



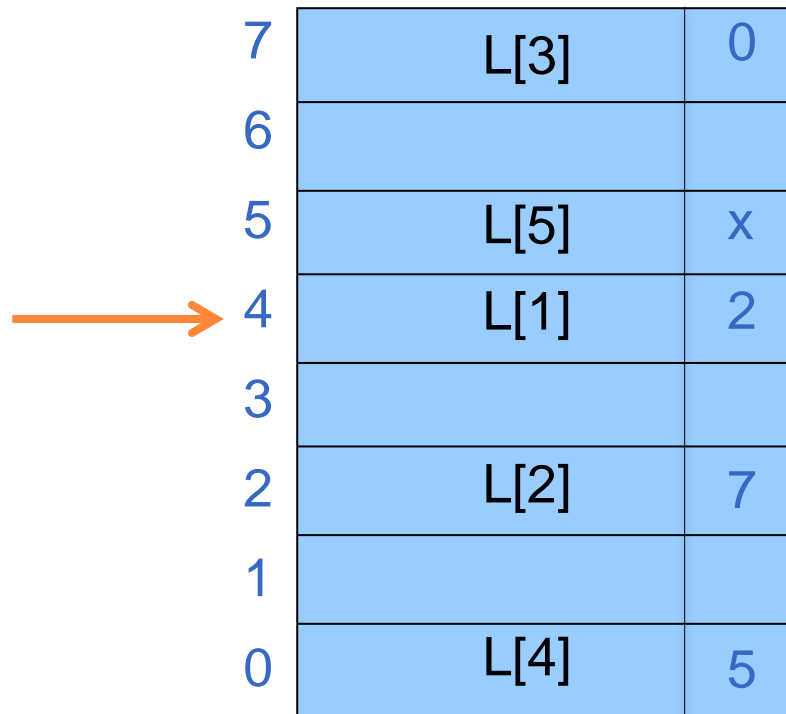
PROGRAMAÇÃO ESTRUTURADA

Pilhas, Filas e Listas

Com Alocação Encadeada Dinâmica

LISTA LINEARES

- Alocação Encadeada Dinâmica

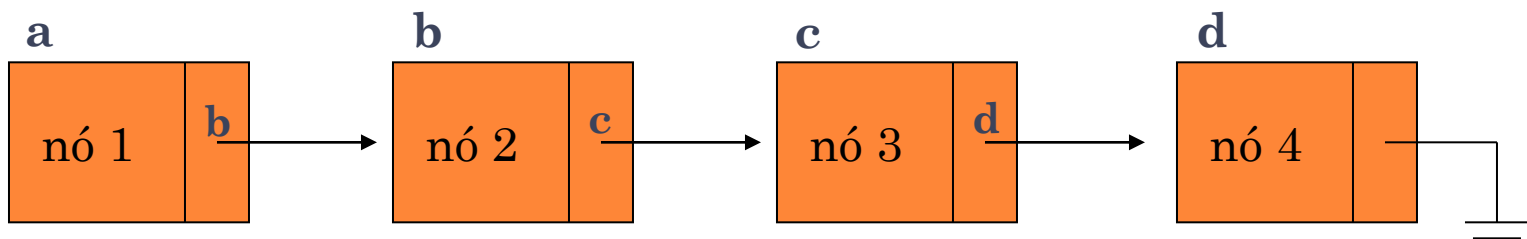


7	L[3]	0
6		
5	L[5]	x
4	L[1]	2
3		
2	L[2]	7
1		
0	L[4]	5

Memória

LISTAS LINEARES - PONTEIROS

- Os elementos são chamados de nós
- Como os elementos não se encontram em posições contíguas de memória é necessário que cada elemento saiba o endereço do próximo
- Logo, cada nó é composto por duas partes:
 - A informação do elemento
 - O endereço do próximo nó → ponteiro



LISTAS LINEARES ENCADEADAS

- Suponha uma lista L com n elementos.
- Cada elemento possui um identificador e o nome de um aluno.
- O identificador pode ser o número do aluno no diário.
- Representação gráfica:

12	Maria
id	nome

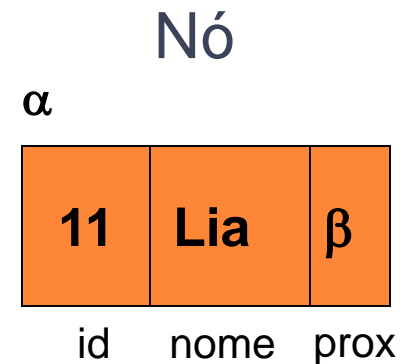
Como seria a
representação
desta lista em C?

LISTAS LINEARES ENCADEADAS

○ Exemplo 1:

- Declaração de uma lista L simplesmente encadeada com números e nomes de alunos

```
struct NO {  
    int id;  
    char nome[40];  
    struct NO *prox;  
}  
typedef struct NO lista;  
  
...  
lista *L;
```

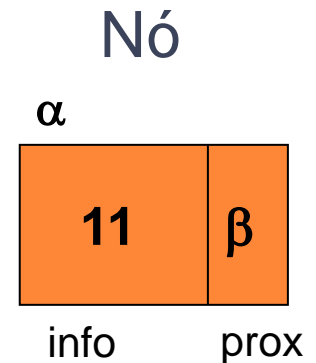


LISTAS LINEARES ENCADEADAS

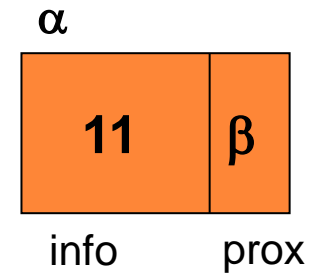
○ Exemplo 2:

- Declaração de uma lista L simplesmente encadeada com números inteiros

```
struct NO {  
    int info;  
    struct NO *prox;  
}  
typedef struct NO lista;  
  
...  
lista *L;
```

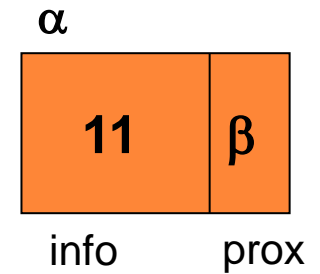


LISTAS LINEARES ENCADEADAS



- Sendo L uma variável do tipo ponteiro, então L guarda um endereço
- L guarda o endereço de um Nó da lista \rightarrow guarda o endereço de um struct NO
 - Cada Nó possui duas partes: info e prox
 - L
 - $L \rightarrow \text{info}$
 - $L \rightarrow \text{prox}$
- O endereço final da lista é marcado por NULL.

LISTAS LINEARES ENCADEADAS

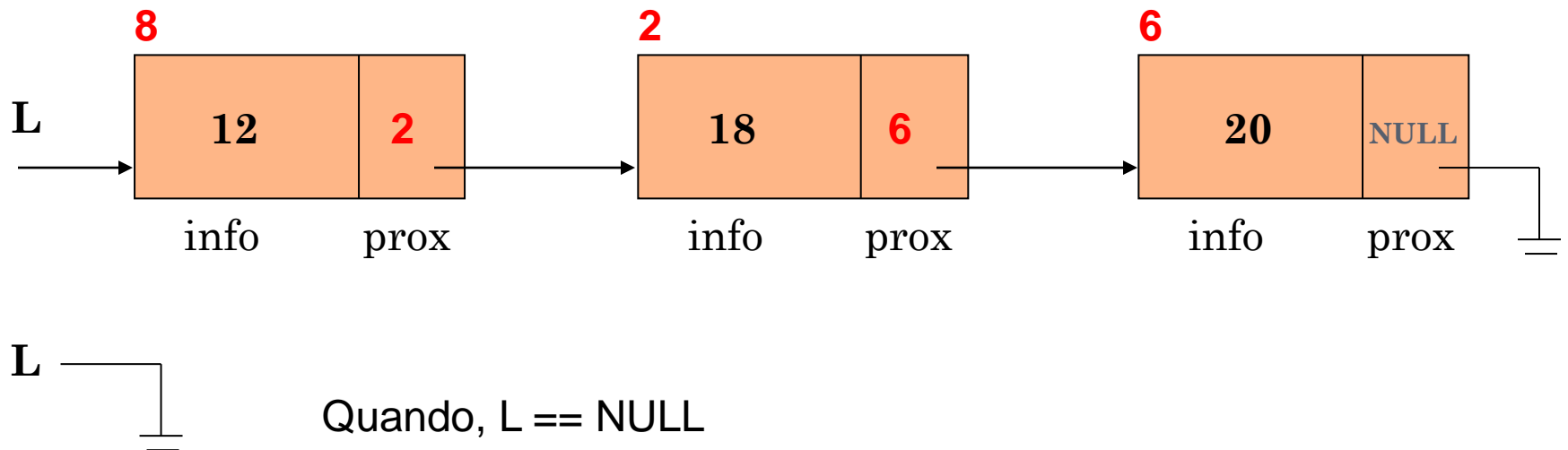


- Para alocar uma nova área de memória, como por exemplo, para criar um novo nó, deve se especificar no comando de alocação, a variável ponteiro que irá guardar o novo endereço alocado.
- Exemplo:

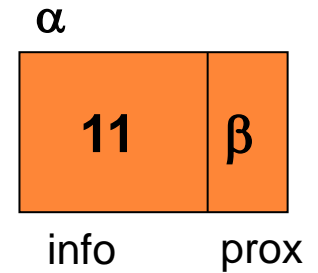
```
lista *L, *aux;  
L = NULL;  
...  
aux = (lista *) malloc (sizeof(lista));
```


LISTAS LINEARES ENCADEADAS

- O endereço inicial da lista deve sempre ser preservado
- É a partir dele que é possível percorrer toda a lista
- A ideia é seguir pelos endereços existentes nos campos que indicam o próximo nó



FUNÇÃO BUSCA ELEMENTO EM L

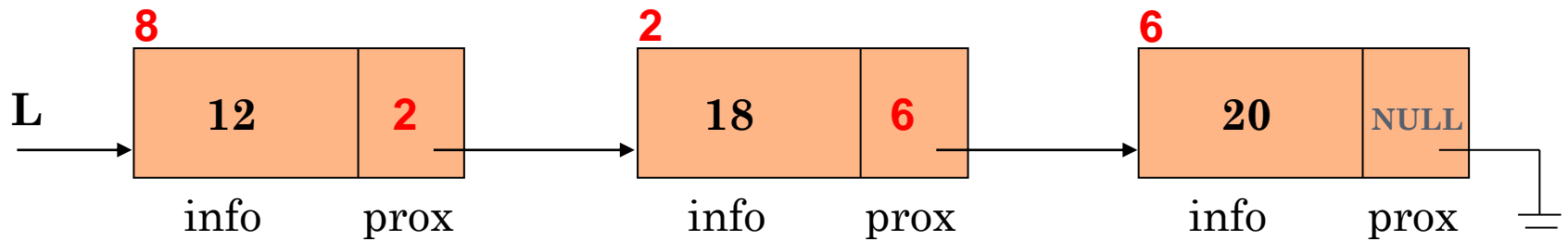


- Considere uma Lista L que guarde números inteiros ordenados
- Cada Nó da lista possui a parte info e a parte prox
- A função buscaElemento terá o seguinte protótipo:

`int buscaElem(lista *L, int elem, lista **pre)`

- L: guarda o endereço inicial da lista
- elem: o elemento que será procurado
- *pre: retorna o endereço do nó anterior ao procurado
- A função retorna 1 se encontrou o elemento e 0, caso contrário

FUNÇÃO BUSCA ELEMENTO EM L (ELEM=20)



```
int buscaElem(lista *L, int elem, lista **pre){
```

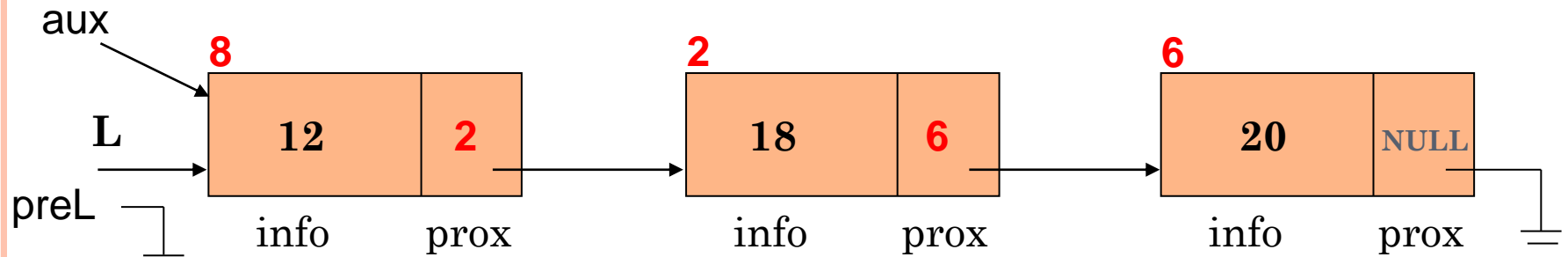
```
    lista *aux, *preL;
```

```
    aux = L;
```

```
    preL = NULL;
```

```
}
```

FUNÇÃO BUSCA ELEMENTO EM L (ELEM=20)



```
int buscaElem(lista *L, int elem, lista **pre){
```

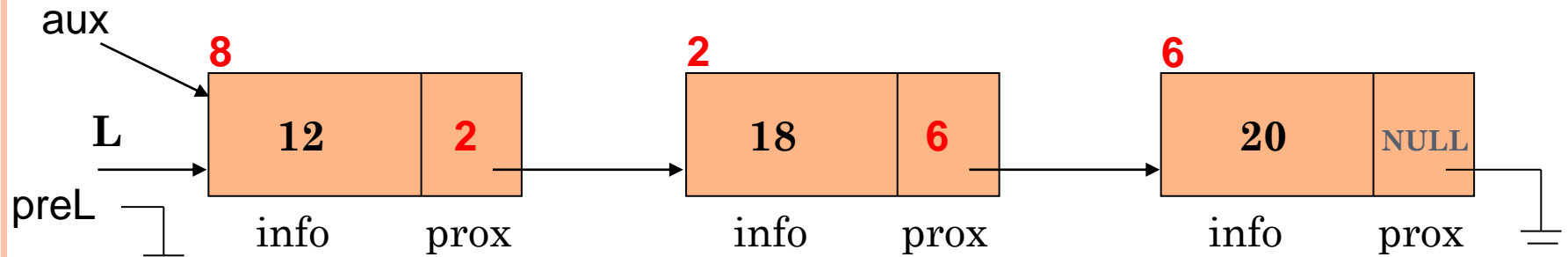
```
    lista *aux, *preL;
```

```
    aux = L;
```

```
    preL = NULL;
```

```
}
```

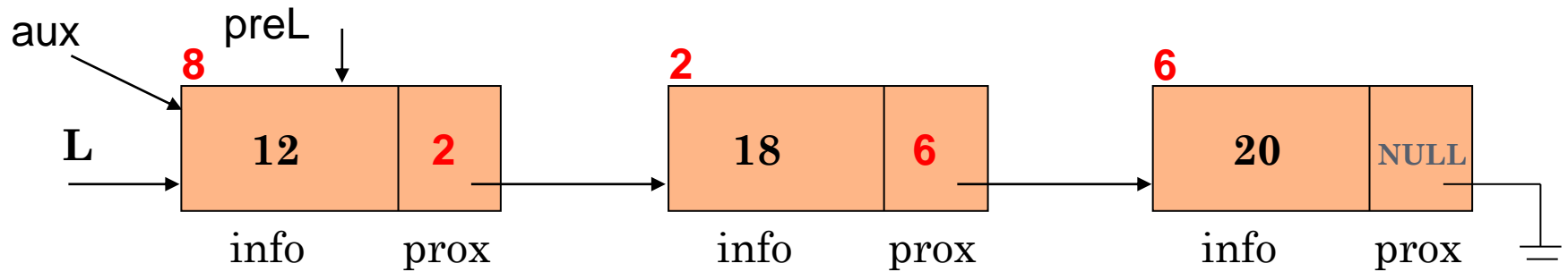
FUNÇÃO BUSCA ELEMENTO EM L (ELEM=20)



```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL = NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
}
```

}

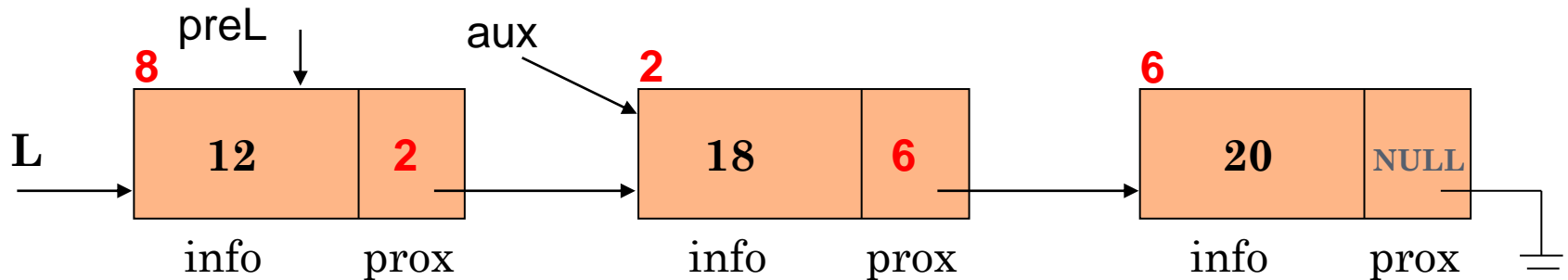
FUNÇÃO BUSCA ELEMENTO EM L (ELEM=20)



```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL = NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
}
```

}

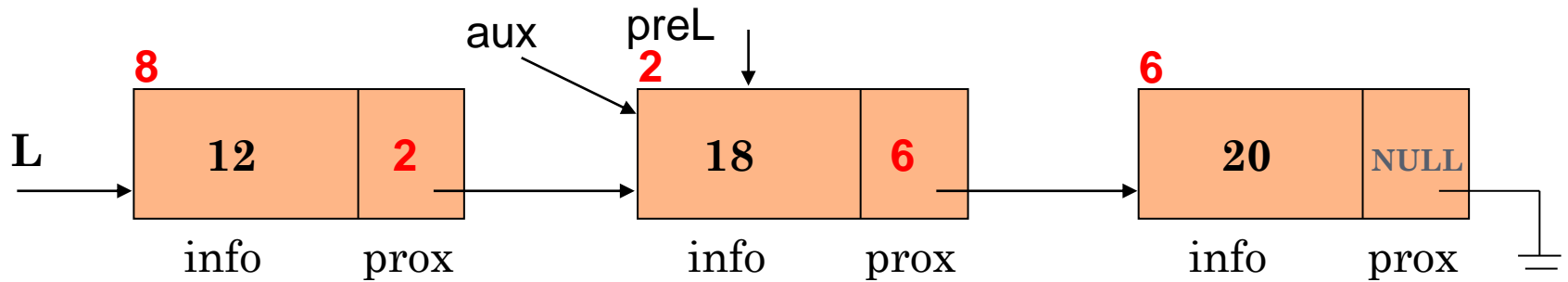
FUNÇÃO BUSCA ELEMENTO EM L (ELEM=20)



```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL= NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
}
```

}

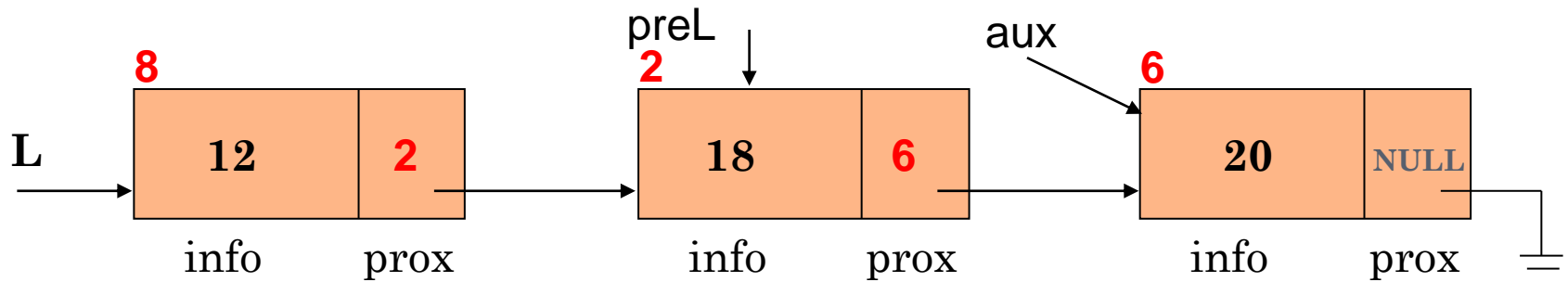
FUNÇÃO BUSCA ELEMENTO EM L (ELEM=20)



```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL= NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
}
```

}

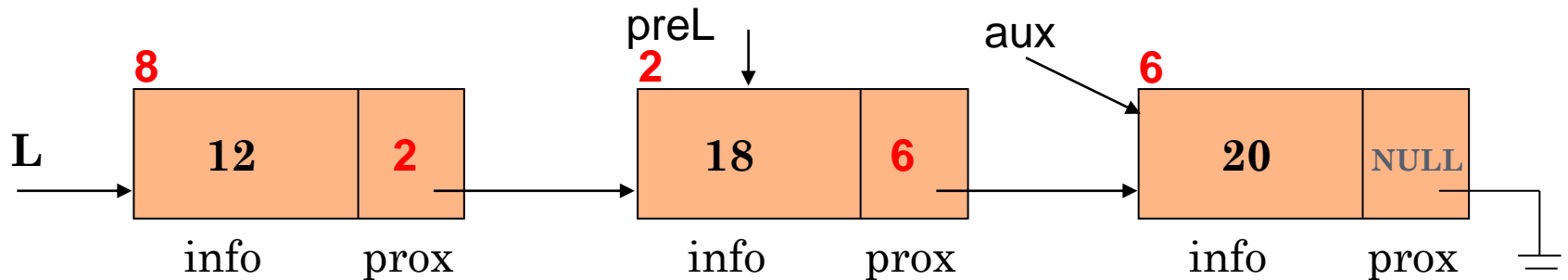
FUNÇÃO BUSCA ELEMENTO EM L (ELEM=20)



```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL= NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
}
```

