

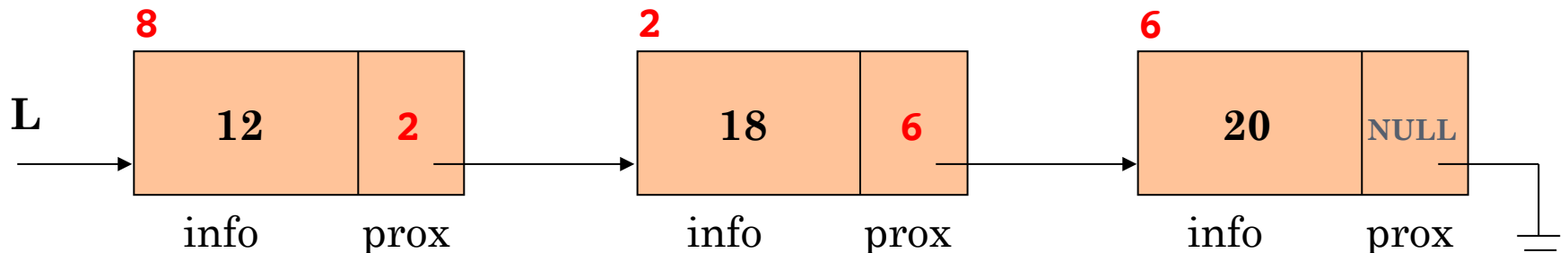


PROGRAMAÇÃO ESTRUTURADA

Listas com Alocação Encadeada Dinâmica
Usando Nó Cabeça

LISTAS COM NÓ CABEÇA

- Como visto anteriormente, é necessária uma variável do tipo ponteiro que guarde o endereço inicial da lista
- Logo, as funções de inclusão e remoção de nós devem apresentar testes para verificar se a ação desejada ocorrerá no início da lista
 - Isto é necessário para saber se o endereço inicial da lista será modificado ou não



LISTAS SEM NÓ CABEÇA

```
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;
}
```

```
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;
}
```

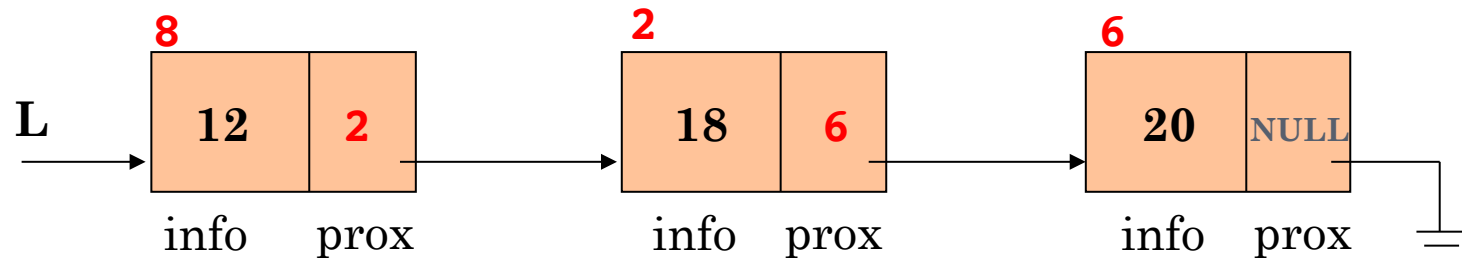
LISTAS COM NÓ CABEÇA

- Para simplificar os algoritmos, eliminando os testes, podemos adotar uma pequena variação na implementação da lista encadeada: a criação do **Nó cabeça**
- O nó cabeça é um nó especial que nunca é removido da lista
- O ponteiro que guarda o início da lista, armazena o endereço do nó cabeça

