i->elemento;  
        i->elemento = j->elemento;  
        j->elemento = tmp;  
        i = i->prox;  
        j = j->ant;  
    }  
}
```

Veja que a condição  $i \neq j$  é para uma lista com número par de elementos e  $j \rightarrow \text{prox} \neq i$ , para as com número ímpar de elementos



## Exercício

- Exercício: Faça um método que inverta a ordem dos elementos da lista **simples**. No exemplo abaixo, após a inversão, os elementos ficarão na ordem crescente





## Exercício

- Exercício: Faça um método que inverta a ordem dos elementos da lista **simples**. No exemplo abaixo, após a inversão, os elementos ficarão na ordem cresc

```
void inverte(){  
    Celula *i = primeiro->prox;   Celula *j = ultimo;  
    Celula *k;  
    while (i != j && j->prox != i){  
        int tmp = i->elemento;  
        i->elemento = j->elemento;  
        j->elemento = tmp;  
        i = i->prox;  
        for (k = primeiro; k->prox != j; k = k->prox); j = k;  
    }
```