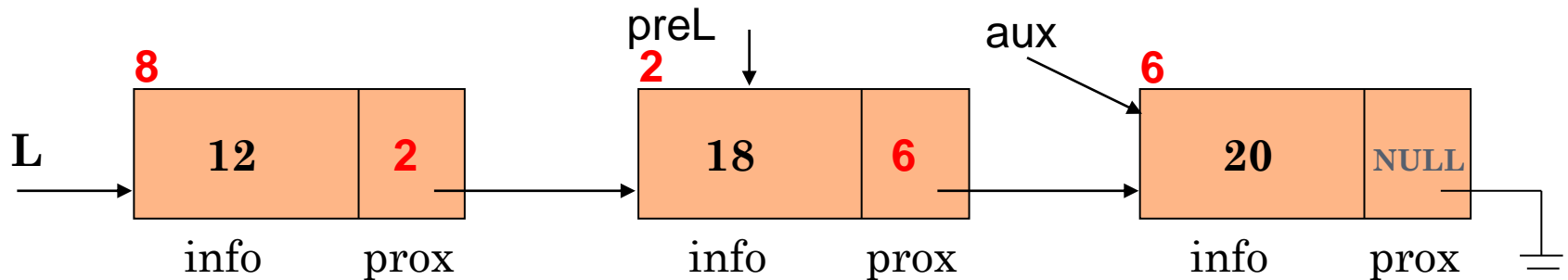
}

FUNÇÃO BUSCA ELEMENTO EM L (ELEM=20)



```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL= NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
    (*pre) = preL;  
    if ((aux != NULL) && (elem == aux->info))  
        return 1;  
    return 0;  
}
```

FUNÇÃO BUSCA ELEMENTO EM L (ELEM=20)

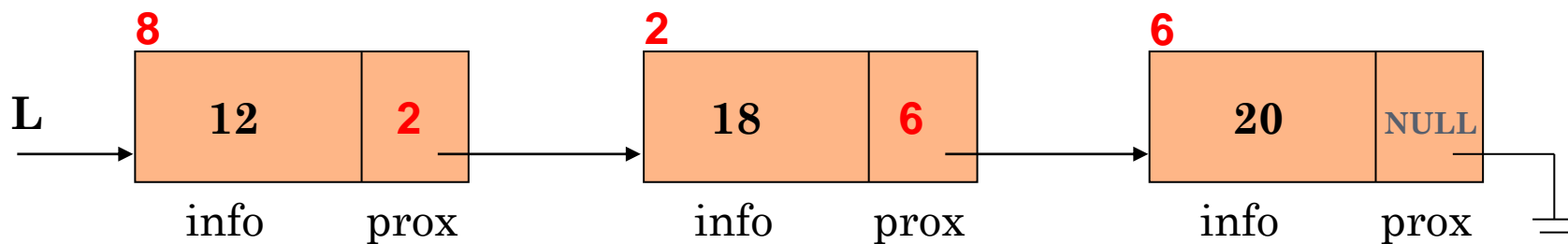


```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL= NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
    (*pre) = preL;  
    if ((aux != NULL) && (elem == aux->info))  
        return 1;  
    return 0;  
}
```

SAÍDA

```
aux = end.6  
aux->info = 20  
preL = end.2  
return 1
```

QUAL A SAÍDA?? (ELEM=12)

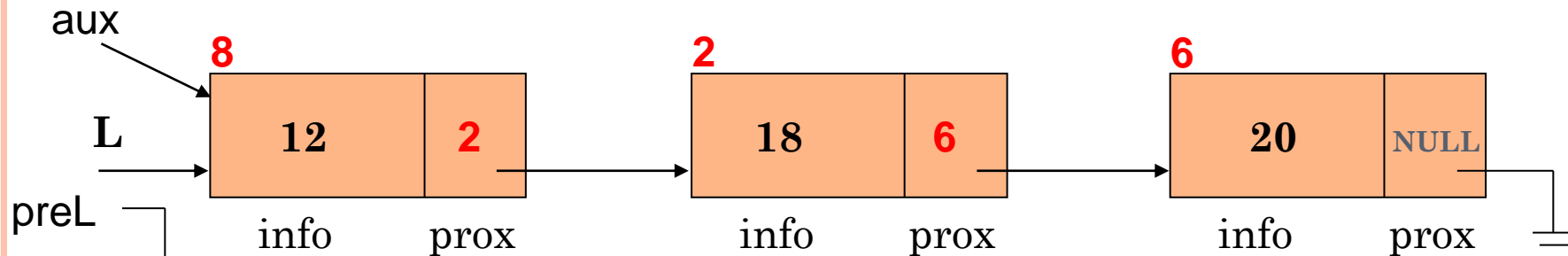


```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL = NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
    (*pre) = preL;  
    if ((aux != NULL) && (elem == aux->info))  
        return 1;  
    return 0;  
}
```

SAÍDA

aux = ???
aux->info = ??
preL = ???
return ?

QUAL A SAÍDA?? (ELEM=12)

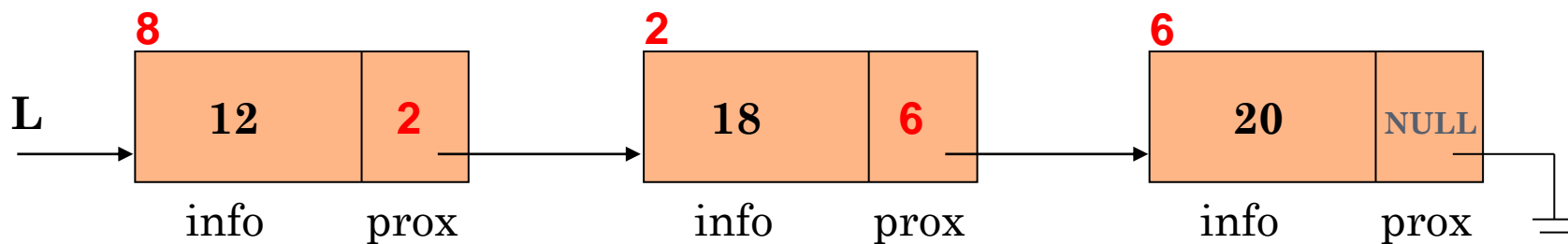


```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL = NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
    (*pre) = preL;  
    if ((aux != NULL) && (elem == aux->info))  
        return 1;  
    return 0;  
}
```

SAÍDA

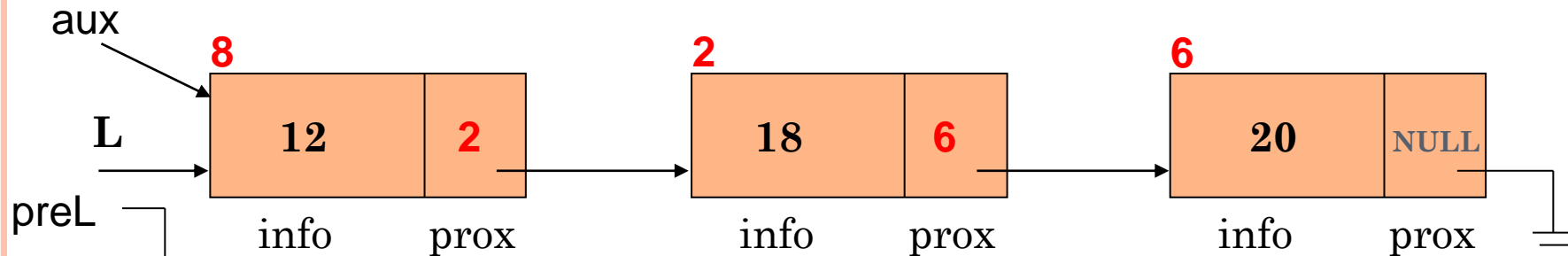
aux = end.8
aux->info = 12
preL = NULL
return 1

QUAL A SAÍDA?? (ELEM=11)



```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL= NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
    (*pre) = preL;  
    if ((aux != NULL) && (elem == aux->info))  
        return 1;  
    return 0;  
}
```

QUAL A SAÍDA?? (ELEM=11)

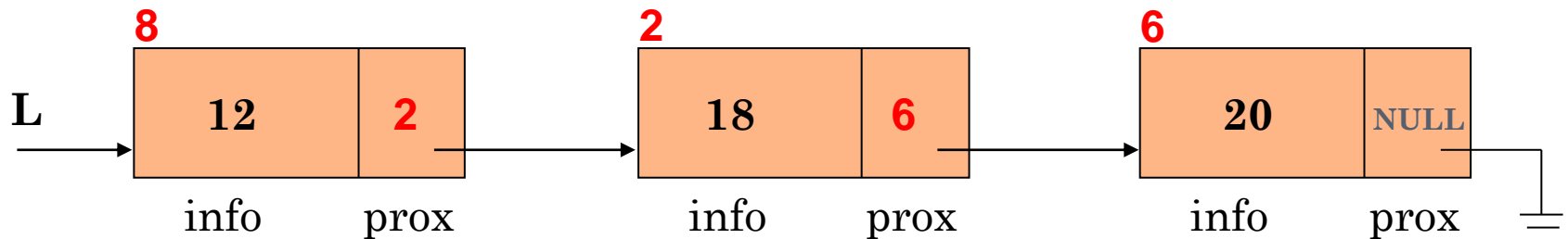


```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL = NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
    (*pre) = preL;  
    if ((aux != NULL) && (elem == aux->info))  
        return 1;  
    return 0;  
}
```

SAÍDA

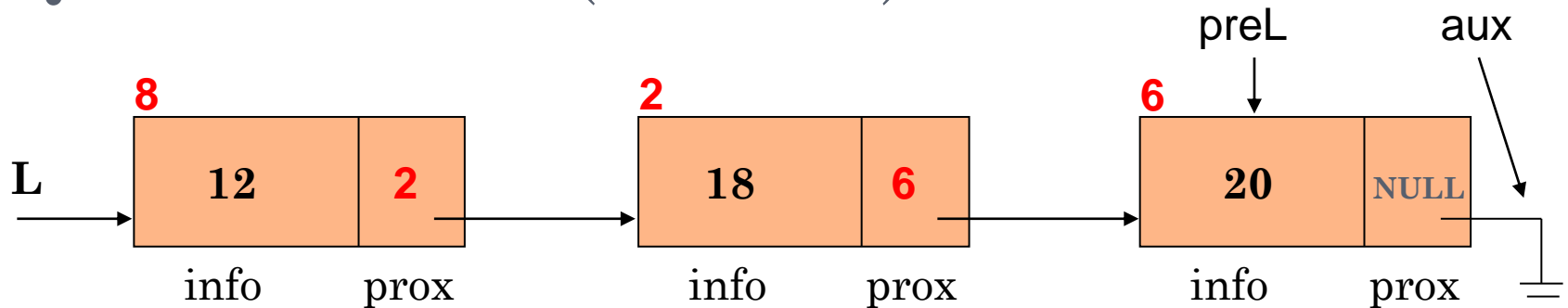
aux = end.8
aux->info = 12
preL = NULL
return 0

QUAL A SAÍDA?? (ELEM=22)



```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL= NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
    (*pre) = preL;  
    if ((aux != NULL) && (elem == aux->info))  
        return 1;  
    return 0;  
}
```


QUAL A SAÍDA?? (ELEM=22)

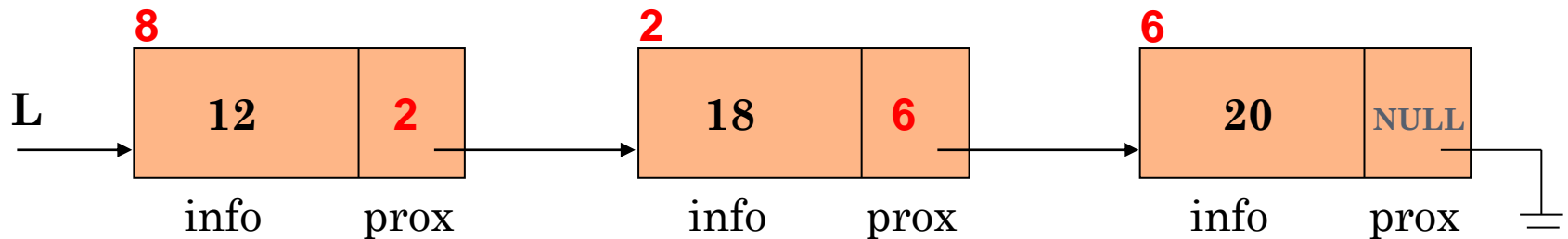


```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL= NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
    (*pre) = preL;  
    if ((aux != NULL) && (elem == aux->info))  
        return 1;  
    return 0;  
}
```

SAÍDA

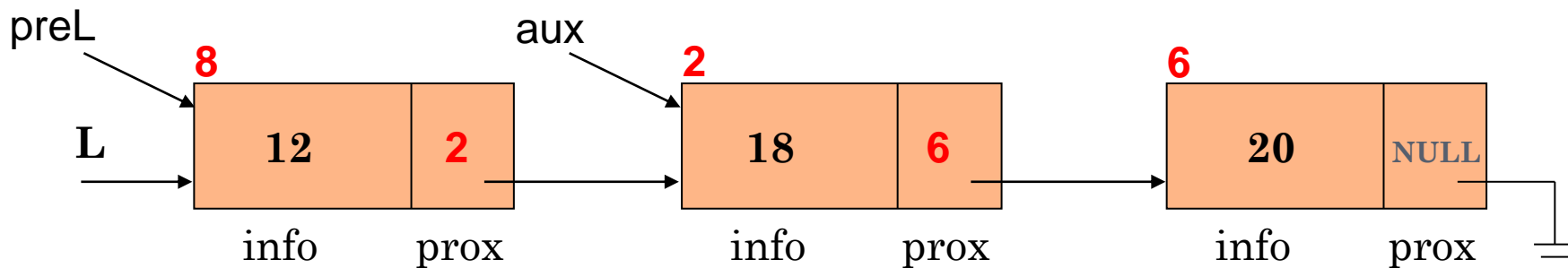
aux = NULL
aux->info = **
preL = end.6
return 0

QUAL A SAÍDA?? (ELEM=14)



```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL= NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
    (*pre) = preL;  
    if ((aux != NULL) && (elem == aux->info))  
        return 1;  
    return 0;  
}
```

QUAL A SAÍDA?? (ELEM=14)



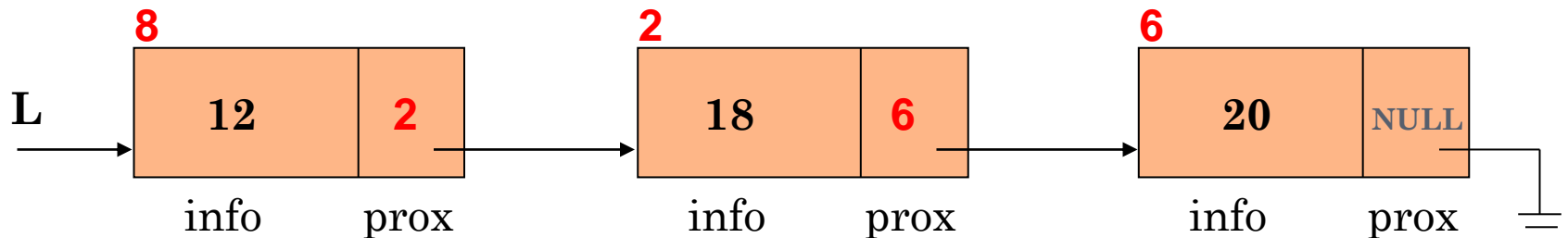
```
int buscaElem(lista *L, int elem, lista **pre){  
    lista *aux, *preL;  
    aux = L;  
    preL = NULL;  
    while ((aux != NULL) && (elem > aux->info)) {  
        preL = aux;  
        aux = aux->prox;  
    }  
    (*pre) = preL;  
    if ((aux != NULL) && (elem == aux->info))  
        return 1;  
    return 0;  
}
```

SAÍDA

aux = end.2
aux->info = 18
preL = end.8
return 0

INSERINDO UM NÓ EM L

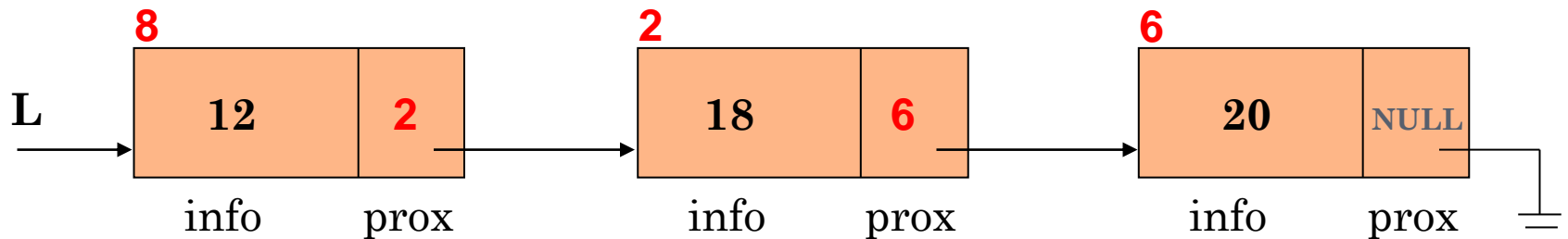
- Considere a lista ordenada de inteiros, com endereço inicial guardado em L



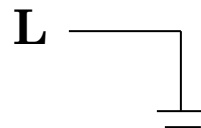
- Não podem existir elementos repetidos
- Logo, o elemento só será inserido na lista L se buscaElem retornar 0
- O novo elemento será inserido após o nó pre, cujo valor retornou de buscaElem

INSERINDO UM NÓ EM L

- Se após buscaEleme, return 0 e pre == NULL
 - Então, elemento será inserido no início da lista ou em uma lista vazia
- Início da lista (elem = 11)

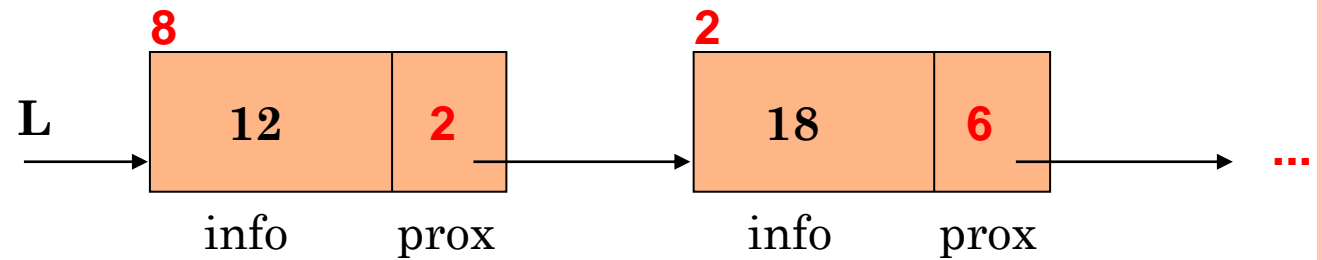


- Lista vazia (elem = 11)



INSERINDO UM NÓ EM L (ELEM = 11)

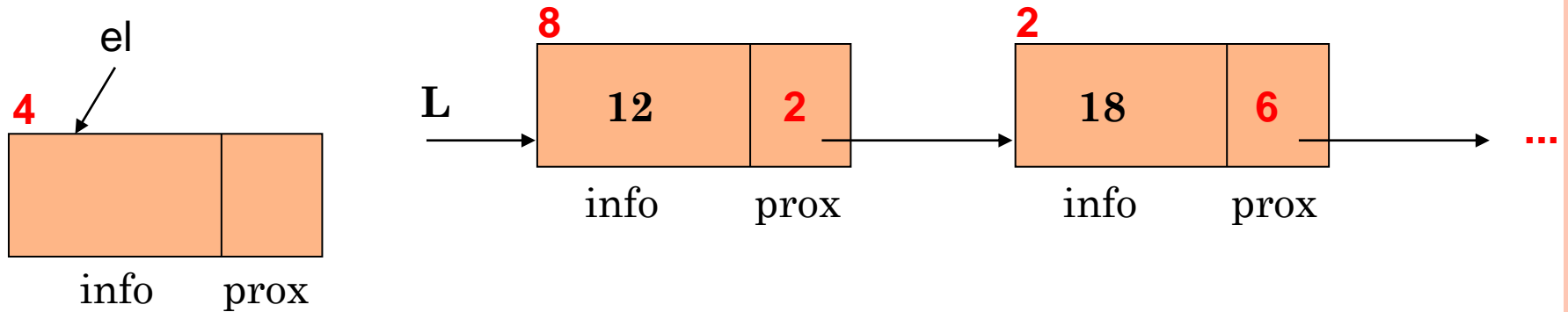
- Início da lista



```
el = (lista *) malloc(sizeof(lista));
```

INSERINDO UM NÓ EM L (ELEM = 11)

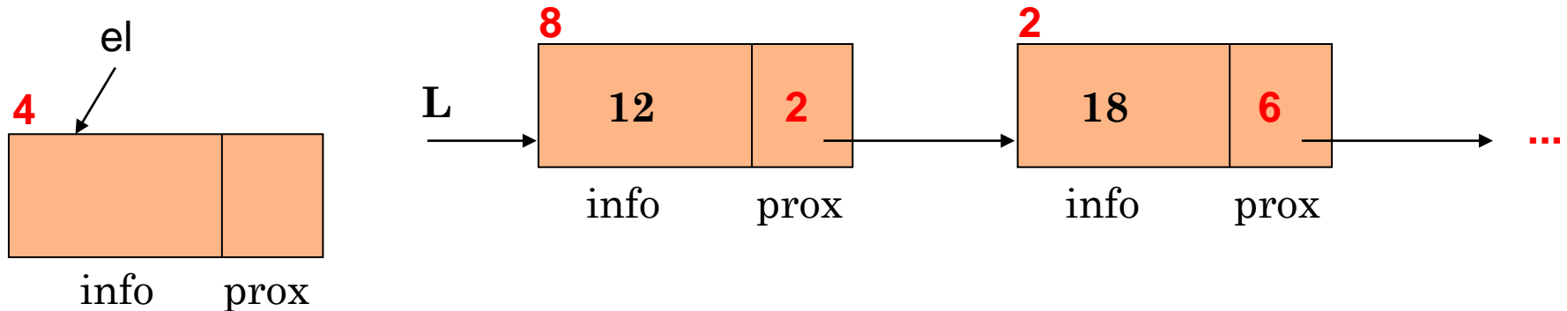
○ Início da lista



```
el = (lista *) malloc(sizeof(lista));
```

INSERINDO UM NÓ EM L (ELEM = 11)

○ Início da lista



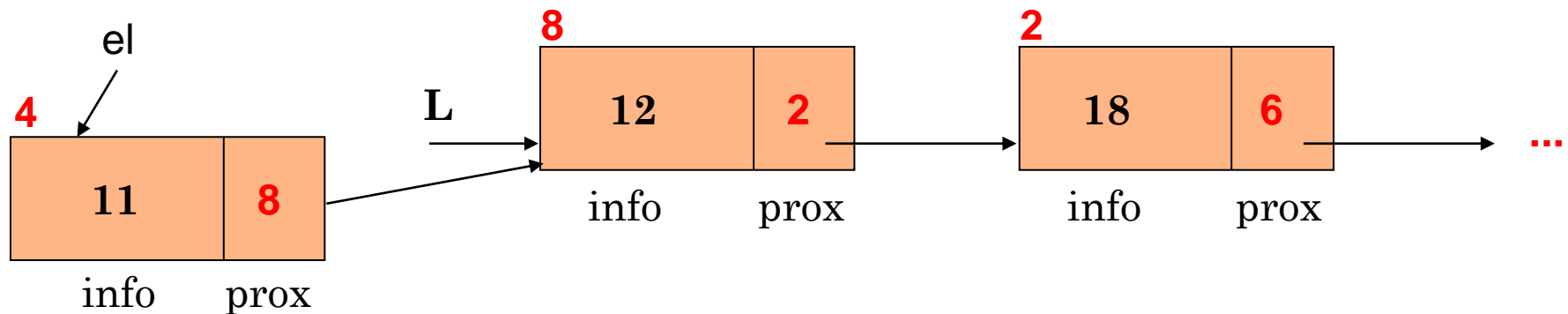
```
el = (lista *) malloc(sizeof(lista));
```

```
el->info = elem;
```

```
el->prox = L;
```