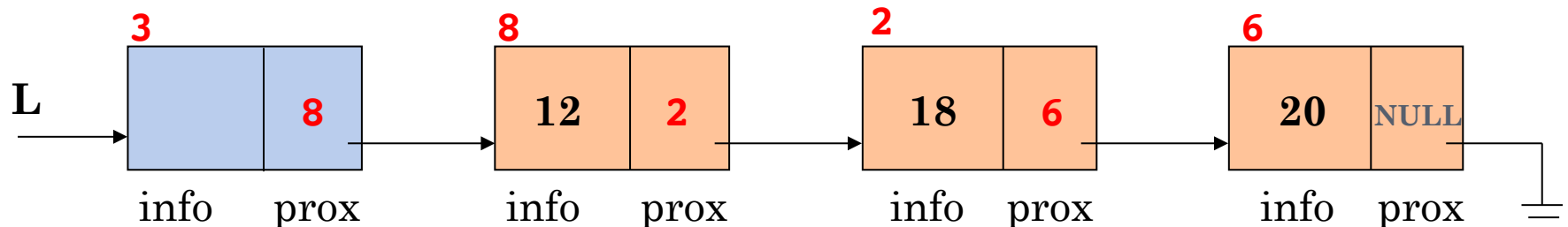
LISTAS COM NÓ CABEÇA

○ Representação Gráfica

• Lista sem Nó Cabeça



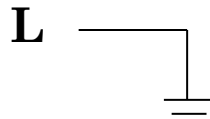
• Lista com Nó Cabeça



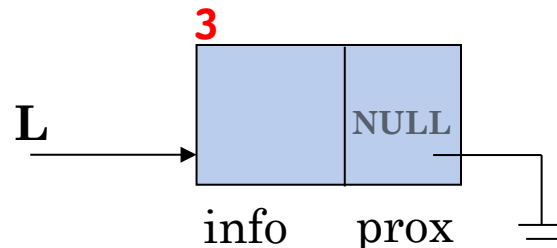
LISTAS COM NÓ CABEÇA

○ Representação Gráfica

- Lista Vazia sem Nó Cabeça



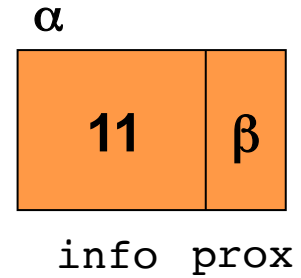
- Lista Vazia com Nó Cabeça



LISTAS COM NÓ CABEÇA

- Em uma lista encadeada com nó cabeça, não existe o caso de incluir um nó em uma lista vazia ou na primeira posição
 - Só temos inclusão de nós no meio ou fim da lista
- Da mesma forma, não existe o caso de se remover o primeiro nó, já que este é sempre o nó cabeça
 - Só temos remoção de nós no meio ou fim da lista

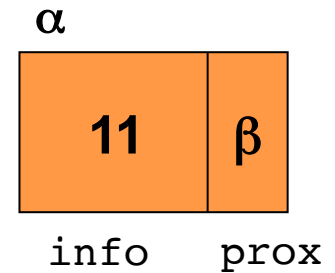
LISTAS COM NÓ CABEÇA



- A estrutura de um nó da lista não muda
- A única modificação é adicionar um novo nó na lista

```
struct NO {  
    int info;  
    struct NO *prox;  
}  
typedef struct NO lista;  
  
...  
lista *L;  
L = (lista*) malloc(sizeof(lista));  
L->prox = NULL;
```