INSERINDO UM NÓ EM L (ELEM = 11)

○ Início da lista



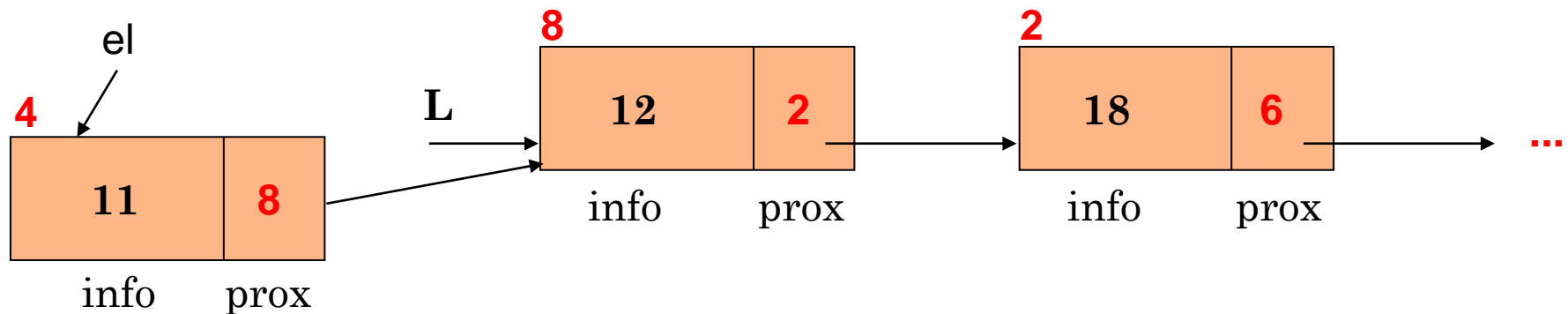
```
el = (lista *) malloc(sizeof(lista));
```

```
el->info = elem;
```

```
el->prox = L;
```

INSERINDO UM NÓ EM L (ELEM = 11)

○ Início da lista



```
el = (lista *) malloc(sizeof(lista));
```

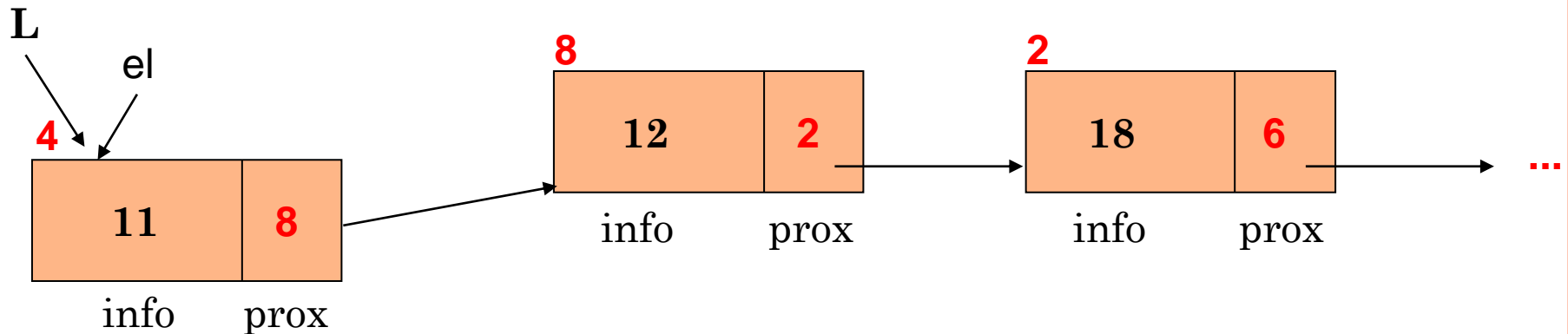
```
el->info = elem;
```

```
el->prox = L;
```

```
L = el;
```

INSERINDO UM NÓ EM L (ELEM = 11)

○ Início da lista



```
el = (lista *) malloc(sizeof(lista));
```

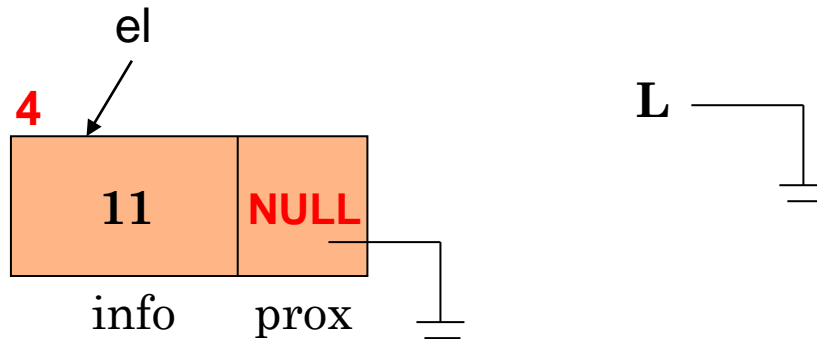
```
el->info = elem;
```

```
el->prox = L;
```

```
L = el;
```

INSERINDO UM NÓ EM L (ELEM = 11)

- Se lista vazia ($L == \text{NULL}$)



```
el = (lista *) malloc(sizeof(lista));
```

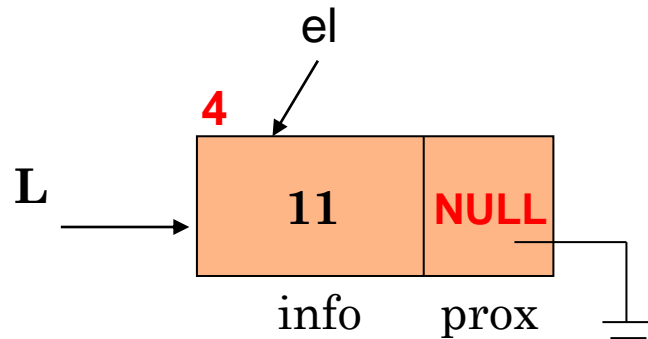
```
el->info = elem;
```

```
el->prox = L;
```

```
L = el;
```

INSERINDO UM NÓ EM L (ELEM = 11)

- Se lista vazia ($L == \text{NULL}$)



```
el = (lista *) malloc(sizeof(lista));
```

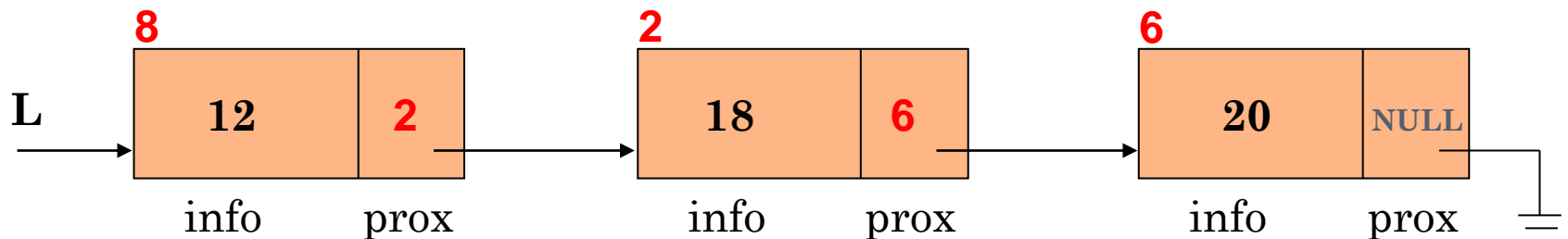
```
el->info = elem;
```

```
el->prox = L;
```

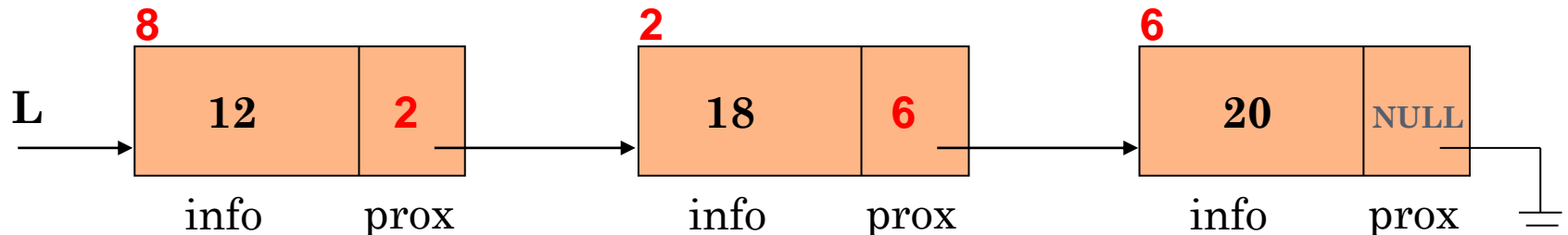
```
L = el;
```

INSERINDO UM NÓ EM L

- Se após buscaElem, return 0 e pre != NULL
 - Então, elemento será inserido no meio ou no final da lista
- Meio da lista (elem = 14)



- Final da lista (elem = 22)



INSERINDO UM NÓ EM L (ELEM = 14)

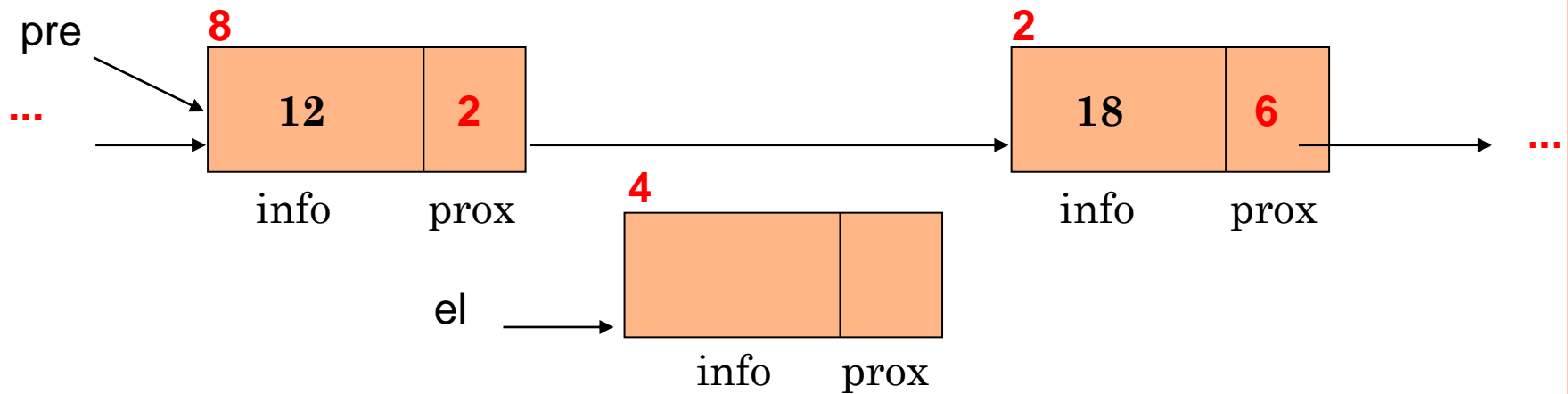
○ Meio da lista



```
el = (lista *) malloc(sizeof(lista));
```

INSERINDO UM NÓ EM L (ELEM = 14)

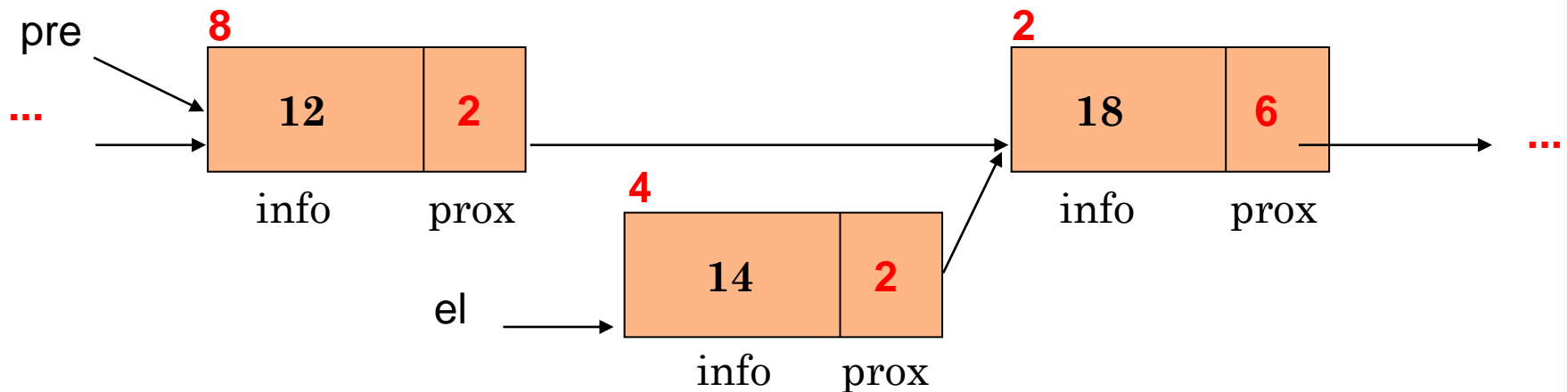
○ Meio da lista



```
el = (lista *) malloc(sizeof(lista));
```


INSERINDO UM NÓ EM L (ELEM = 14)

o Meio da lista



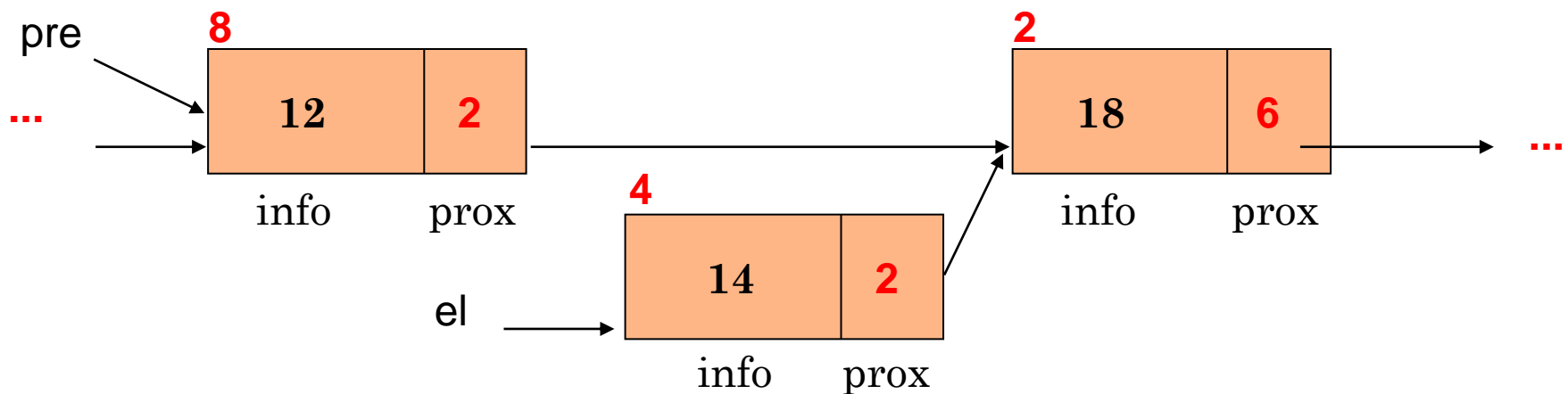
```
el = (lista *) malloc(sizeof(lista));
```

```
el->info = elem;
```

```
el->prox = pre->prox;
```

INSERINDO UM NÓ EM L (ELEM = 14)

○ Meio da lista



```
el = (lista *) malloc(sizeof(lista));
```

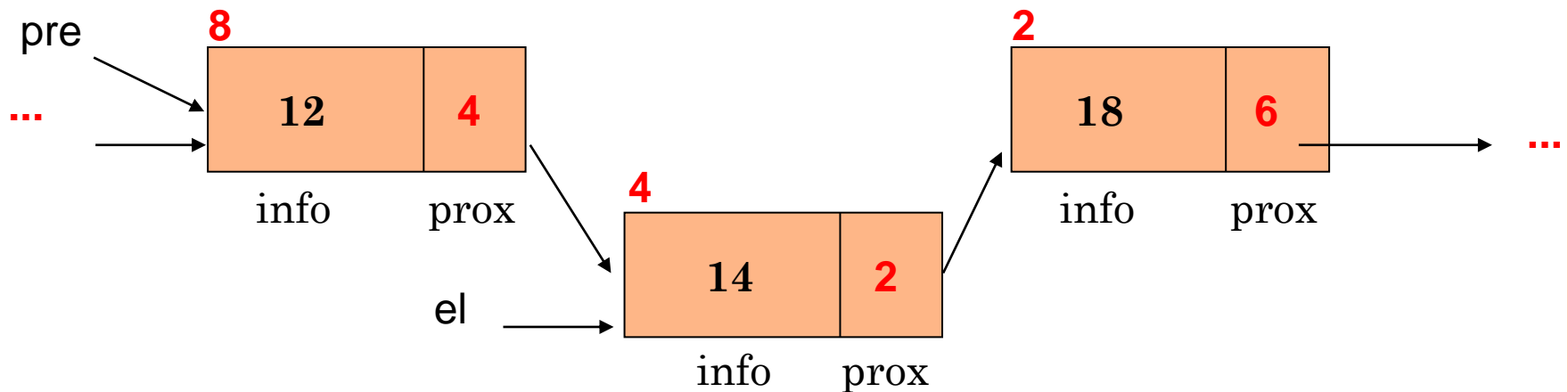
```
el->info = elem;
```

```
el->prox = pre->prox;
```

```
pre->prox = el;
```

INSERINDO UM NÓ EM L (ELEM = 14)

o Meio da lista



```
el = (lista *) malloc(sizeof(lista));
```

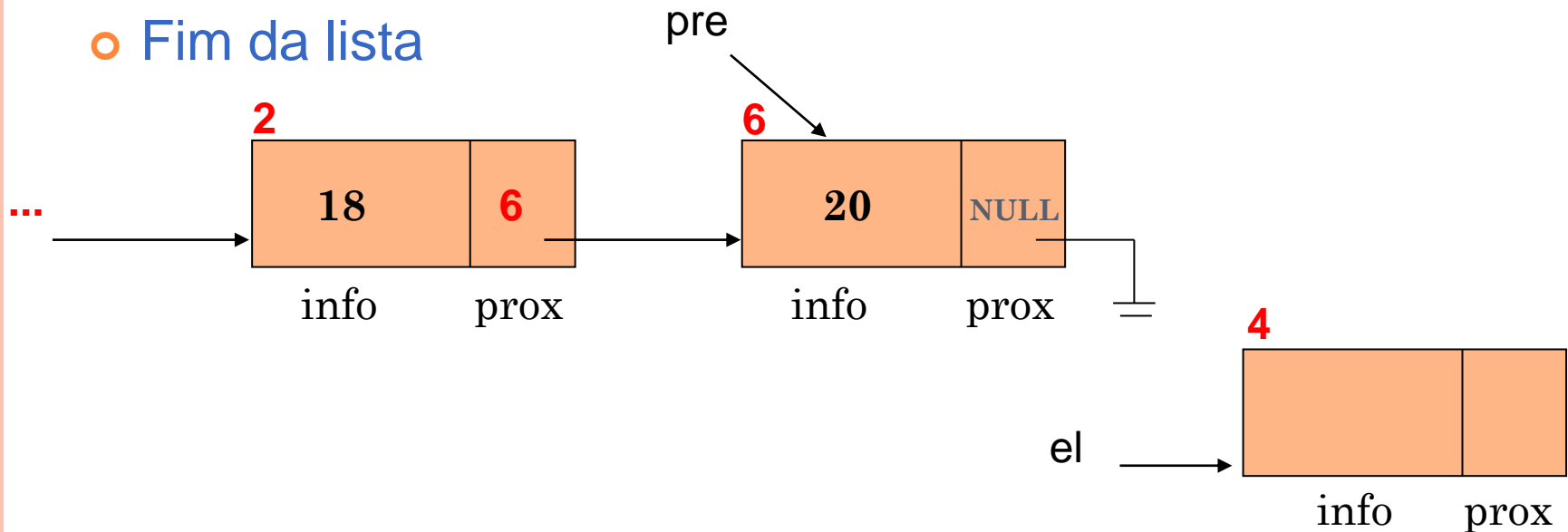
```
el->info = elem;
```

```
el->prox = pre->prox;
```

```
pre->prox = el;
```

INSERINDO UM NÓ EM L (ELEM = 22)

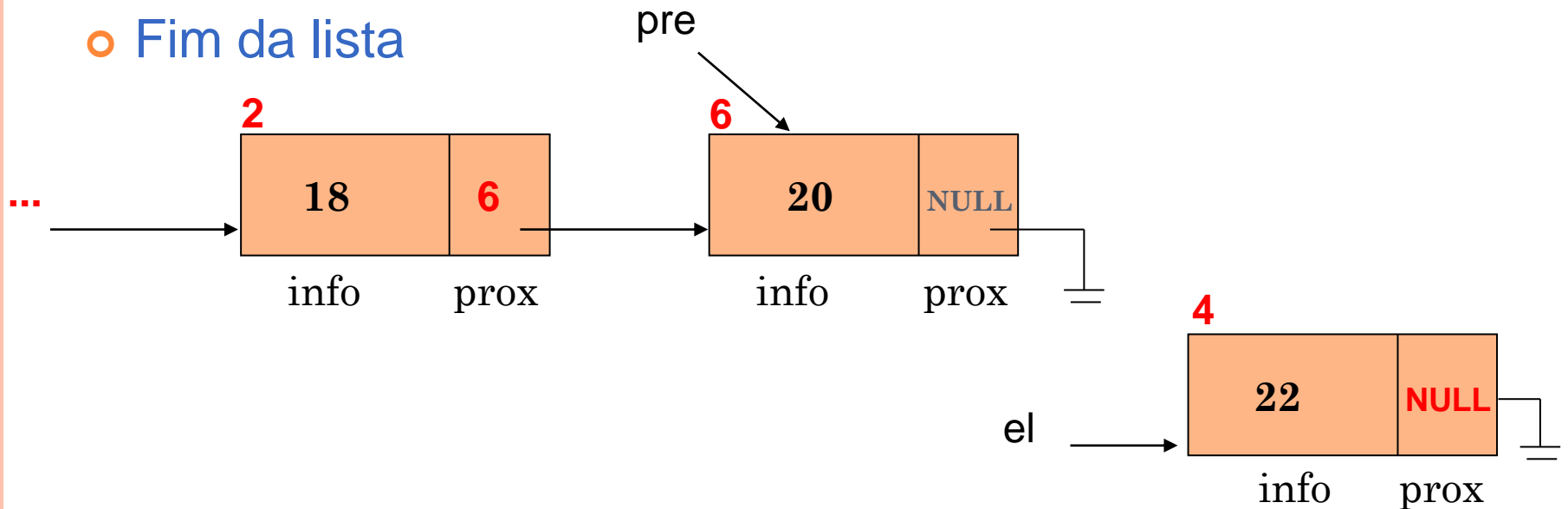
○ Fim da lista



```
el = (lista *) malloc(sizeof(lista));
```

INSERINDO UM NÓ EM L (ELEM = 22)

○ Fim da lista



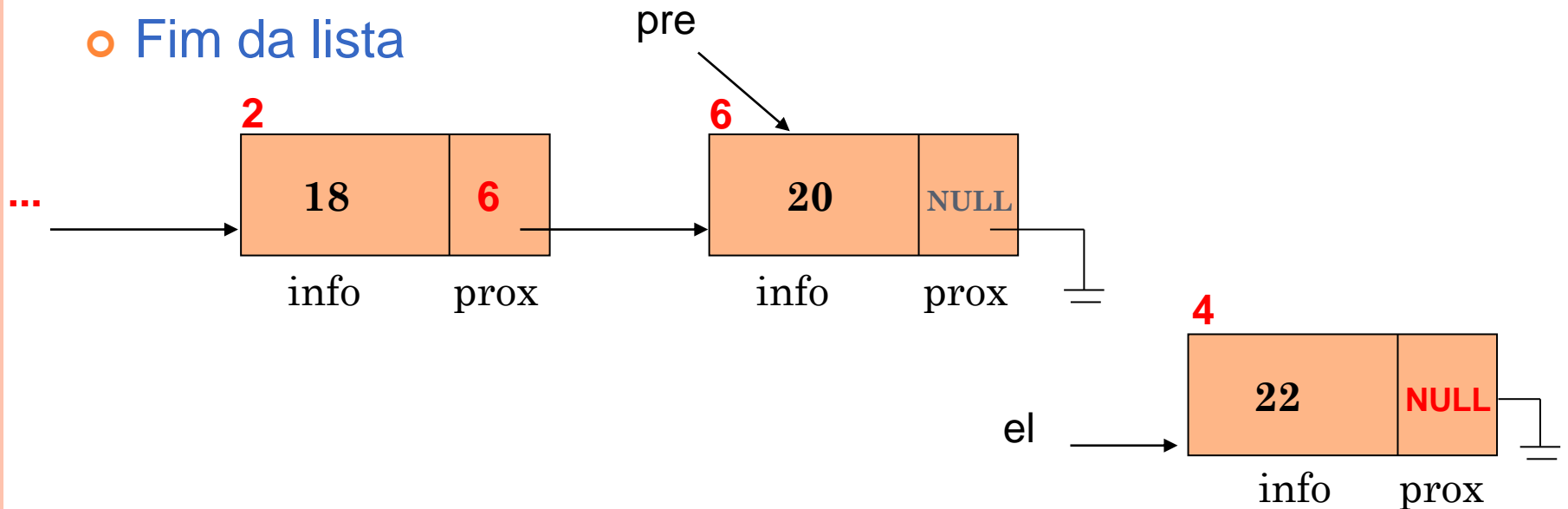
```
el = (lista *) malloc(sizeof(lista));
```

```
el->info = elem;
```

```
el->prox = pre->prox;
```

INSERINDO UM NÓ EM L (ELEM = 22)

○ Fim da lista



```
el = (lista *) malloc(sizeof(lista));
```

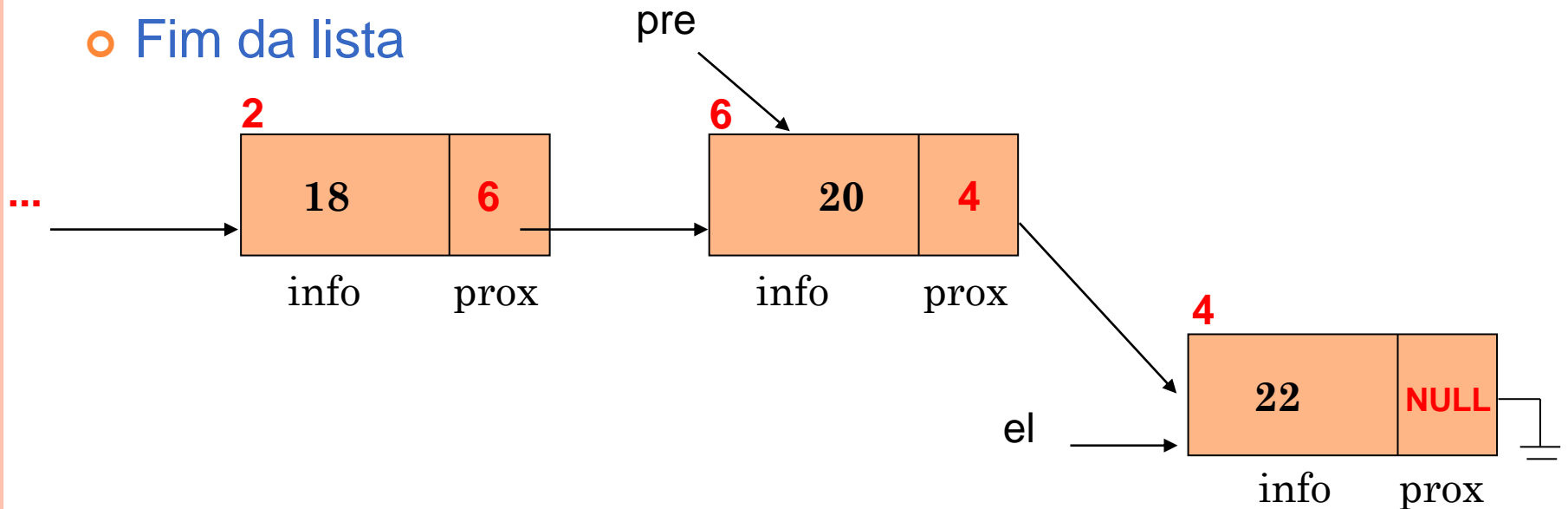
```
el->info = elem;
```

```
el->prox = pre->prox;
```

```
pre->prox = el;
```

INSERINDO UM NÓ EM L (ELEM = 22)

○ Fim da lista



```
el = (lista *) malloc(sizeof(lista));
```

```
el->info = elem;
```

```
el->prox = pre->prox;
```

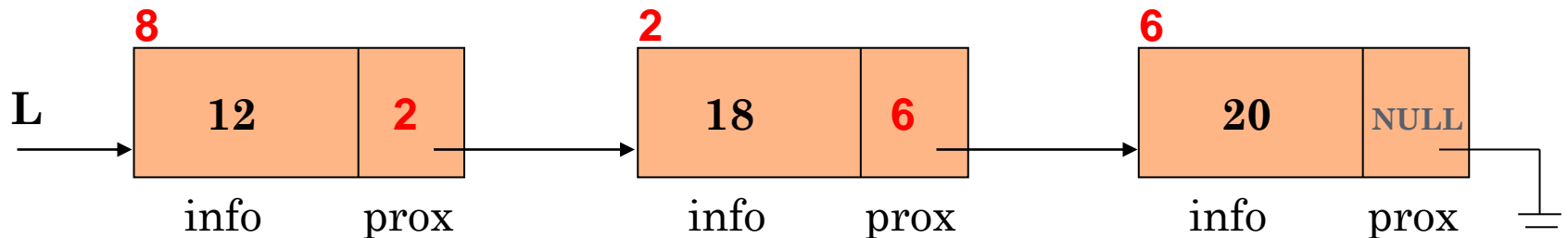
```
pre->prox = el;
```

INSERINDO UM NÓ EM L

```
lista *insereElem(lista *L, int elem) {  
    lista *pre, *el;  
    if (!buscaElem(L,elem,&pre)) {  
        el = (lista *)malloc(sizeof(lista));  
        el->info = elem;  
        if (L == NULL || pre == NULL) {  
            el->prox = L;  
            L = el;  
        } else {  
            el->prox = pre->prox;  
            pre->prox = el;  
        }  
    }  
    return L;  
}
```


REMOVENDO UM NÓ DE L

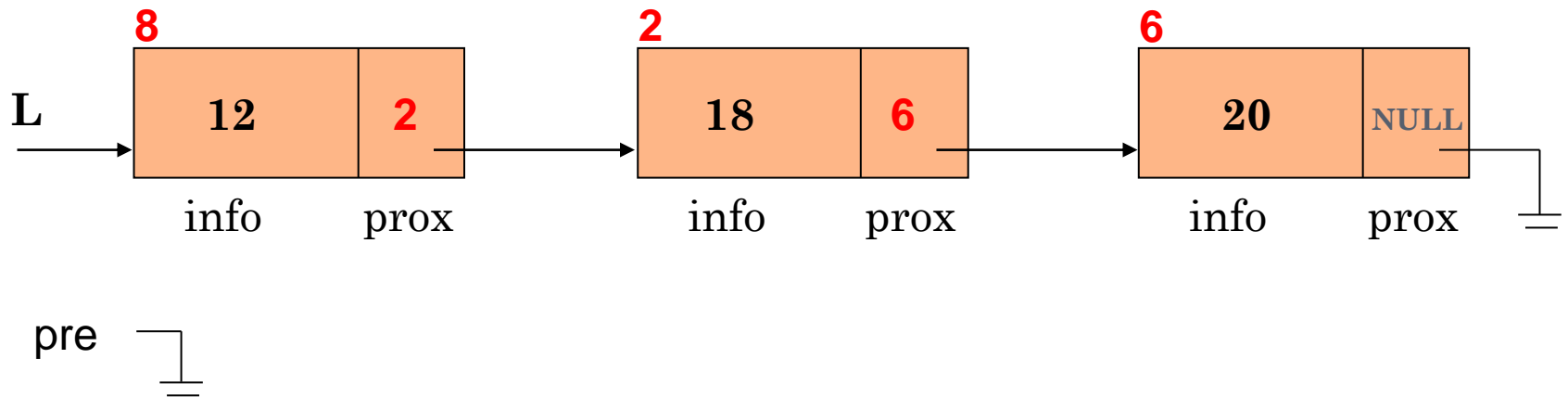
- Considere a lista ordenada de inteiros, com endereço inicial guardado em L



- O elemento só será inserido na lista L se buscaElem retornar 1, ou seja, quando o elemento for encontrado
- O elemento a ser removido, encontra-se após o nó pre, cujo valor retornou de buscaElem

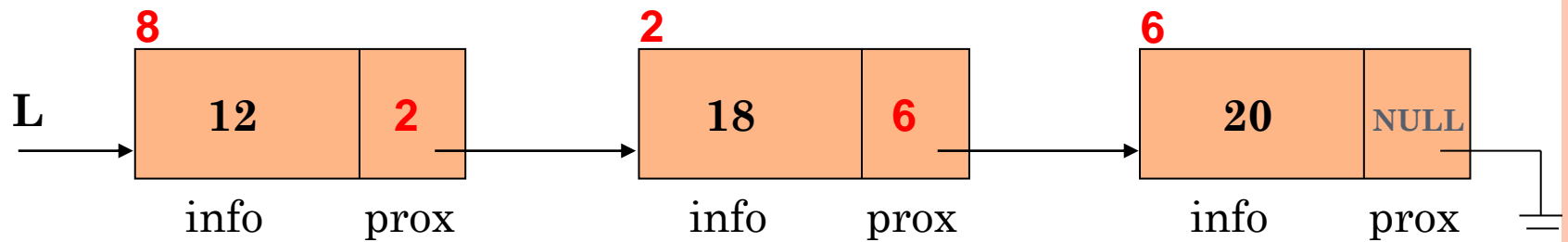
REMOVENDO UM NÓ DE L

- Se após buscaElem, return 1 e pre == NULL
 - Então, elemento será removido do início da lista
 - Início da lista (elem = 12)



REMOVENDO UM NÓ DE L (ELEM = 12)

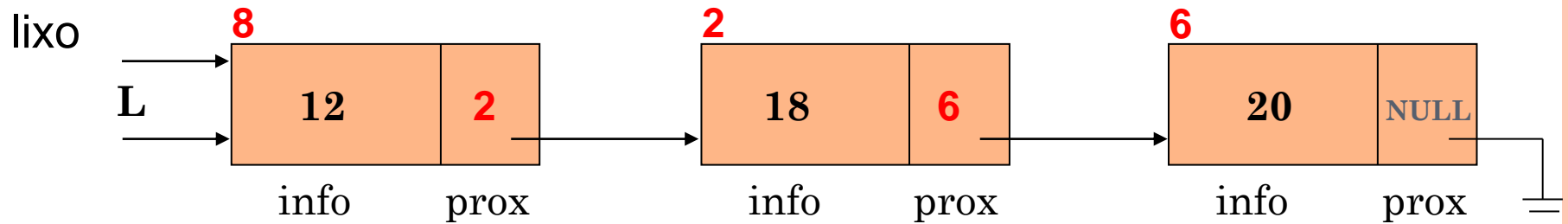
- Início da lista



lixo = L;

REMOVENDO UM NÓ DE L (ELEM = 12)

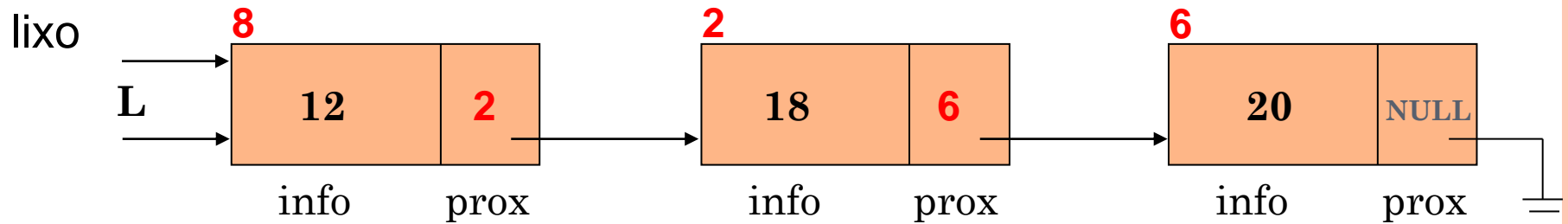
○ Início da lista



lixo = L;

REMOVENDO UM NÓ DE L (ELEM = 12)

○ Início da lista

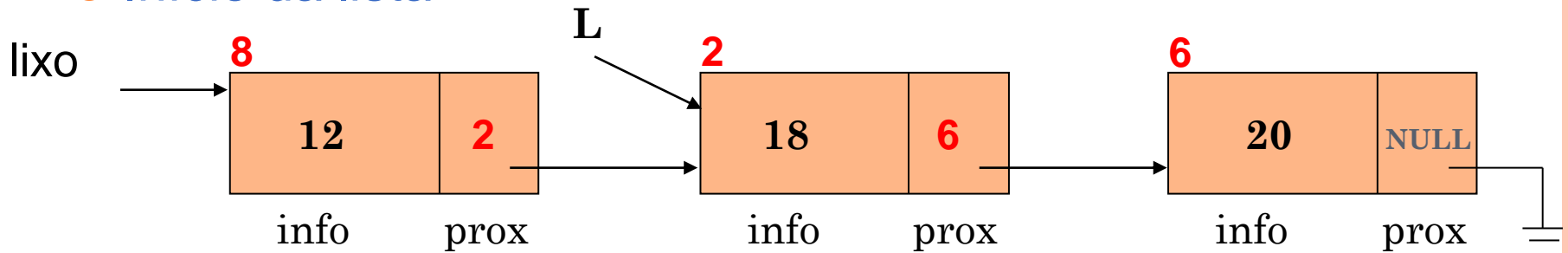


lixo = L;

L = L->prox;

REMOVENDO UM NÓ DE L (ELEM = 12)

○ Início da lista

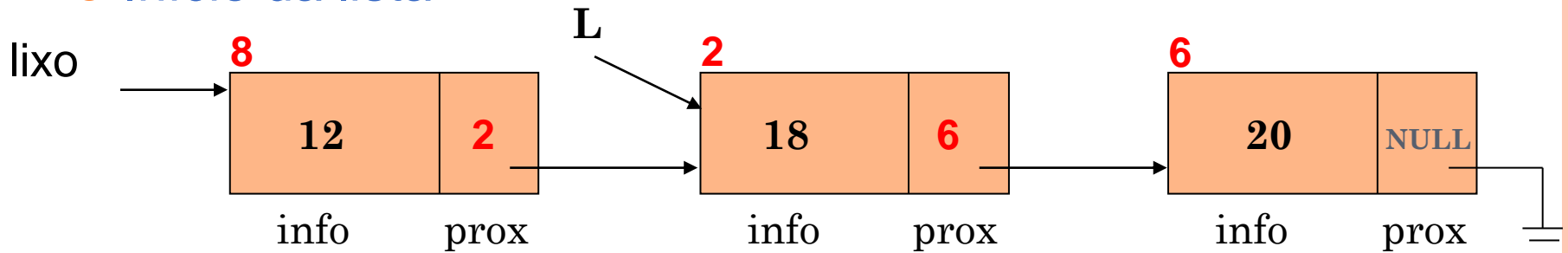


lixo = L;

L = L->prox;

REMOVENDO UM NÓ DE L (ELEM = 12)

○ Início da lista



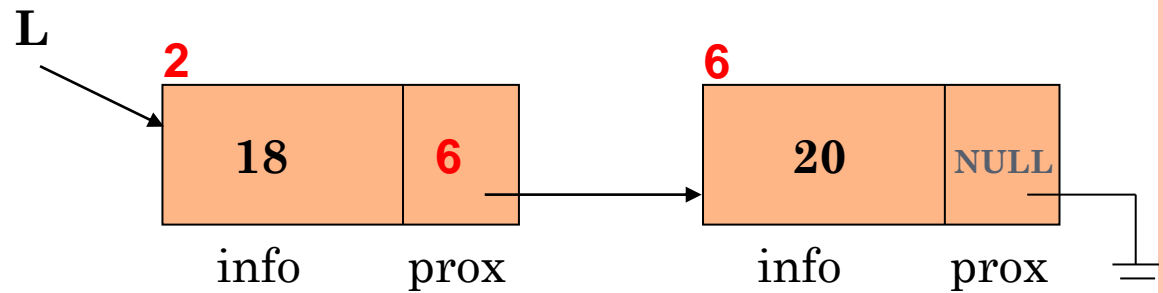
```
lixo = L;
```

```
L = L->prox;
```

```
free(lixo);
```

REMOVENDO UM NÓ DE L (ELEM = 12)

- Início da lista



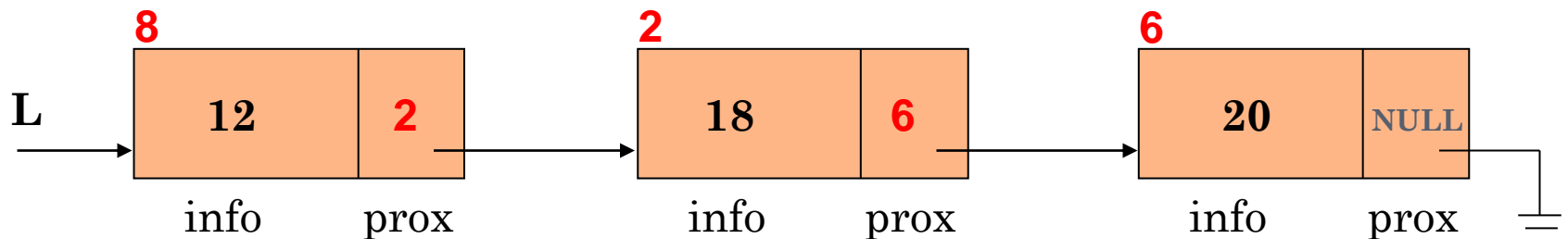
```
lixo = L;
```

```
L = L->prox;
```

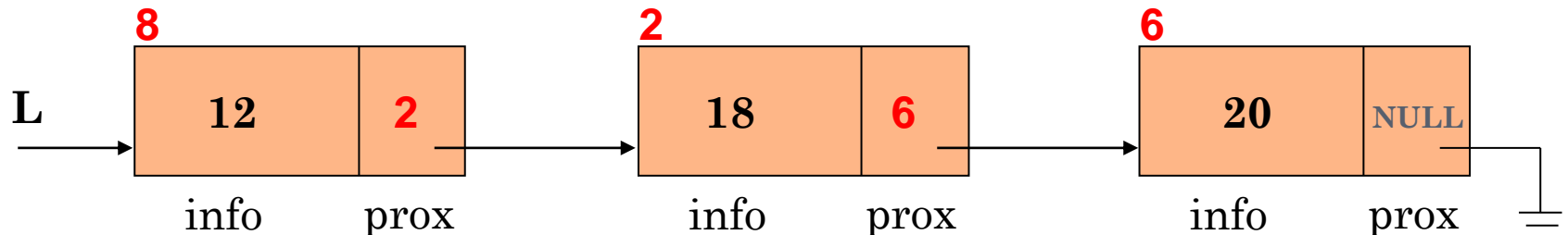
```
free(lixo);
```


REMOVENDO UM NÓ DE L

- Se após buscaElem, return 1 e pre != NULL
 - Então, elemento será removido do meio ou do final da lista
- Meio da lista (elem = 18)

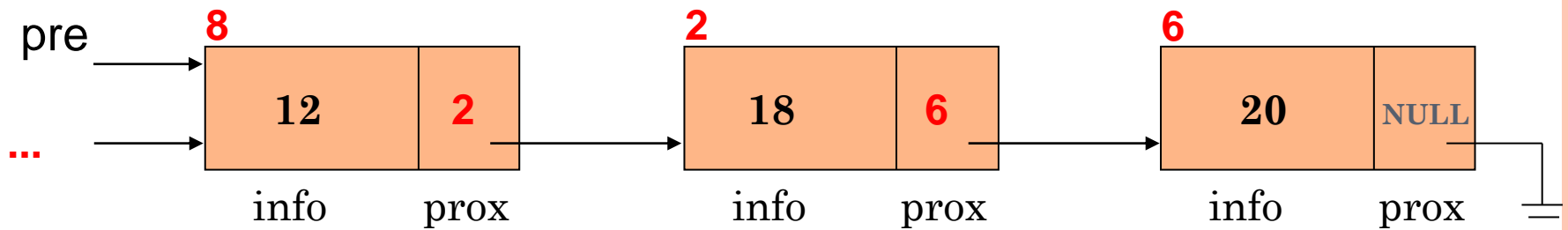


- Final da lista (elem = 20)



REMOVENDO UM NÓ DE L (ELEM = 18)