FUNÇÃO BUSCA ELEMENTO EM L

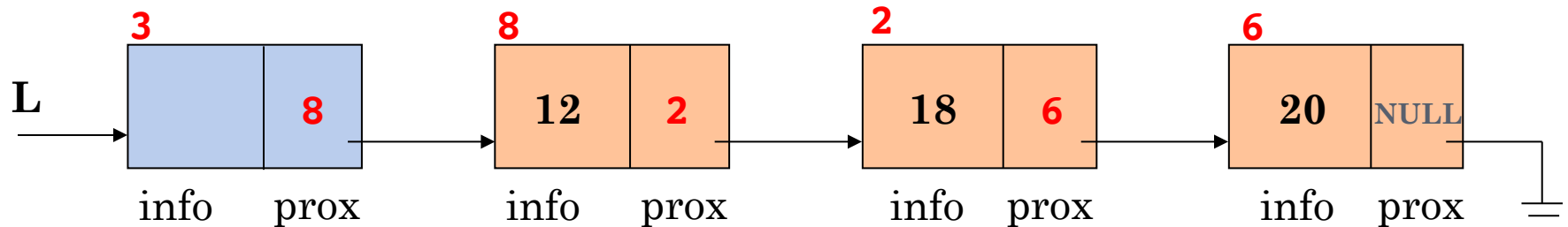


- Considere uma Lista L com nó cabeça, simplesmente encadeada, que guarde números inteiros ordenados
- Cada Nó da lista possui a parte `info` e a parte `prox`
- A função `buscaElemento` terá o seguinte protótipo:

```
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

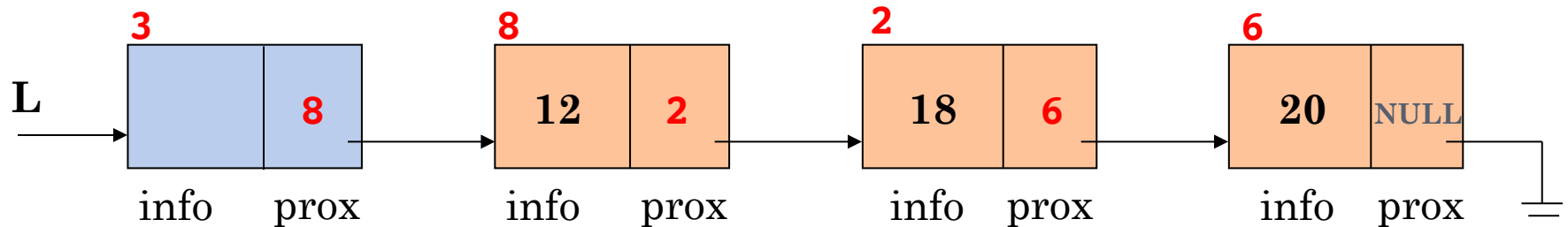
FUNÇÃO BUSCA ELEMENTO EM L



```
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->next;
    }
    (*pre) = preL;
    if ((aux != NULL) && (elem == aux->info))
        return 1;
    return 0;
}
```

O que mudaria nesta função?

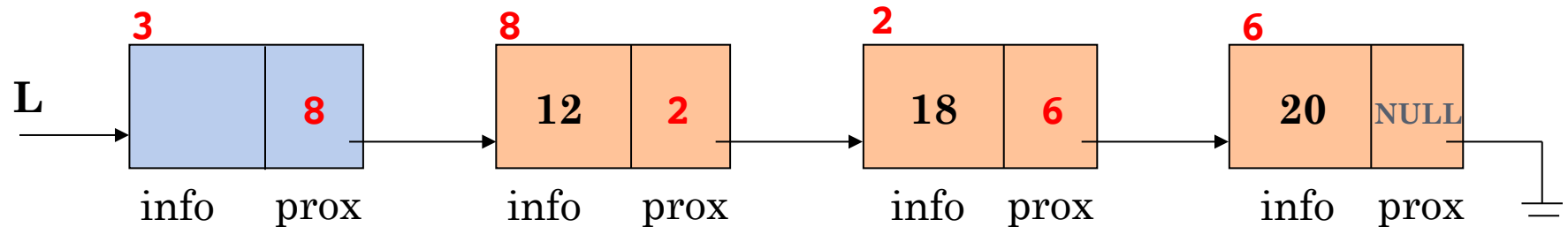
FUNÇÃO BUSCA ELEMENTO EM L



```
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->next;
    }
    (*pre) = preL;
    if ((aux != NULL) && (elem == aux->info))
        return 1;
    return 0;
}
```

O que mudaria nesta função?

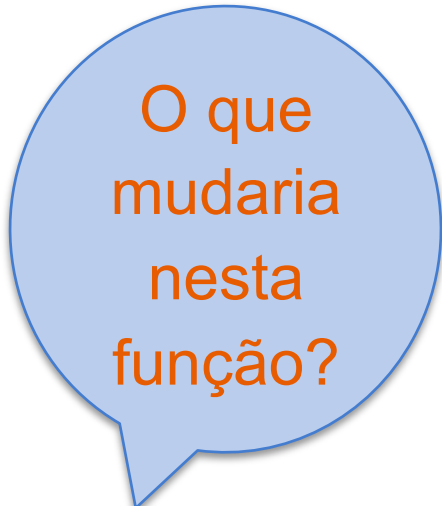
FUNÇÃO BUSCA ELEMENTO EM L, COM NÓ CABEÇA



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

FUNÇÃO INSERE ELEMENTO 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;  
}
```



O que
mudaria
nesta
função?

FUNÇÃO INSERE ELEMENTO 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;  
}
```

O que mudaria nesta função?

Não é necessário testar se a lista é vazia ou se o nó a ser inserido é o primeiro da lista, pois agora sempre existe o nó cabeça

FUNÇÃO INSERE ELEMENTO EM L, COM NÓ CABEÇA

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

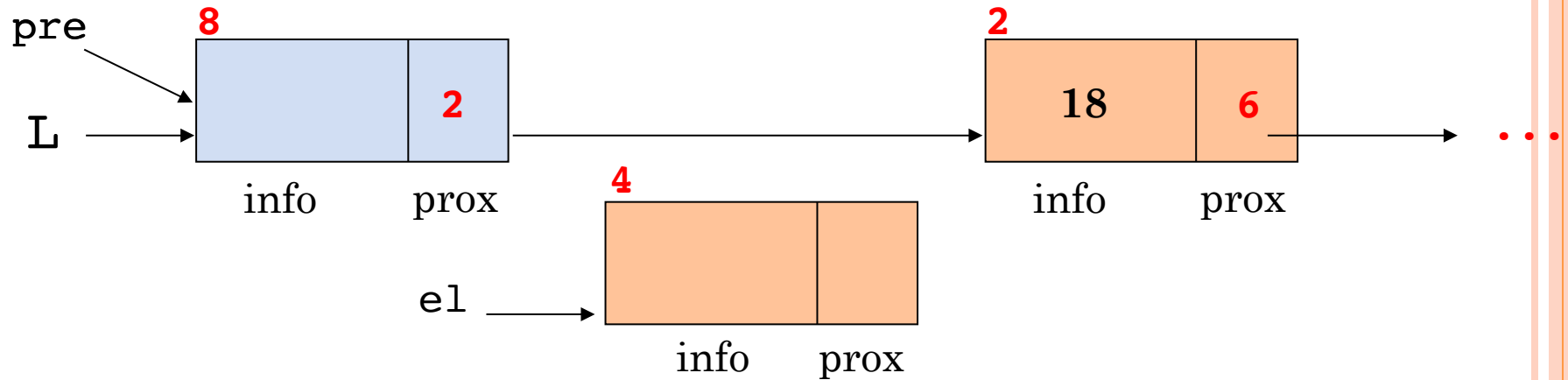
Não é necessário testar se a lista é vazia ou se o nó a ser inserido é o primeiro da lista, pois agora sempre existe o nó cabeça

INSERINDO UM NÓ EM L, COM NÓ CABEÇA (ELEM = 14)



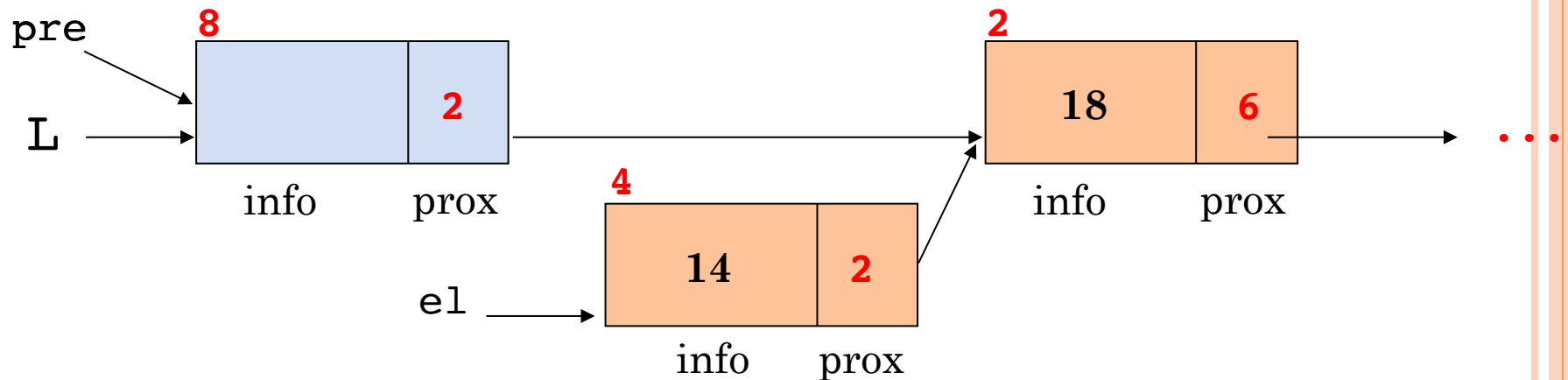
```
el = (lista *) malloc(sizeof(lista));
```


INSERINDO UM NÓ EM L, COM NÓ CABEÇA (ELEM = 14)



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

INSERINDO UM NÓ EM L, COM NÓ CABEÇA (ELEM = 14)

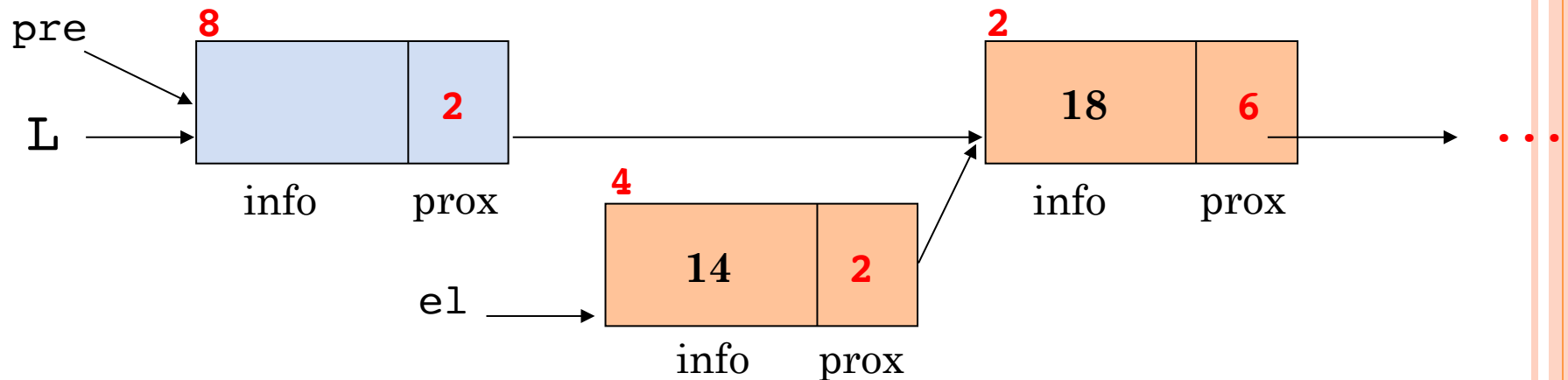