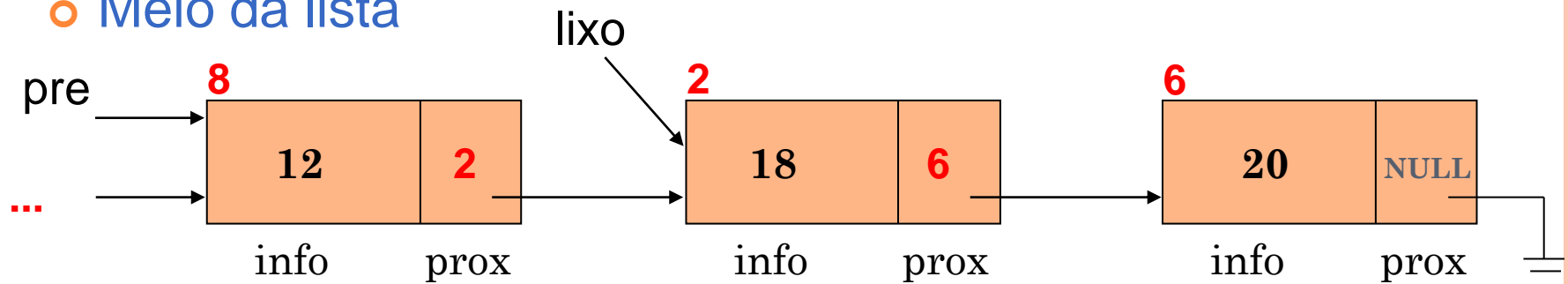
○ Meio da lista



```
lixo = pre->prox;
```

REMOVENDO UM NÓ DE L (ELEM = 18)

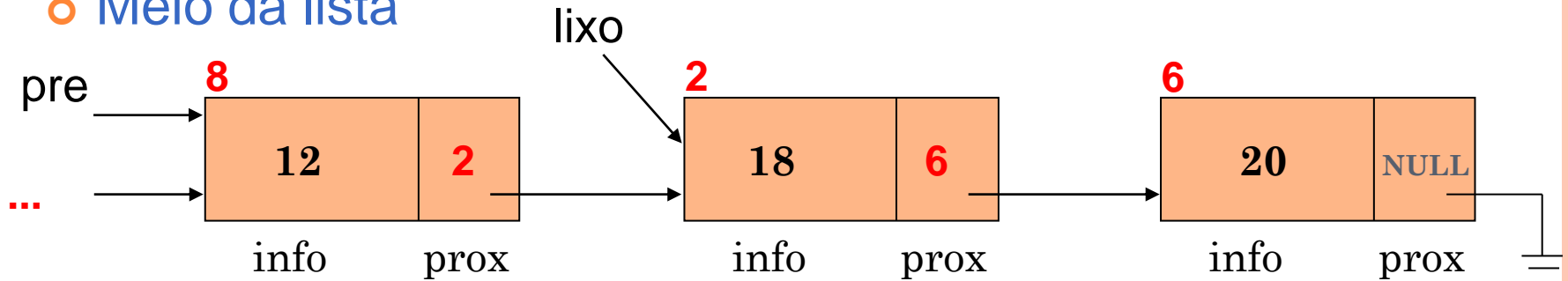
○ Meio da lista



```
lixo = pre->prox;
```

REMOVENDO UM NÓ DE L (ELEM = 18)

○ Meio da lista

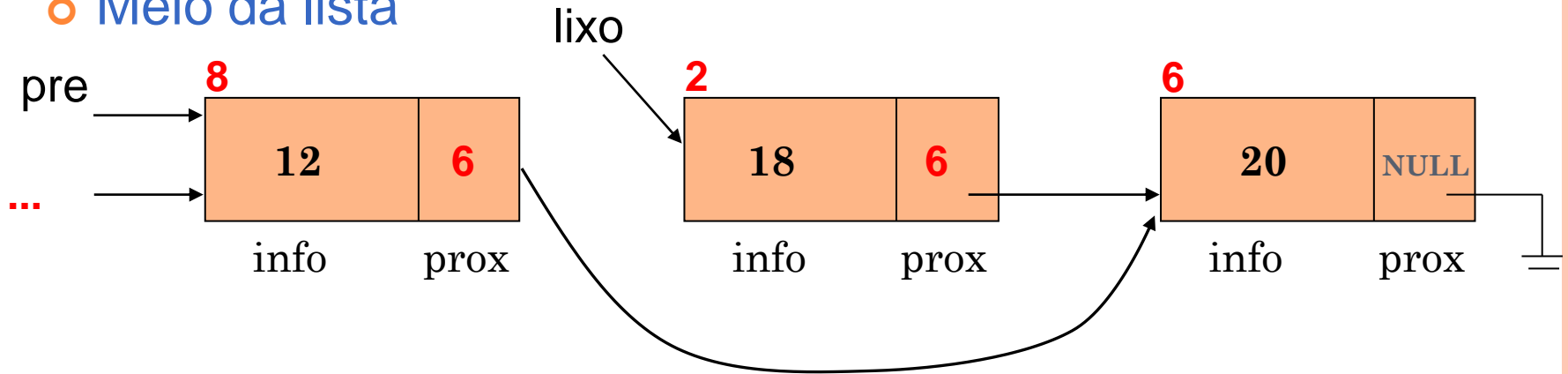


lixo = pre->prox;

pre->prox = lixo->prox;

REMOVENDO UM NÓ DE L (ELEM = 18)

○ Meio da lista

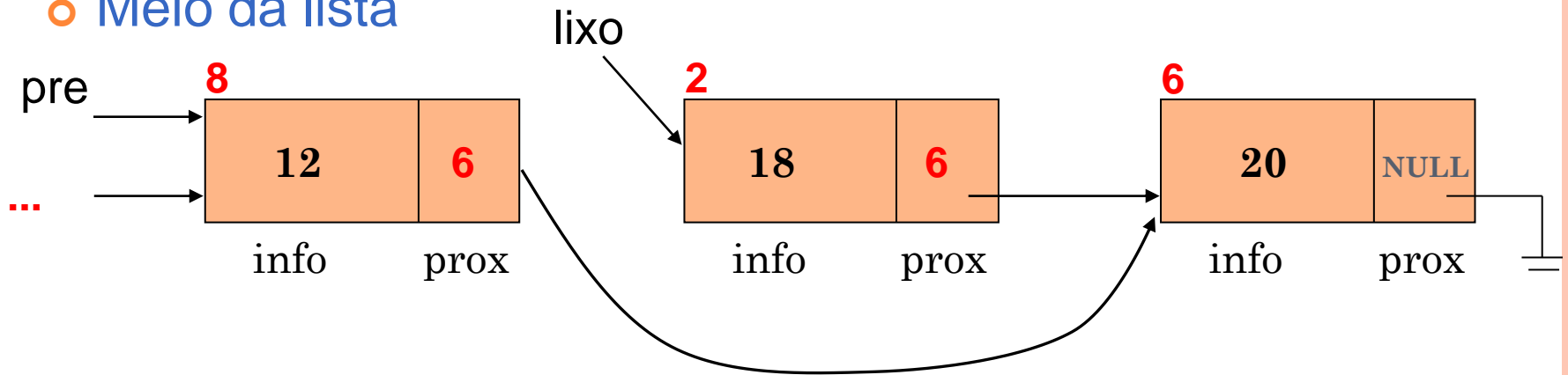


`lixo = pre->prox;`

`pre->prox = lixo->prox;`

REMOVENDO UM NÓ DE L (ELEM = 18)

○ Meio da lista



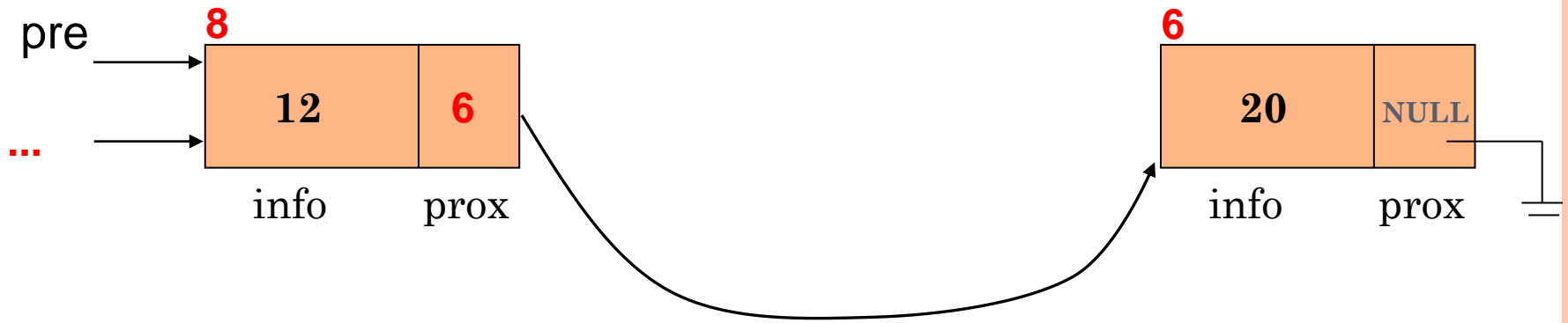
```
lixo = pre->prox;
```

```
pre->prox = lixo->prox;
```

```
free(lixo);
```

REMOVENDO UM NÓ DE L (ELEM = 18)

○ Meio da lista



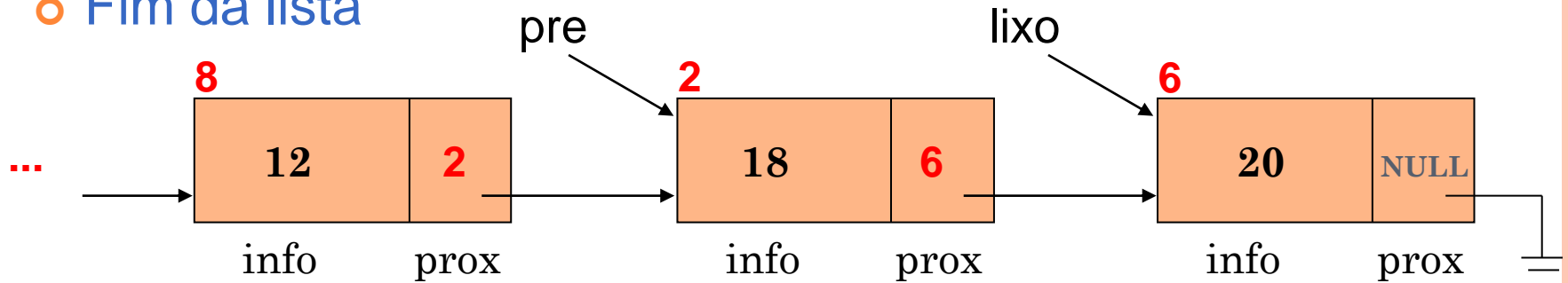
```
lixo = pre->prox;
```

```
pre->prox = lixo->prox;
```

```
free(lixo);
```

REMOVENDO UM NÓ DE L (ELEM = 20)

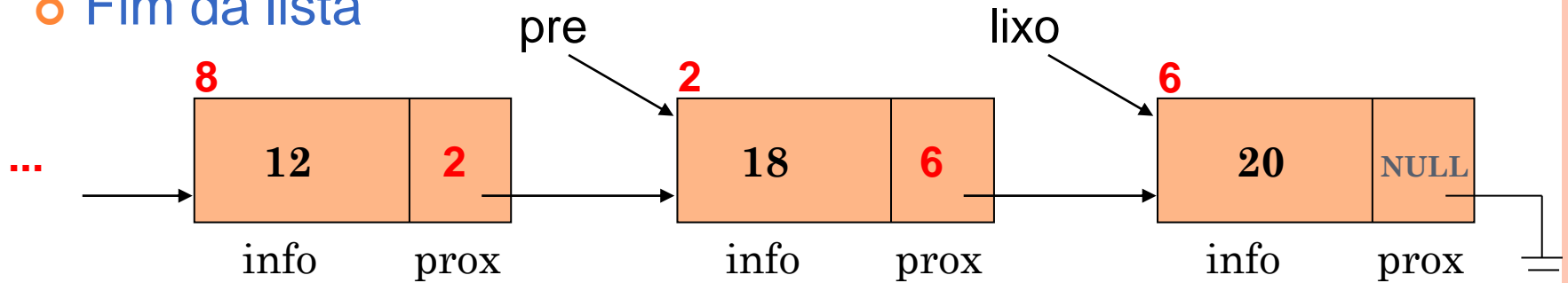
- Fim da lista



`lixo = pre->prox;`

REMOVENDO UM NÓ DE L (ELEM = 20)

○ Fim da lista

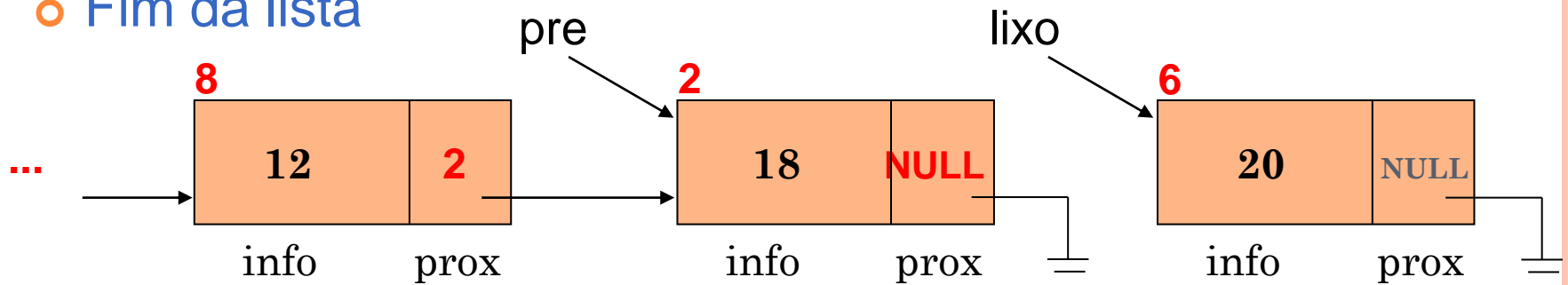


lixo = pre->prox;

pre->prox = lixo->prox;

REMOVENDO UM NÓ DE L (ELEM = 20)

○ Fim da lista

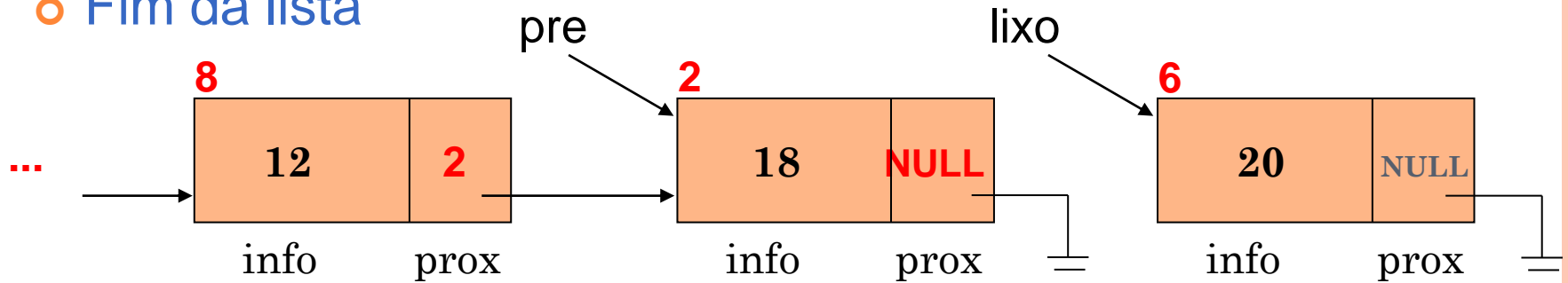


lixo = pre->prox;

pre->prox = lixo->prox;

REMOVENDO UM NÓ DE L (ELEM = 20)

○ Fim da lista



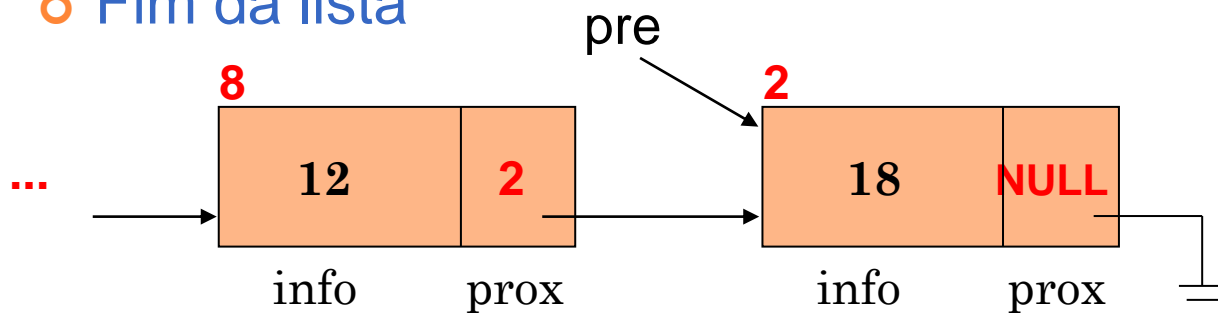
```
lixo = pre->prox;
```

```
pre->prox = lixo->prox;
```

```
free(lixo);
```

REMOVENDO UM NÓ DE L (ELEM = 20)

○ Fim da lista



```
lixo = pre->prox;
```

```
pre->prox = lixo->prox;
```

```
free(lixo);
```

REMOVENDO UM NÓ DE L

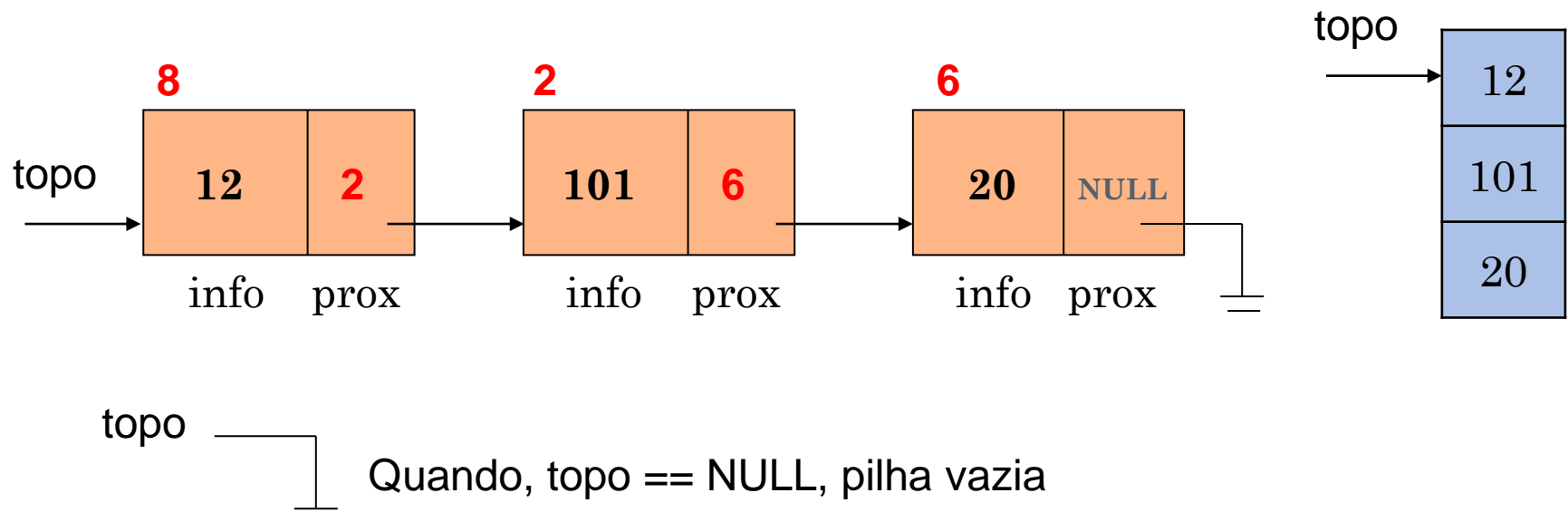
```
lista *removeElem(lista *L, int elem) {  
    lista *pre, *lixo;  
    if (buscaElem(L,elem,&pre)) {  
        if (L->info == elem) {  
            lixo = L;  
            L = L->prox;  
        } else {  
            lixo = pre->prox;  
            pre->prox = lixo->prox;  
        }  
        free(lixo);  
    }  
    return L;  
}
```



PILHAS E FILAS COM ALOCAÇÃO ENCADEADA

PILHAS

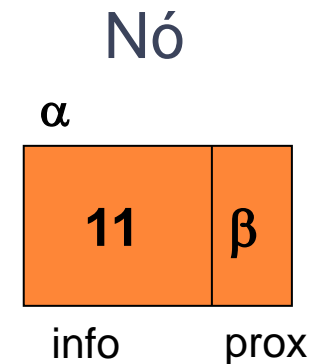
- Inclusões e Remoções de nós são realizadas em um único extremo: TOPO
 - Simplificar os algoritmos de Inclusão e Remoção de nós
- Considere uma pilha de números inteiros



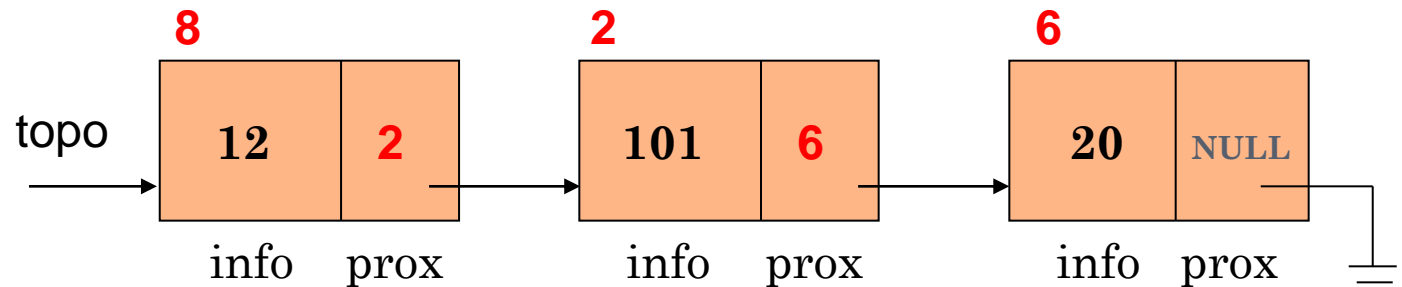
PILHAS

- Declaração de uma pilha de números inteiros

```
struct NO {  
    int info;  
    struct NO *prox;  
}  
typedef struct NO pilha;  
  
...  
pilha *topo;  
topo = NULL;
```