```
el = (lista *) malloc(sizeof(lista));
```

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

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

INSERINDO UM NÓ EM L, COM NÓ CABEÇA (ELEM = 14)



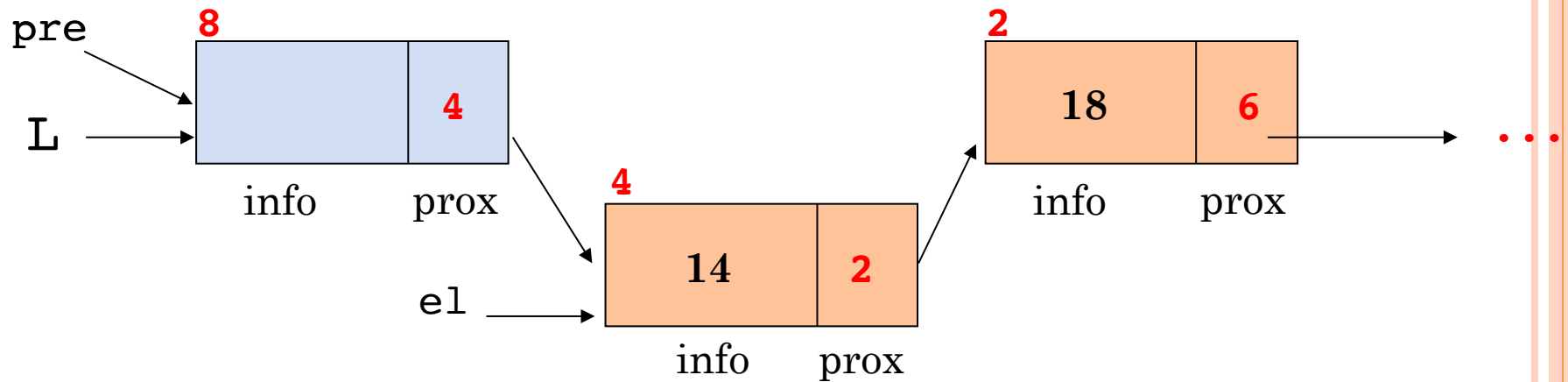
```
el = (lista *) malloc(sizeof(lista));
```

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

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

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

INSERINDO UM NÓ EM L, COM NÓ CABEÇA (ELEM = 14)



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

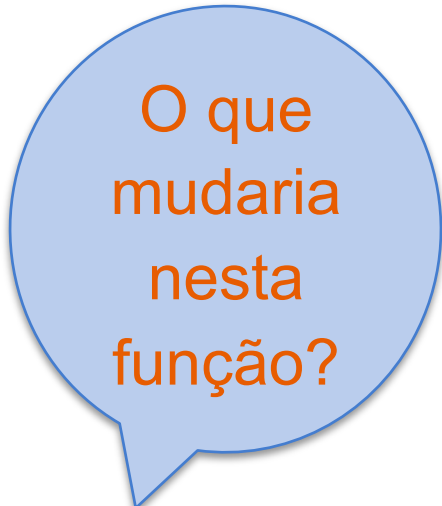
```
el->info = elem;
```

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

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

FUNÇÃO REMOVE ELEMENTO DE L

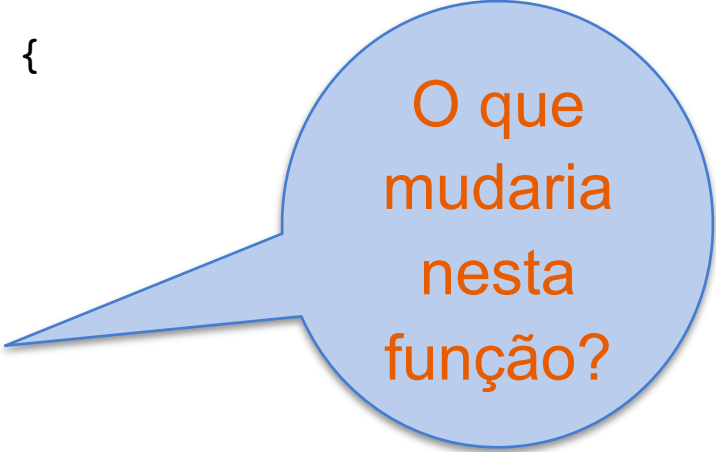
```
lista *removeElem(lista *L, int elem) {  
    lista *pre, *lixo;  
  
    if (buscaElem(L, elem, &pre)) {  
        if (L->info == elem) {  
            lixo = L;  
            L = L->prox;  
        } else {  
            lixo = pre->prox;  
            pre->prox = lixo->prox;  
        }  
   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



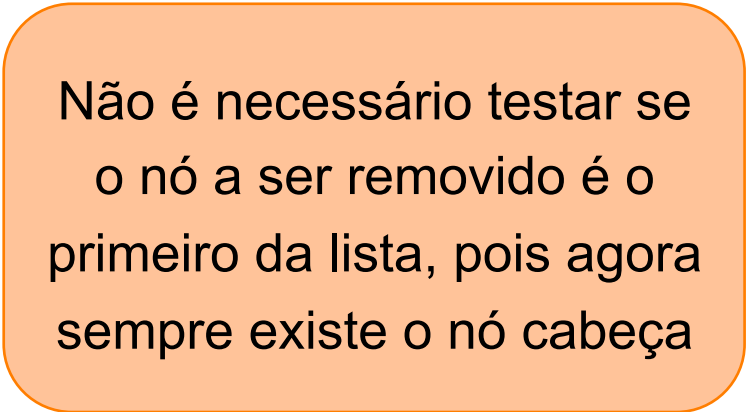
O que
mudaria
nesta
função?