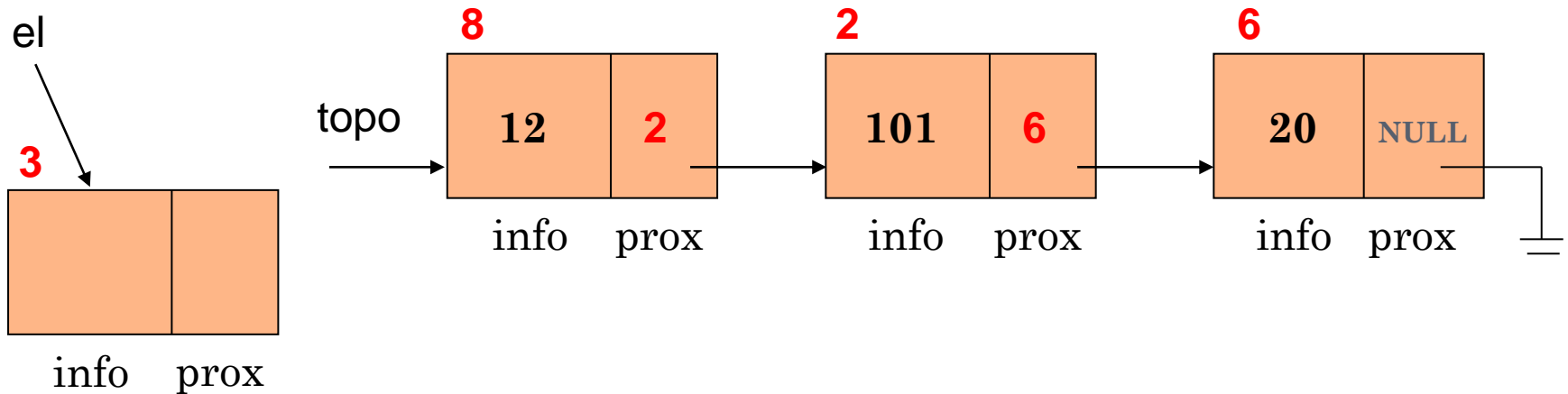


INSERINDO UM NÓ NA PILHA (ELEM = 31)



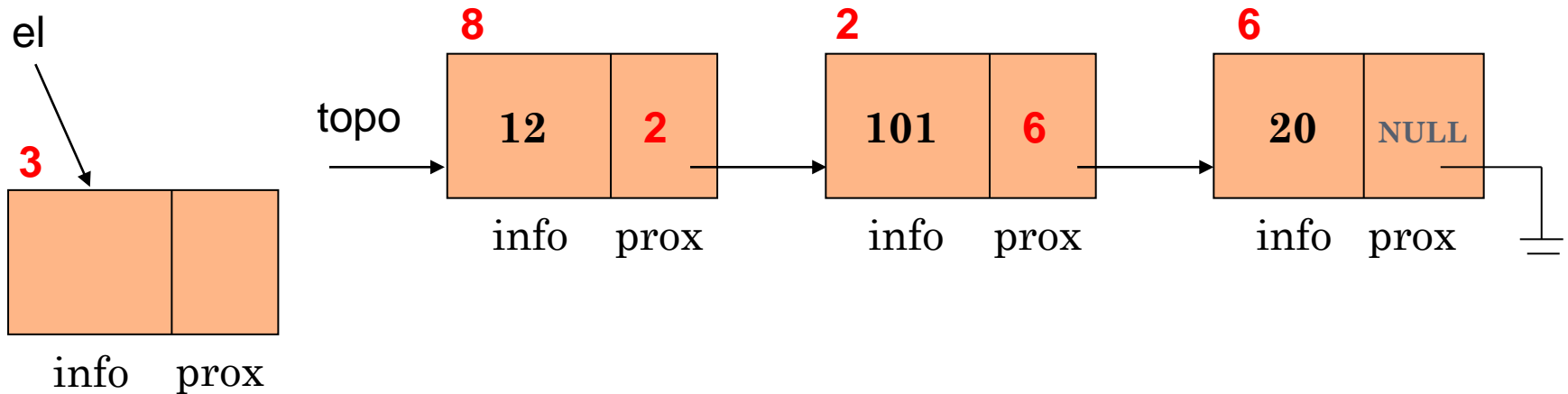
```
el = (pilha *) malloc(sizeof(pilha));
```

INSERINDO UM NÓ NA PILHA (ELEM = 31)



```
el = (pilha *) malloc(sizeof(pilha));
```

INSERINDO UM NÓ NA PILHA (ELEM = 31)

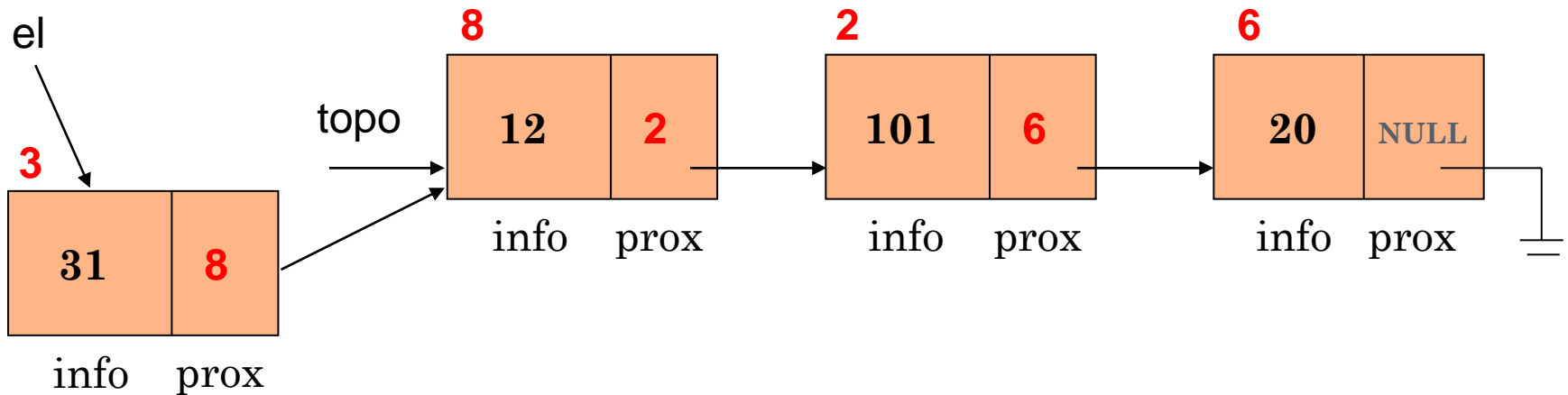


```
el = (pilha *) malloc(sizeof(pilha));
```

```
el->info = elem;
```

```
el->prox = topo;
```

INSERINDO UM NÓ NA PILHA (ELEM = 31)

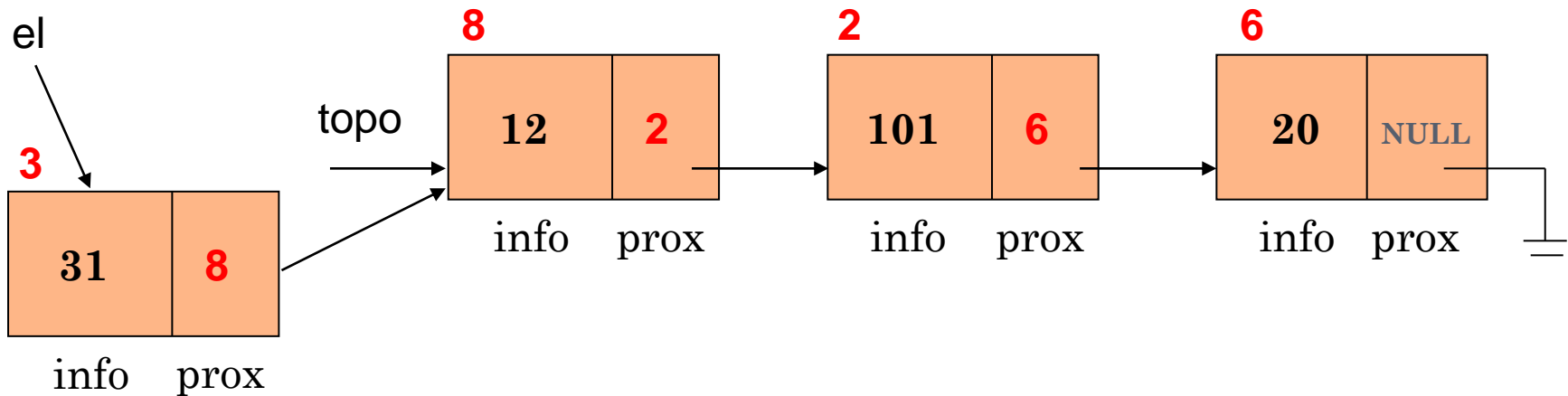


```
el = (pilha *) malloc(sizeof(pilha));
```

```
el->info = elem;
```

```
el->prox = topo;
```

INSERINDO UM NÓ NA PILHA (ELEM = 31)



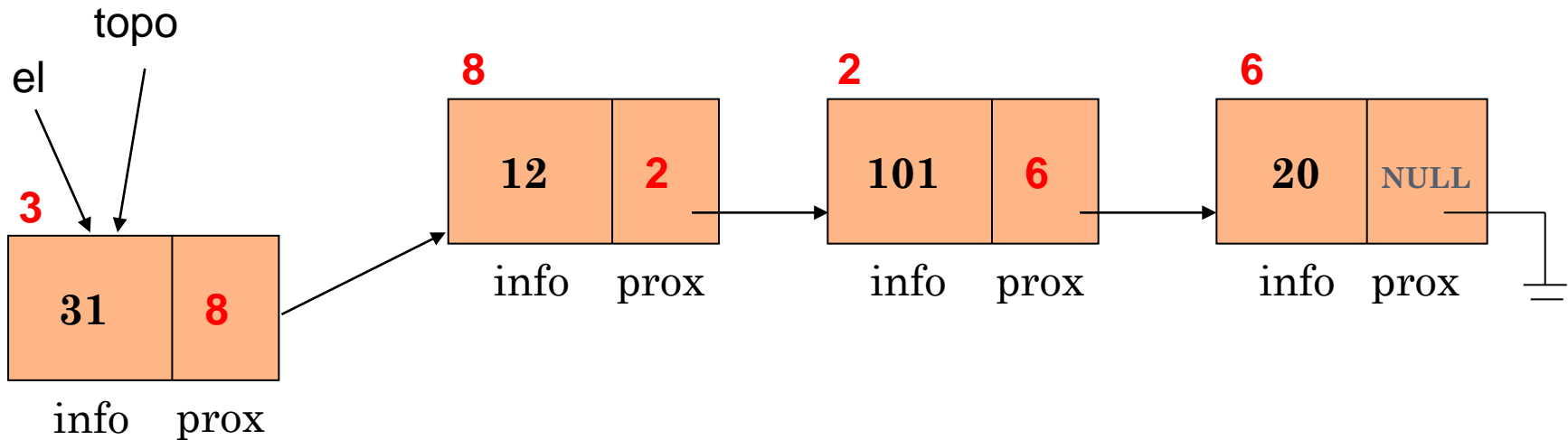
```
el = (pilha *) malloc(sizeof(pilha));
```

```
el->info = elem;
```

```
el->prox = topo;
```

```
topo = el;
```

INSERINDO UM NÓ NA PILHA (ELEM = 31)



```
el = (pilha *) malloc(sizeof(pilha));
```

```
el->info = elem;
```

```
el->prox = topo;
```

```
topo = el;
```

INSERINDO UM NÓ NA PILHA

```

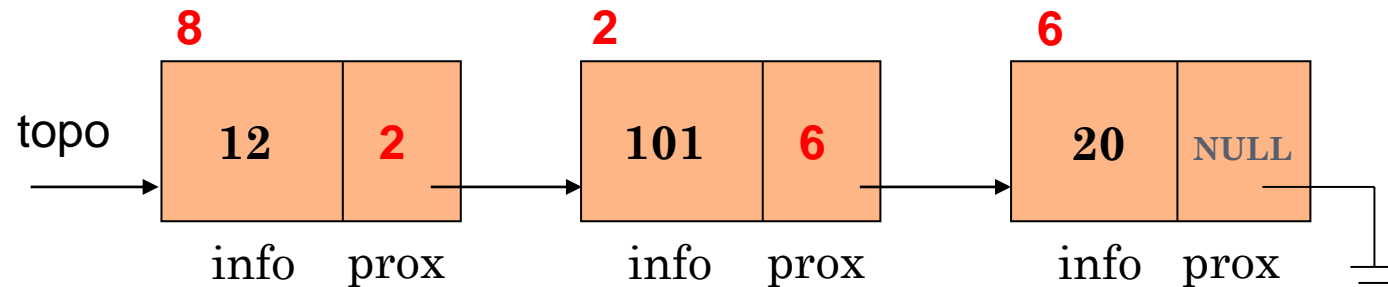
pilha *push(pilha *topo, int elem){

    pilha *el;

    el = (pilha*) malloc(sizeof(pilha));
    el->info = elem;
    el->prox = topo;
    topo = el;
    return topo;

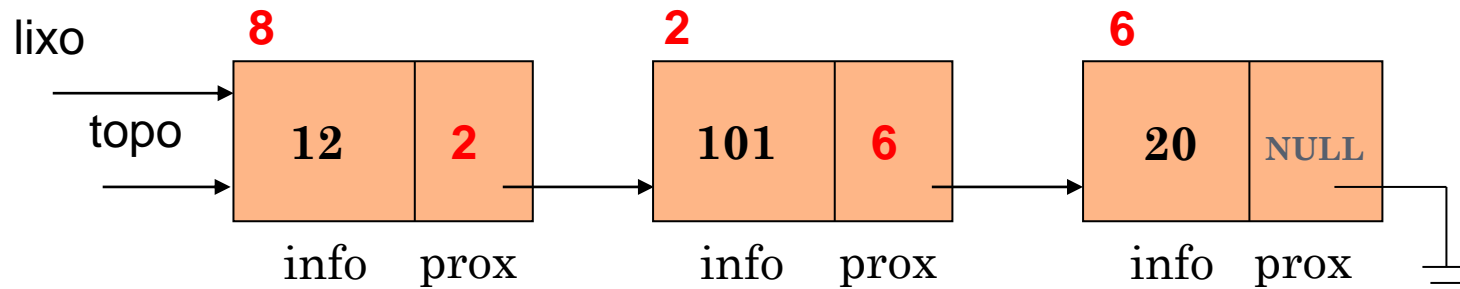
}
```

REMOVENDO UM NÓ DA PILHA



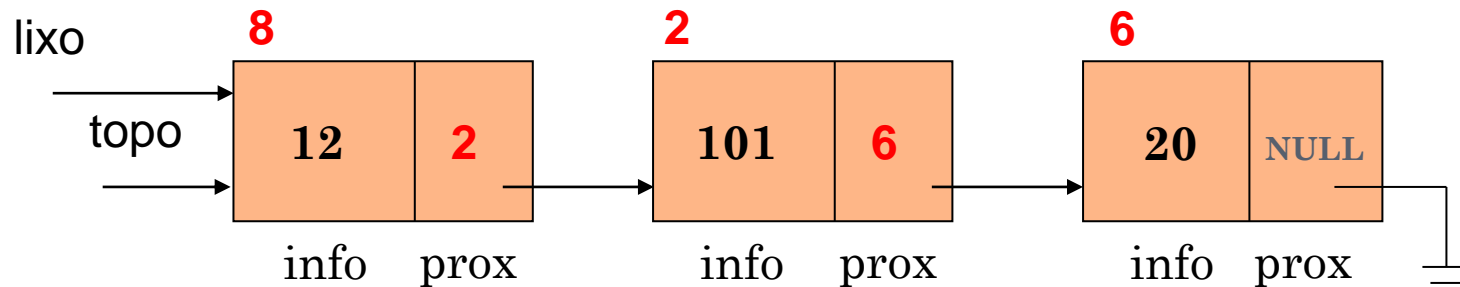
lixo = topo;

REMOVENDO UM NÓ DA PILHA



`lixo = topo;`

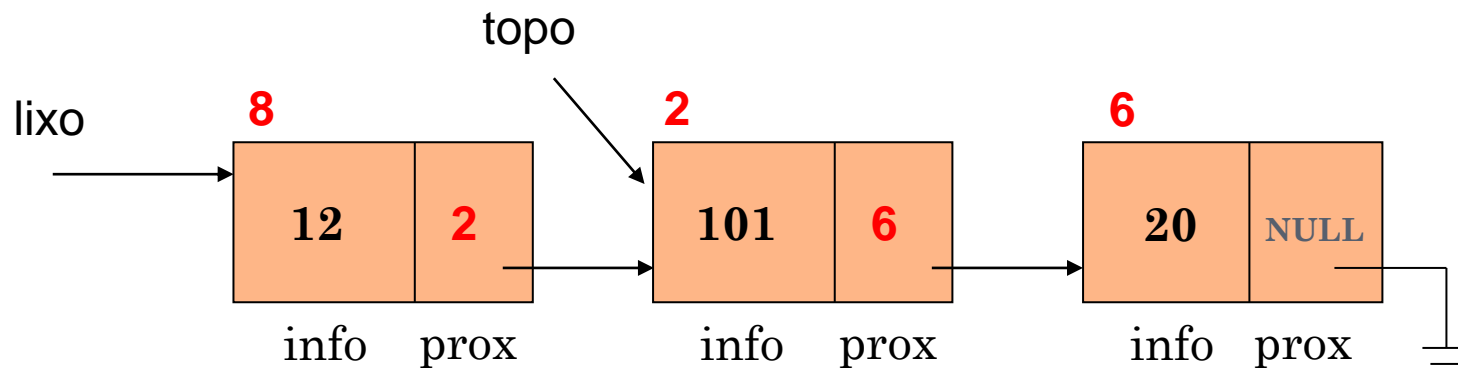
REMOVENDO UM NÓ DA PILHA



`lixo = topo;`

`topo = topo->prox;`

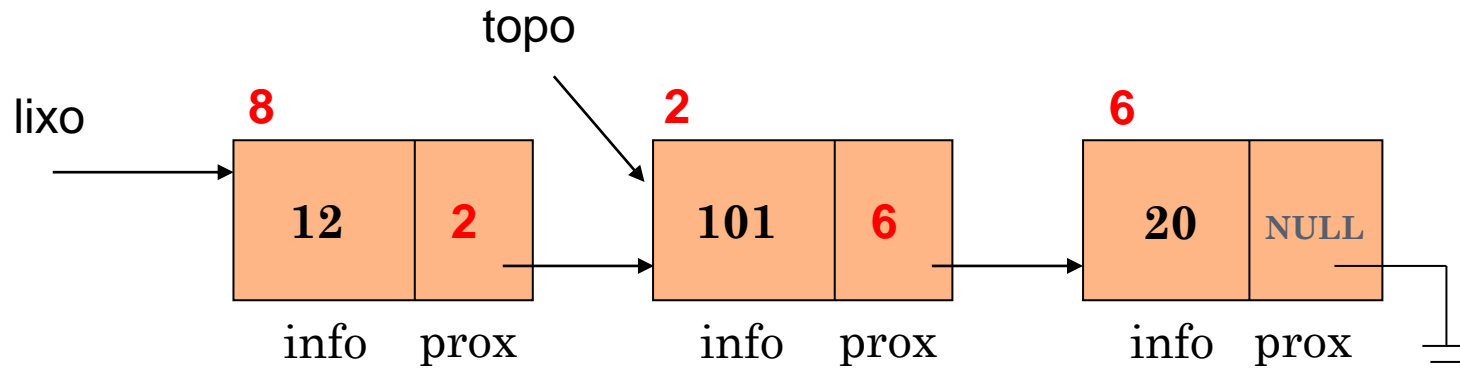
REMOVENDO UM NÓ DA PILHA



`lixo = topo;`

`topo = topo->prox;`

REMOVENDO UM NÓ DA PILHA

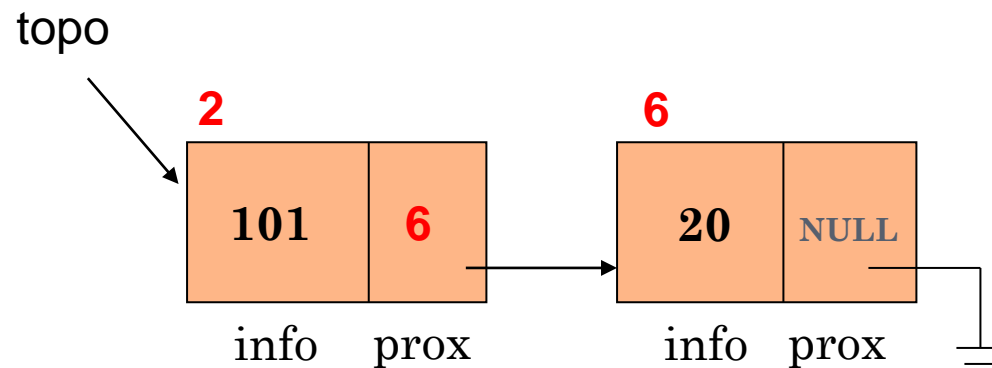


```
lixo = topo;
```

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

```
free(lixo);
```

REMOVENDO UM NÓ DA PILHA



```
lixo = topo;
```

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

```
free(lixo);
```

REMOVENDO UM NÓ DA PILHA

```

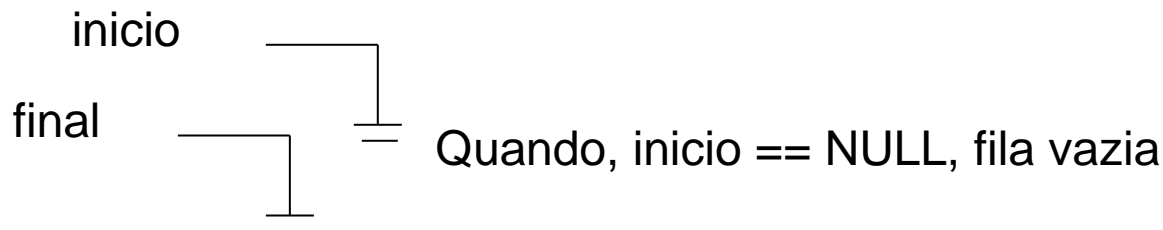
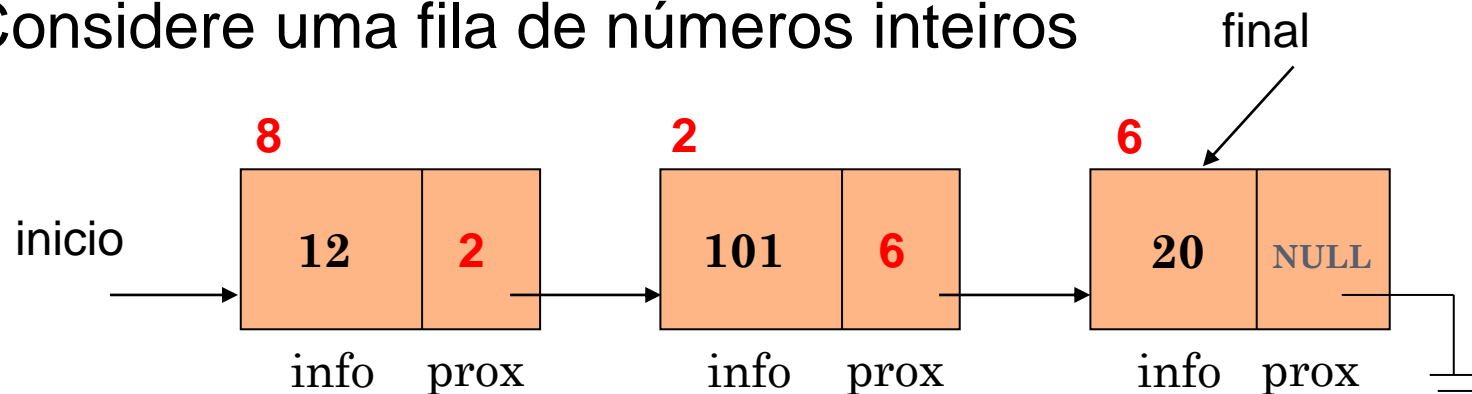
pilha *pop(pilha *topo){

    pilha *lixo;

    if (topo != NULL) {
        lixo = topo;
        topo = topo->prox;
        free(lixo);
    }
    return topo;
}
```