FUNÇÃO REMOVE ELEMENTO 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;  
}
```



O que mudaria nesta função?



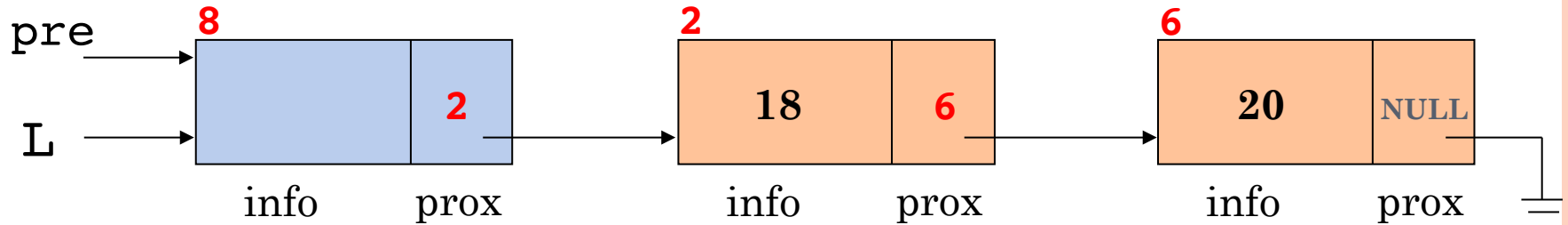
Não é necessário testar se o nó a ser removido é o primeiro da lista, pois agora sempre existe o nó cabeça

FUNÇÃO REMOVE ELEMENTO DE L, COM NÓ CABEÇA

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

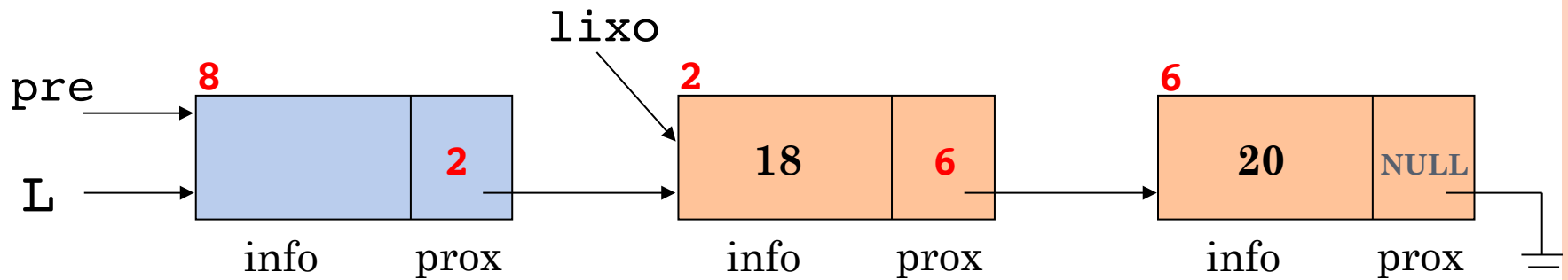
Não é necessário testar se o nó a ser removido é o primeiro da lista, pois agora sempre existe o nó cabeça

REMOVENDO UM NÓ DE L, COM NÓ CABEÇA (ELEM = 18)



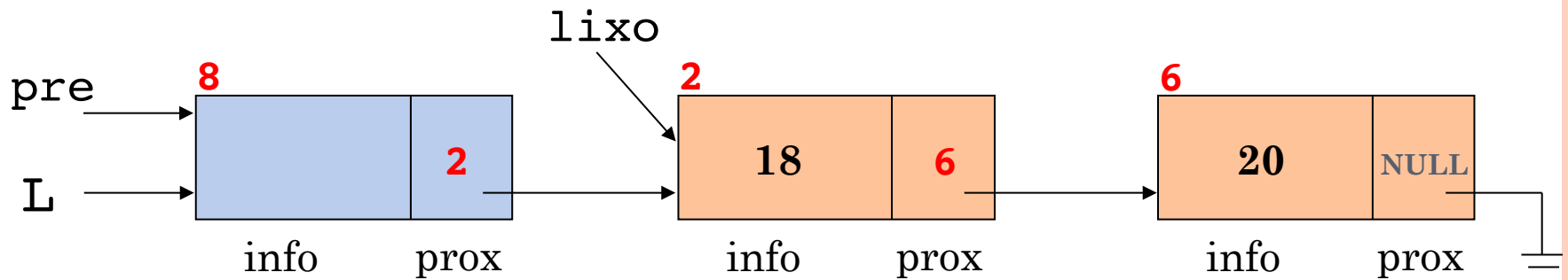
```
lixo = pre->prox;
```


REMOVENDO UM NÓ DE L, COM NÓ CABEÇA (ELEM = 18)



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

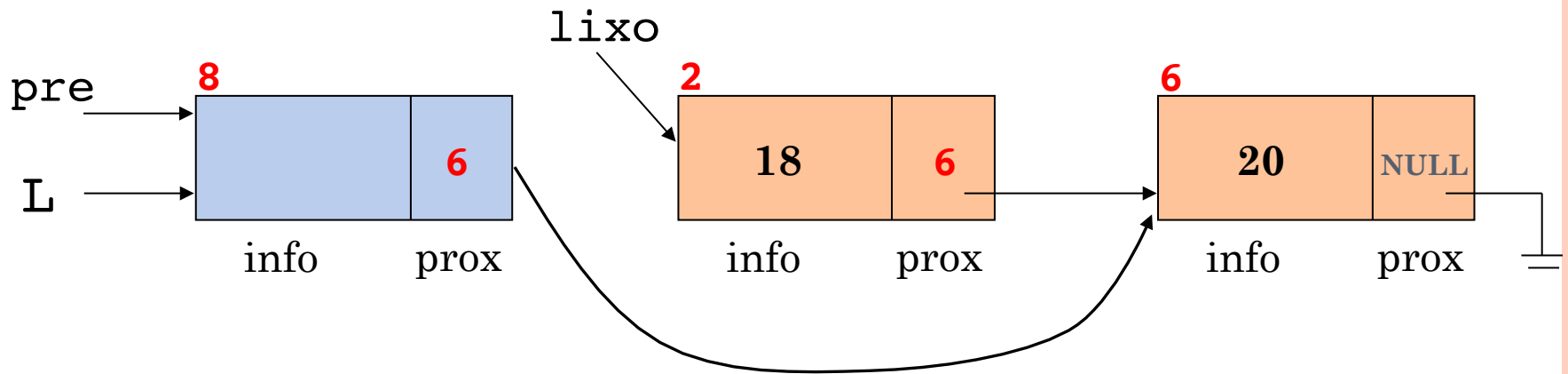
REMOVENDO UM NÓ DE L, COM NÓ CABEÇA (ELEM = 18)



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

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

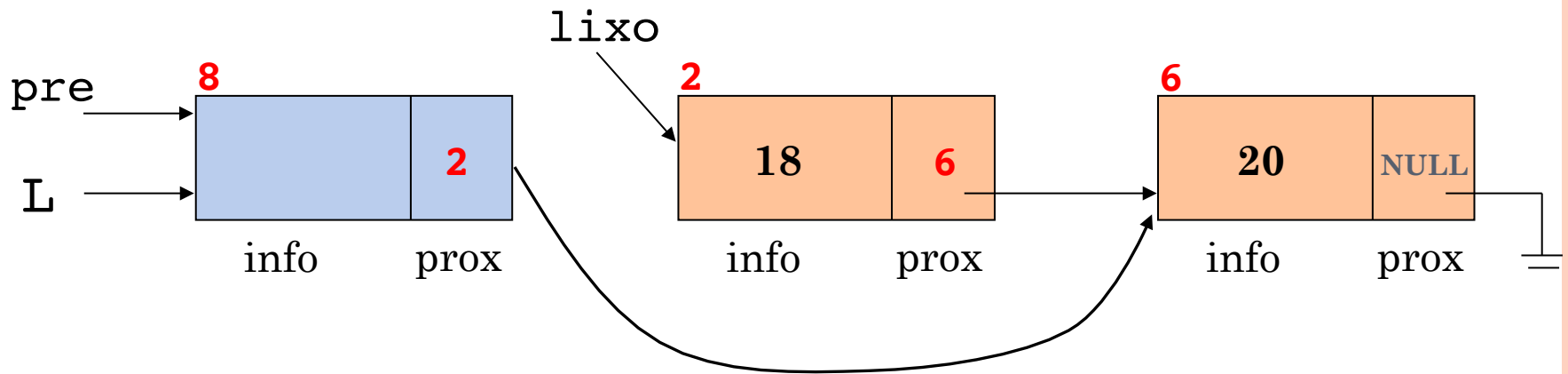
REMOVENDO UM NÓ DE L, COM NÓ CABEÇA (ELEM = 18)



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

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

REMOVENDO UM NÓ DE L, COM NÓ CABEÇA (ELEM = 18)

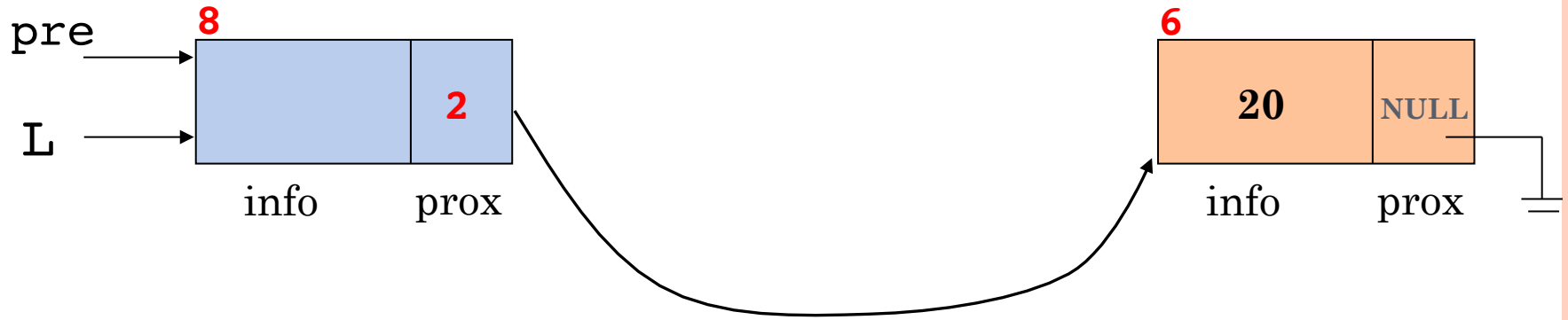


```
lixo = pre->prox;
```

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

```
free(lixo);
```

REMOVENDO UM NÓ DE L, COM NÓ CABEÇA (ELEM = 18)



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

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

```
free(lixo);
```

EXERCÍCIO 1

- Faça um programa que:
 - Crie uma lista encadeada L com nomes de frutas
 - Os nomes devem ser inseridos na lista em ordem crescente até que o usuário digite a palavra 'fim'
 - A lista não deve possuir nomes repetidos
 - Ao final do programa, imprima a lista
 - Faça funções para inserir um elemento e imprimir a lista

EXERCÍCIO 1 - CONTINUAÇÃO

- Continue no programa anterior:
 - Após preencher a lista crie um MENU com as opções:
 - 1 – Insere elemento
 - 2 – Remove elemento
 - 3 – Mostra lista
 - 4 – Informa a quantidade de nós
 - 5 – Fim do programa
 - Além das funções já criadas, crie funções para remover um elemento, apresentar o menu na tela e uma função recursiva para contar os nós da lista