FILAS

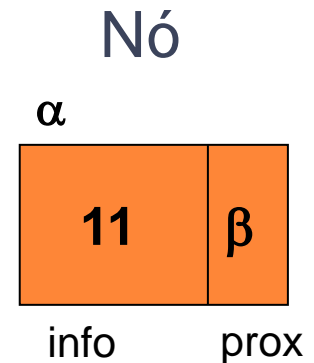
- Inclusões de nós são realizadas em um extremo (FINAL) e as Remoções no outro extremo (INÍCIO)
 - Simplificar os algoritmos de Inclusão e Remoção de nós
- Considere uma fila de números inteiros



FILAS

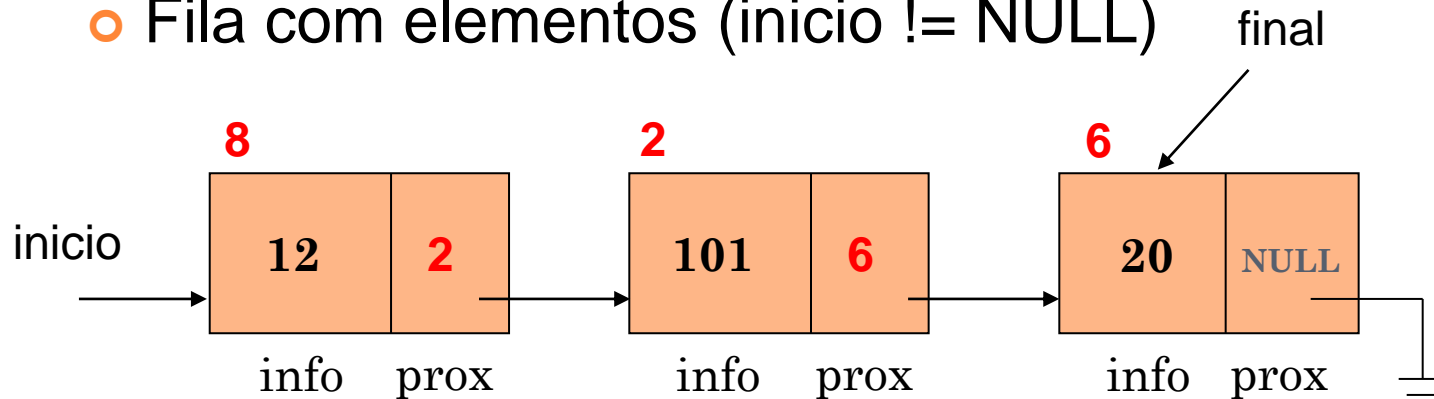
- Declaração de uma fila de números inteiros

```
struct NO {  
    int info;  
    struct NO *prox;  
}  
typedef struct NO fila;  
  
...  
fila *inicio, *final;  
inicio = NULL;  
final = NULL;
```



INSERINDO UM NÓ NA FILA(ELEM = 31)

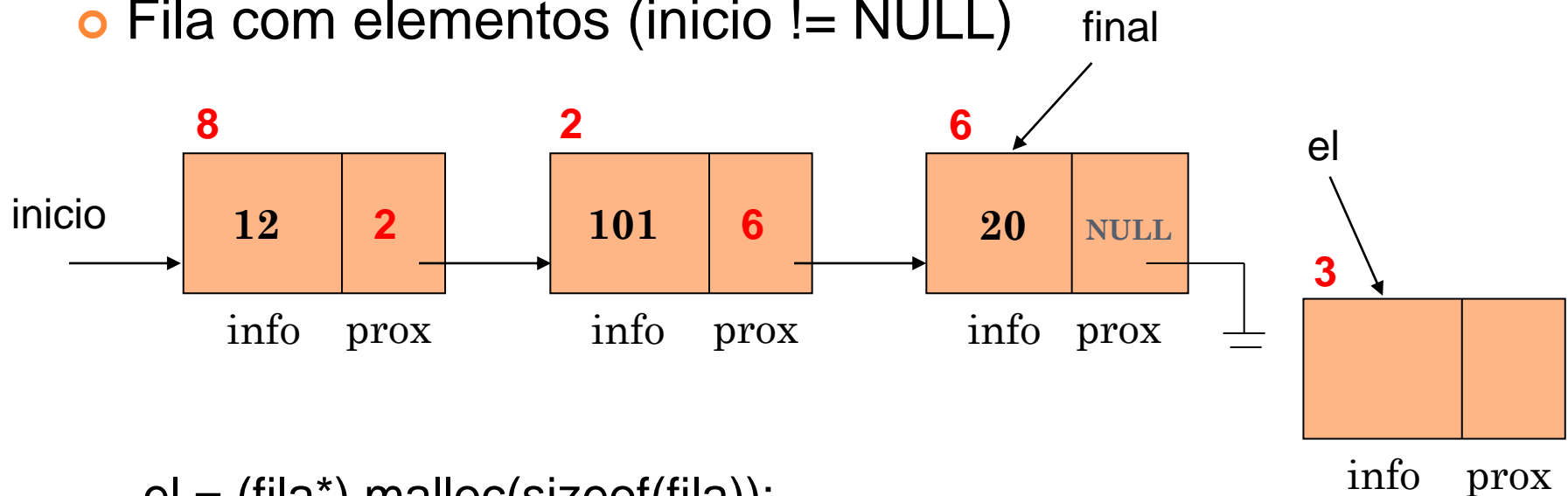
- Fila com elementos (inicio != NULL)



```
el = (fila*) malloc(sizeof(fila));
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

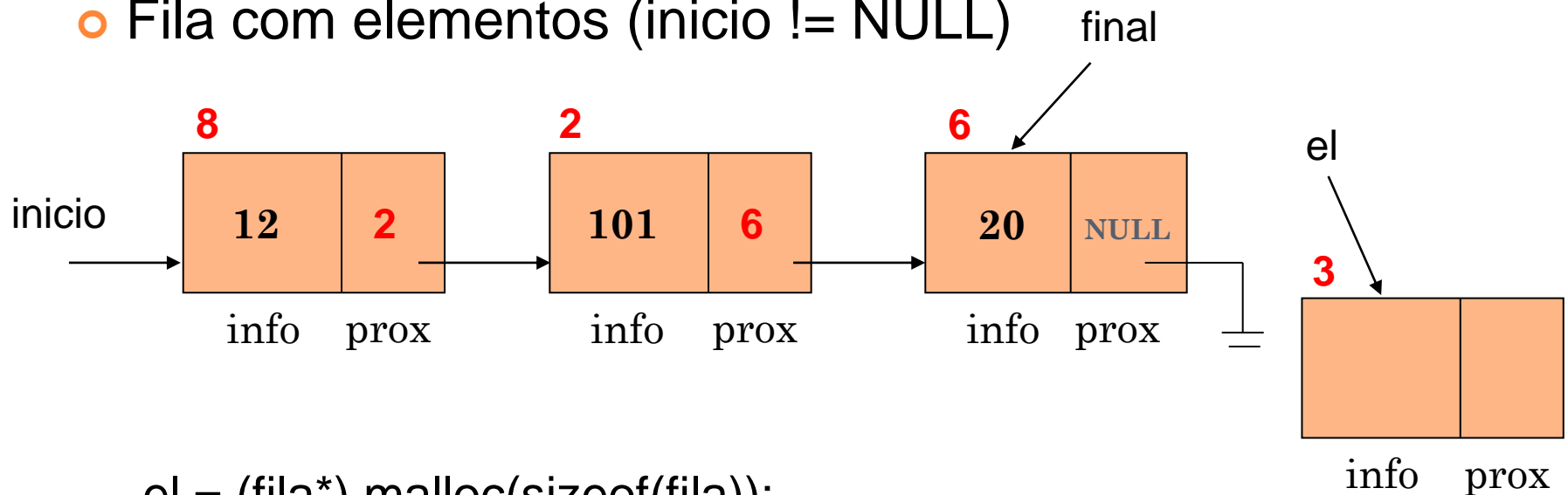
- Fila com elementos (inicio != NULL)



```
el = (fila*) malloc(sizeof(fila));
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila com elementos (inicio != NULL)



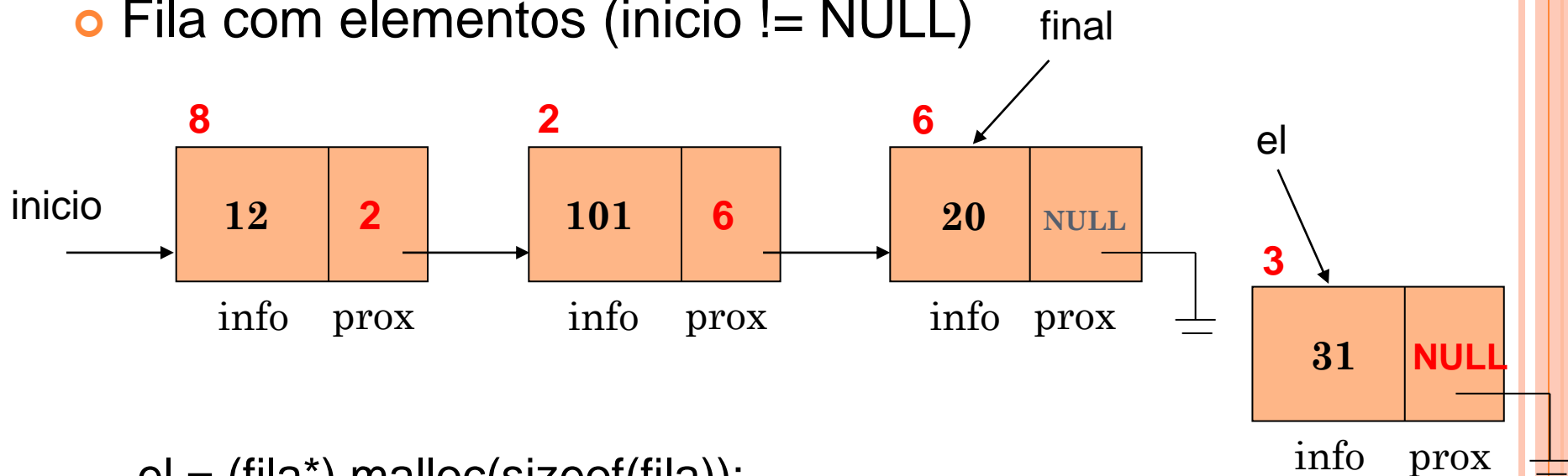
```
el = (fila*) malloc(sizeof(fila));
```

```
el->info = elem;
```

```
el->prox = NULL;
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila com elementos (inicio != NULL)



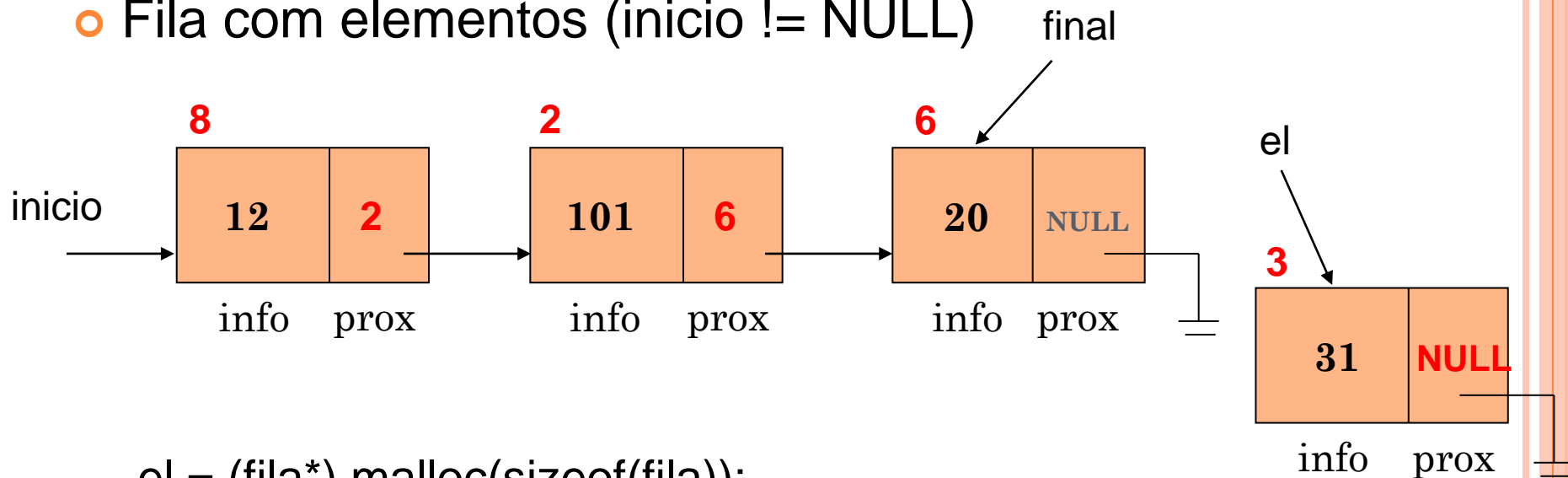
```
el = (fila*) malloc(sizeof(fila));
```

```
el->info = elem;
```

```
el->prox = NULL;
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila com elementos (inicio != NULL)



```
el = (fila*) malloc(sizeof(fila));
```

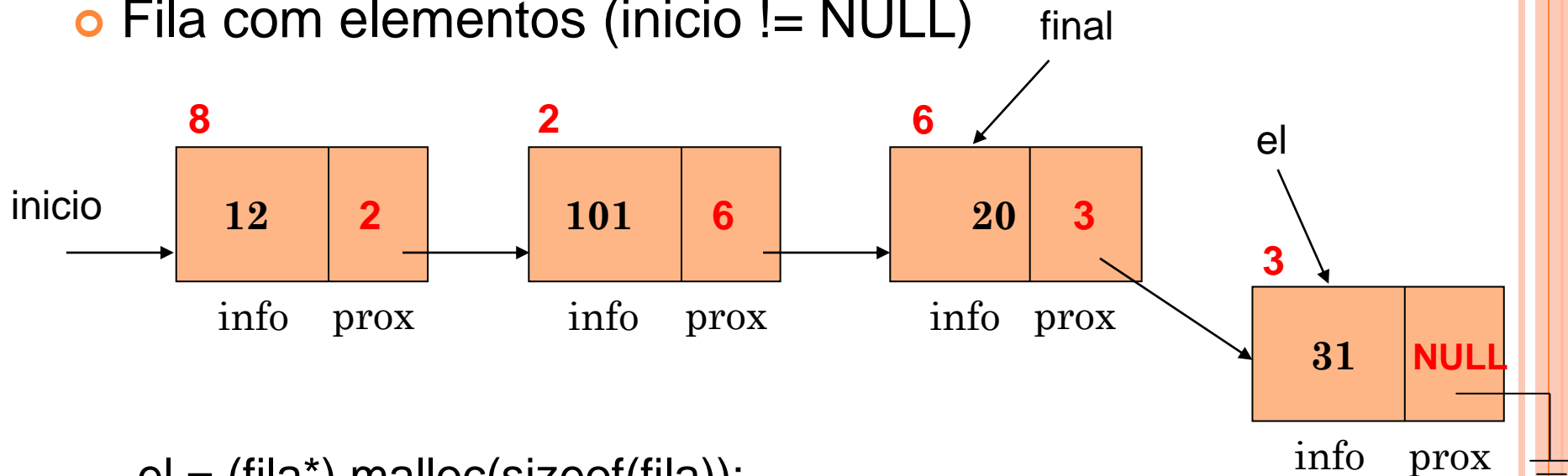
```
el->info = elem;
```

```
el->prox = NULL;
```

```
final->prox = el;
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila com elementos (inicio != NULL)



```
el = (fila*) malloc(sizeof(fila));
```

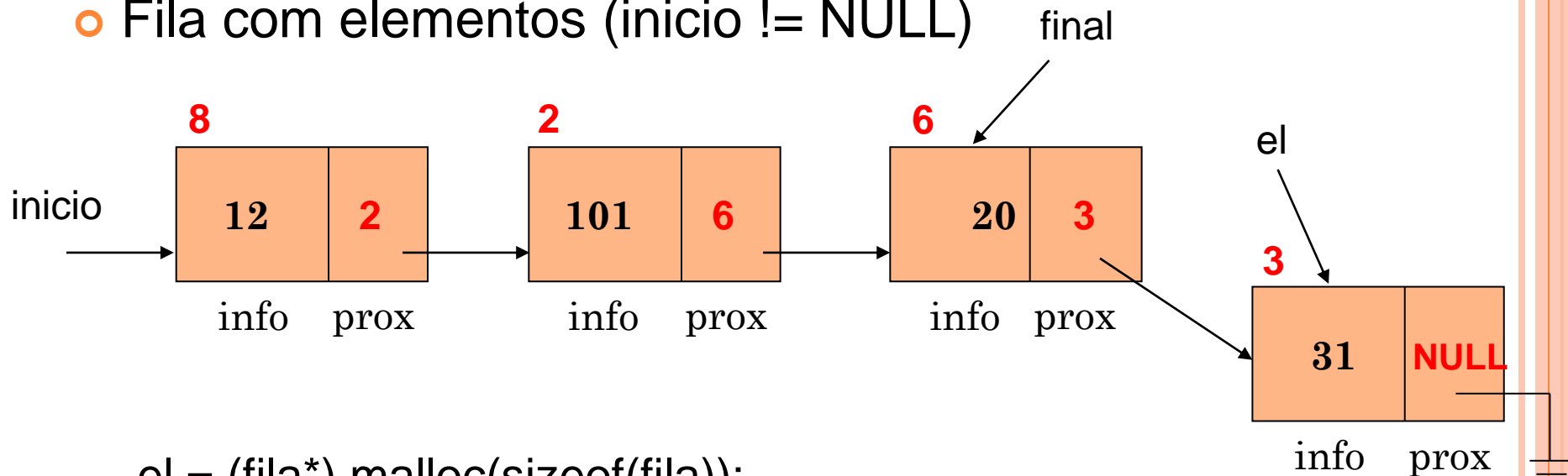
```
el->info = elem;
```

```
el->prox = NULL;
```

```
final->prox = el;
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila com elementos (inicio != NULL)



```
el = (fila*) malloc(sizeof(fila));
```

```
el->info = elem;
```

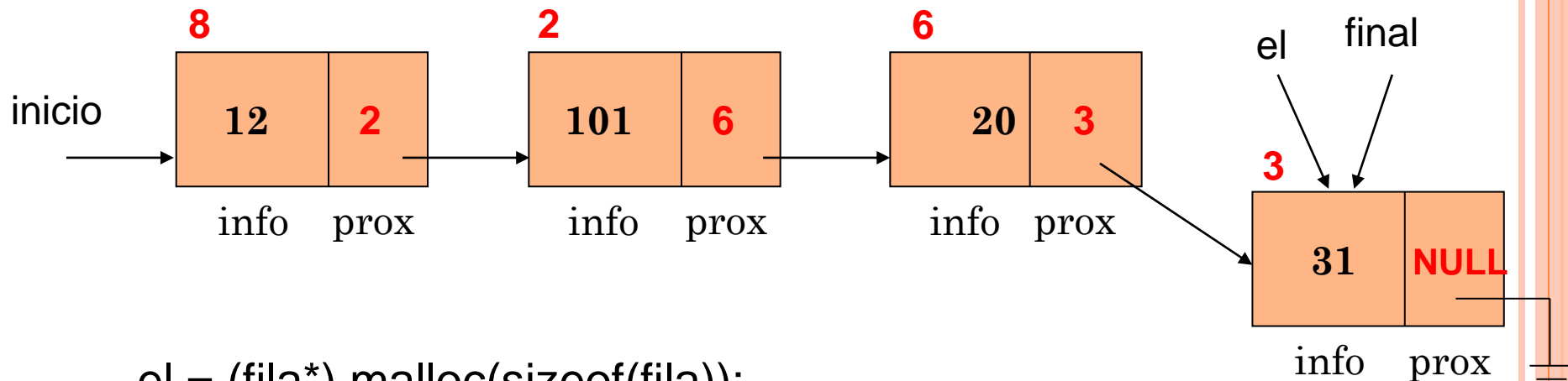
```
el->prox = NULL;
```

```
final->prox = el;
```

```
final = el;
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila com elementos (inicio != NULL)



```
el = (fila*) malloc(sizeof(fila));
```

```
el->info = elem;
```

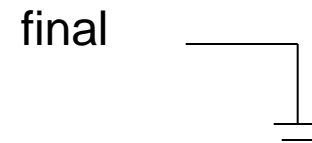
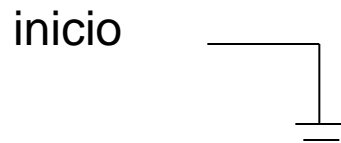
```
el->prox = NULL;
```

```
final->prox = el;
```

```
final = el;
```


INSERINDO UM NÓ NA FILA(ELEM = 31)

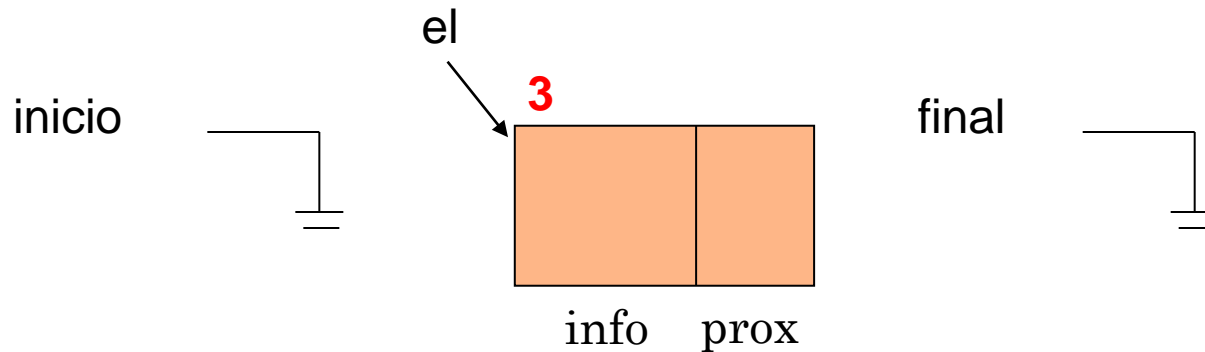
- Fila vazia (inicio == NULL)



```
el = (fila*) malloc(sizeof(fila));
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

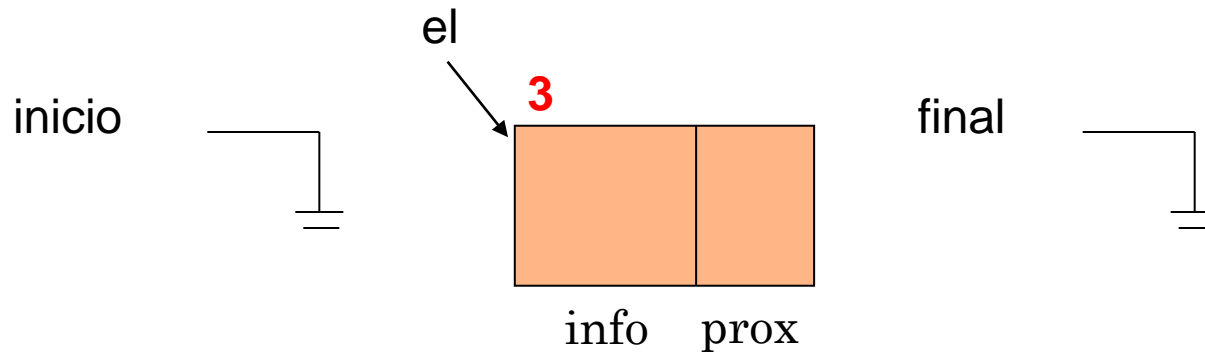
- Fila vazia (inicio == NULL)



```
el = (fila*) malloc(sizeof(fila));
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila vazia (inicio == NULL)



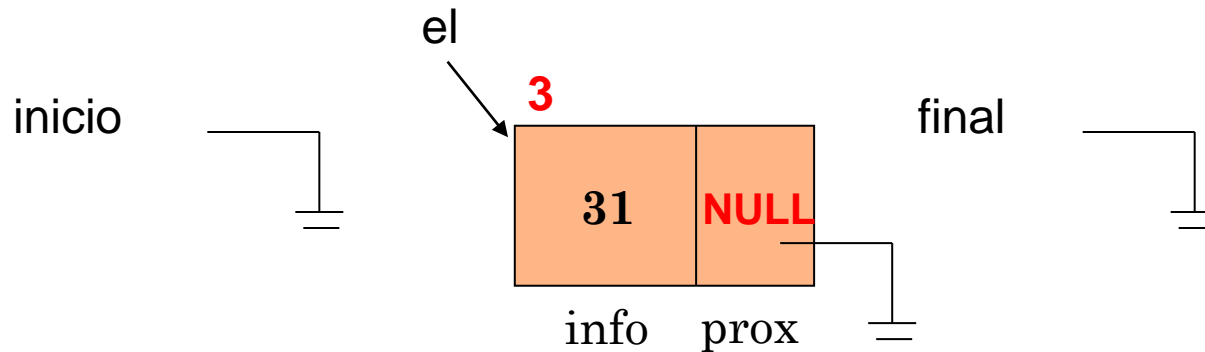
```
el = (fila*) malloc(sizeof(fila));
```

```
el->info = elem;
```

```
el->prox = NULL;
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila vazia (inicio == NULL)



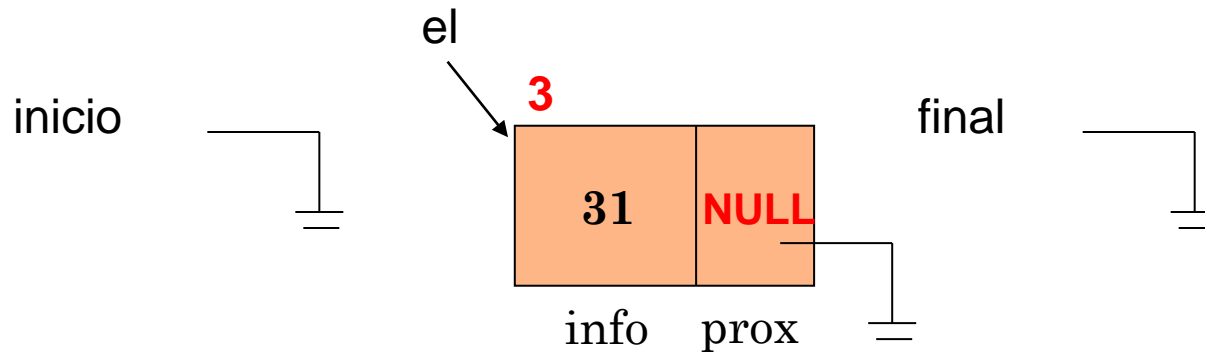
```
el = (fila*) malloc(sizeof(fila));
```

```
el->info = elem;
```

```
el->prox = NULL;
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila vazia (inicio == NULL)



```
el = (fila*) malloc(sizeof(fila));
```

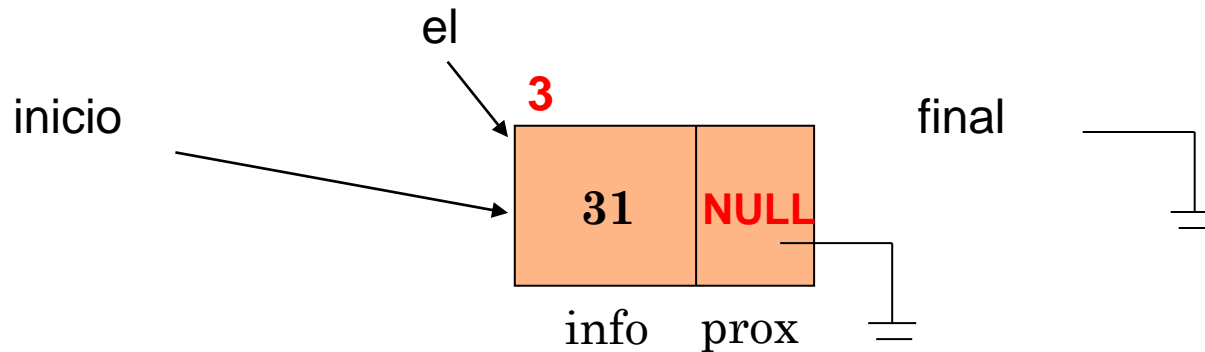
```
el->info = elem;
```

```
el->prox = NULL;
```

```
inicio = el;
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila vazia (inicio == NULL)



```
el = (fila*) malloc(sizeof(fila));
```

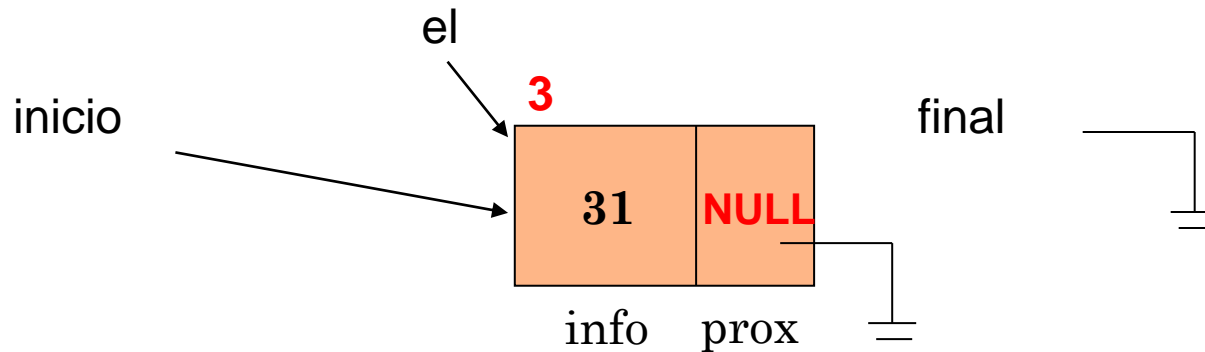
```
el->info = elem;
```

```
el->prox = NULL;
```

```
inicio = el;
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila vazia (inicio == NULL)



```
el = (fila*) malloc(sizeof(fila));
```

```
el->info = elem;
```

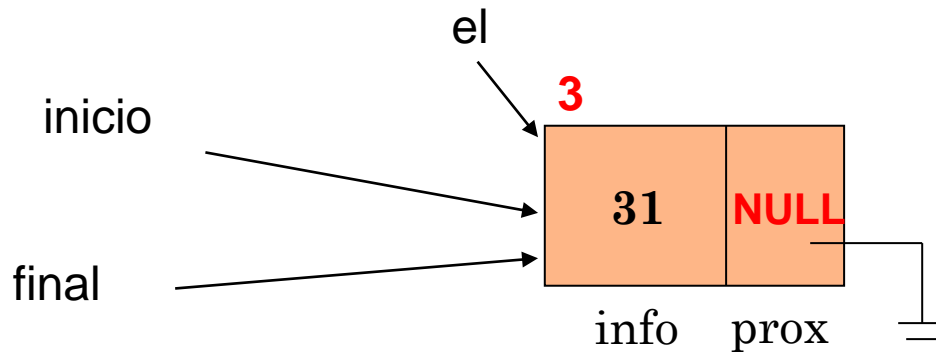
```
el->prox = NULL;
```

```
inicio = el;
```

```
final = el;
```

INSERINDO UM NÓ NA FILA(ELEM = 31)

- Fila vazia (inicio == NULL)



```
el = (fila*) malloc(sizeof(fila));
```

```
el->info = elem;
```

```
el->prox = NULL;
```

```
inicio = el;
```

```
final = el;
```


INSERINDO UM NÓ NA FILA

```
void insereElem(fila **inicio, fila **final, int elem){
```

```
    fila *el;
```

```
    el = (fila*) malloc(sizeof(fila));
```

```
    el->info = elem;
```

```
    el->prox = NULL;
```

```
    if ((*inicio) == NULL)
```

```
        (*inicio) = el;
```

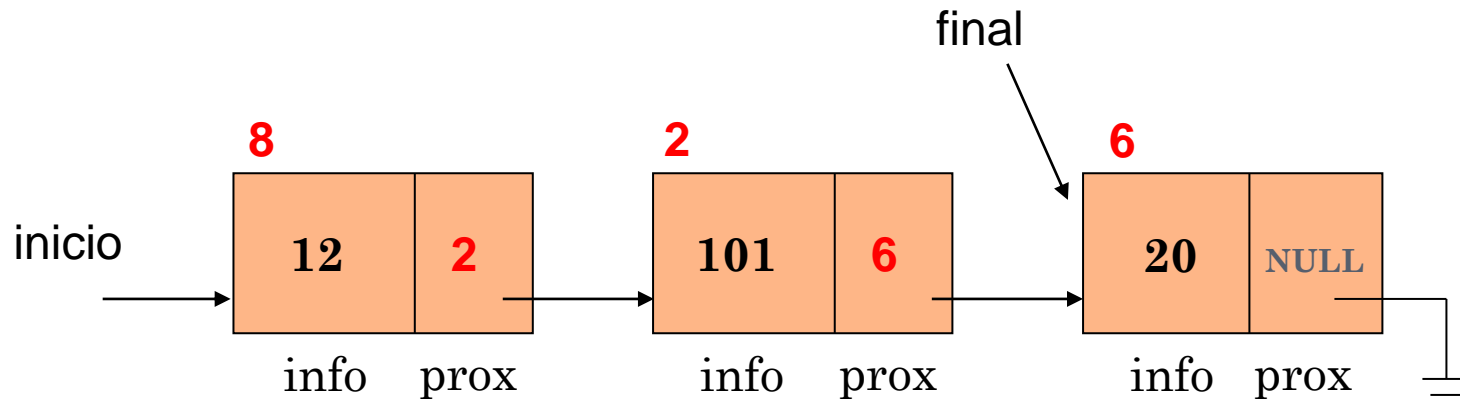
```
    else
```

```
        (*final)->prox = el;
```

```
    (*final) = el;
```

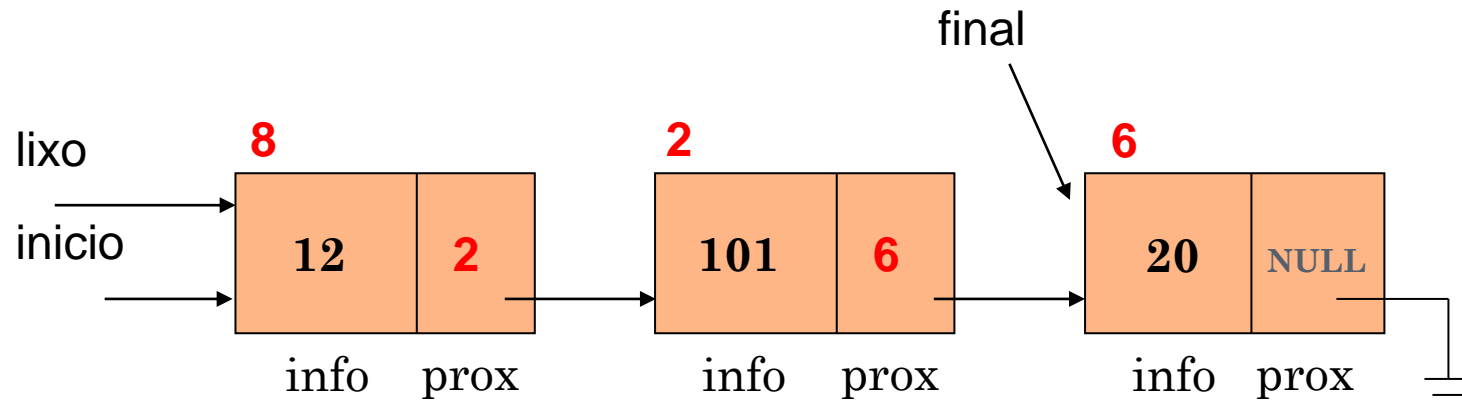
```
}
```

REMOVENDO UM NÓ DA FILA



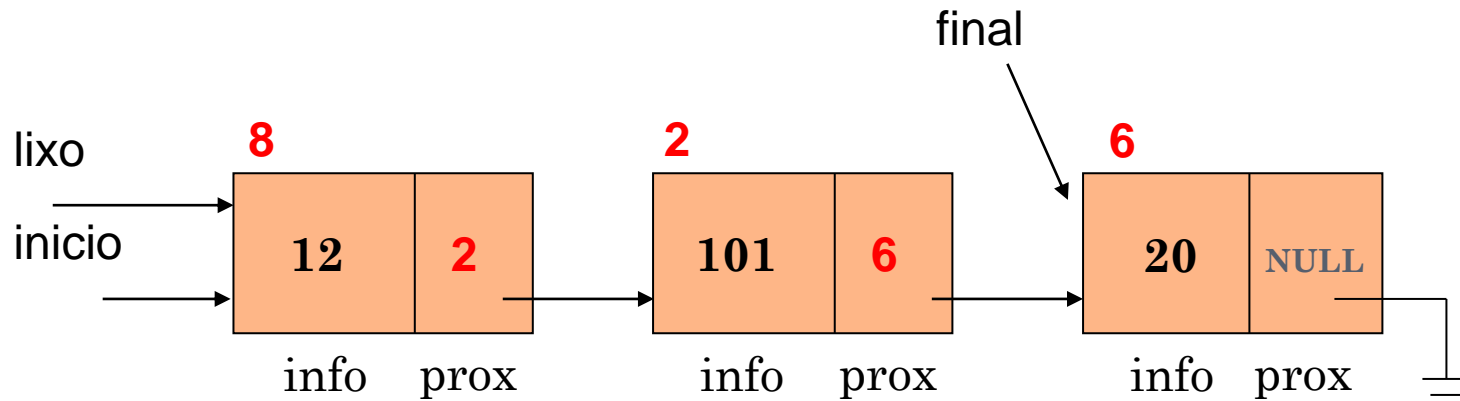
lixo = inicio;

REMOVENDO UM NÓ DA FILA



lixo = inicio;

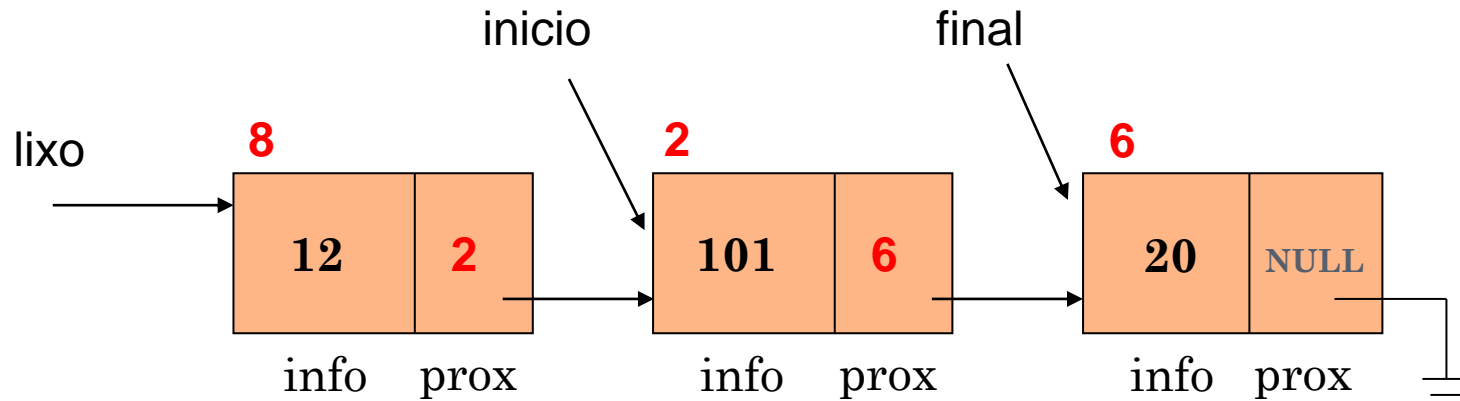
REMOVENDO UM NÓ DA FILA



`lixo = início;`

`início = início->prox;`

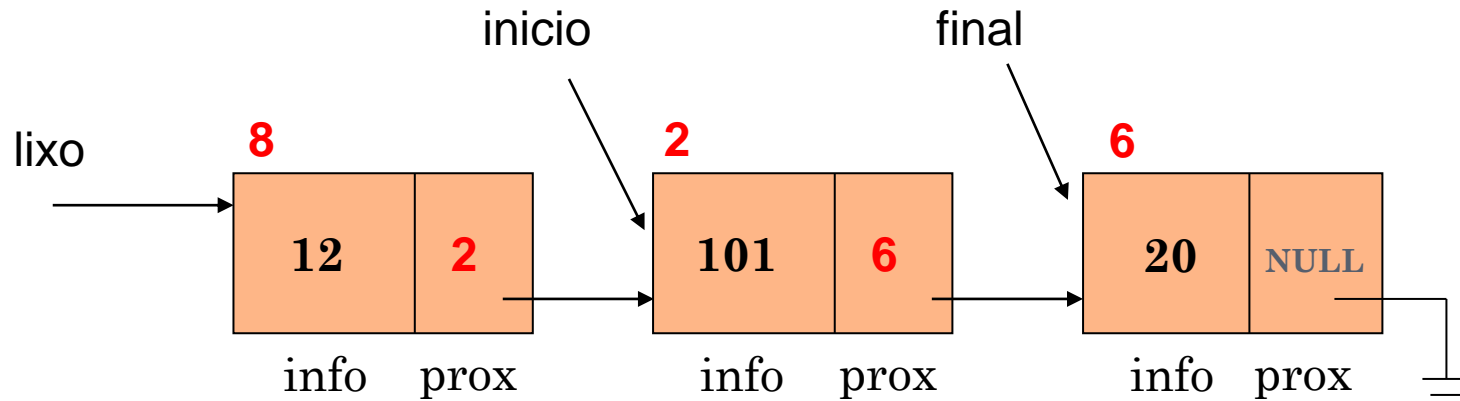
REMOVENDO UM NÓ DA FILA



`lixo = inicio;`

`inicio = inicio->prox;`

REMOVENDO UM NÓ DA FILA

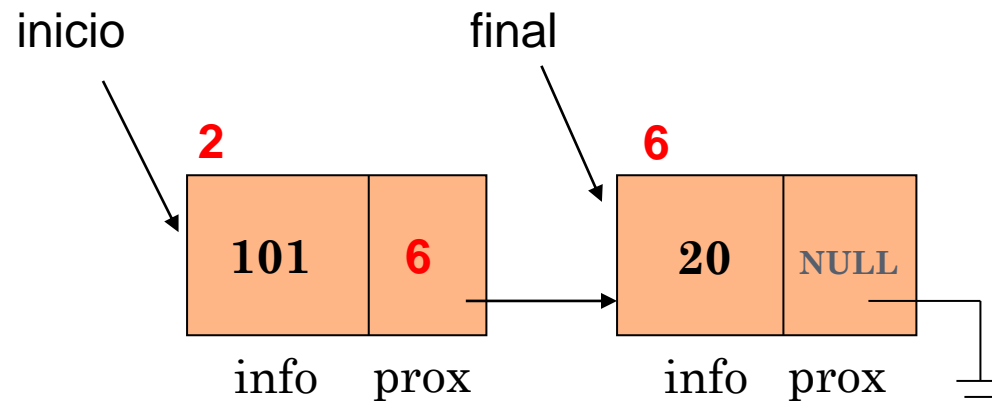


```
lixo = inicio;
```

```
inicio = inicio->prox;
```

```
free(lixo);
```

REMOVENDO UM NÓ DA FILA



```
lixo = inicio;
```

```
inicio = inicio->prox;
```

```
free(lixo);
```

REMOVENDO UM NÓ DA FILA

```
fila *removeElem(fila *inicio){  
  
    fila *lixo;  
  
    if (inicio != NULL) {  
        lixo = inicio;  
        inicio = inicio->prox;  
        free(lixo);  
    }  
    return inicio;  
}
```


EXERCÍCIO 1

- Faça um programa que:
 - Crie uma lista encadeada L com números inteiros
 - Os números devem ser inseridos na lista em ordem crescente até que o usuário digite um número negativo.
 - A lista não deve possuir números repetidos
 - Após preencher a lista crie um MENU na tela que permita o usuário (1) incluir novo elemento, (2) excluir elementos, (3) imprimir a lista e (4) sair.
 - Devem ser criadas as funções de buscaElemento, insereElemento, removeElemento e ImprimeLista

EXERCÍCIO 2

- Faça um programa que:
 - Crie 2 pilhas encadeada , P1 e P2, com números inteiros
 - Os números devem ser inseridos em cada pilha até que o usuário digite um número negativo
 - Após preencher as pilhas, crie uma terceira pilha (P3) que conterà os elementos de P1 e P2 alternadamente. Caso, uma das pilhas termine primeiro que a outra, P3 continuará a ser preenchida com os elementos da pilha restante.
 - Ao final do programa, imprima a P3
 - Devem ser criadas as funções push e pop

EXERCÍCIO 3

- Faça um programa que:
 - Crie uma fila encadeada com números inteiros
 - Os números devem ser inseridos na fila até que o usuário digite um número negativo
 - Ao final do programa, imprima a fila
 - Devem ser criadas as funções insere e remove