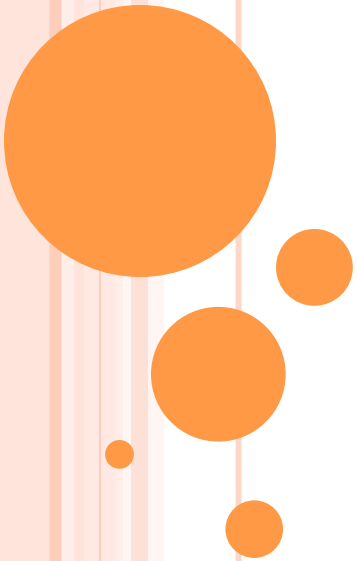
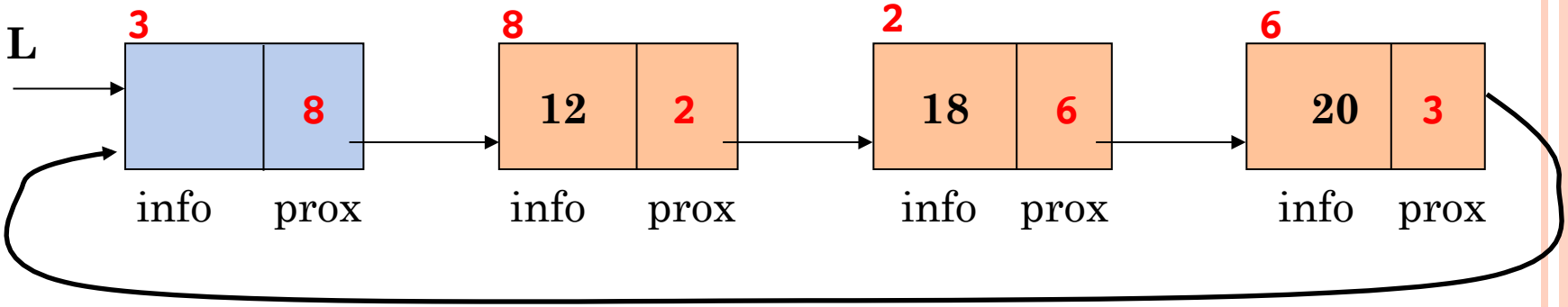


Programação Estruturada

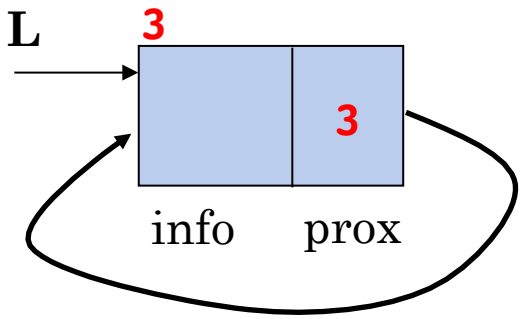
Listas com Alocação Encadeada Dinâmica
Duplamente Encadeadas
Circulares

Listas Circulares

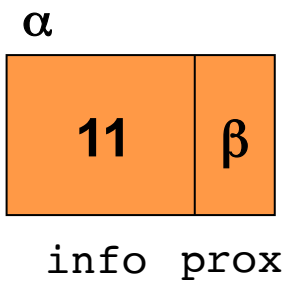
- Em uma lista circular encadeada, o último nó da lista aponta para o primeiro
  - Normalmente, usamos nó cabeça



  - Lista Vazia



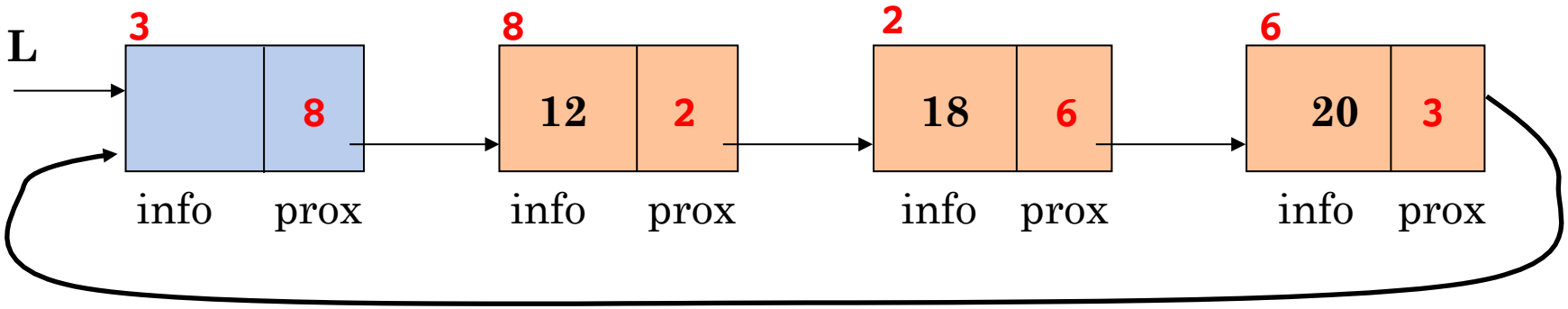
Listas Circulares com Nó Cabeça



- A estrutura de um nó da lista não muda
- A inicialização muda, pois não temos NULL

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

...
lista *L;
L = (lista*) malloc(sizeof(lista));
L->prox = L;
```

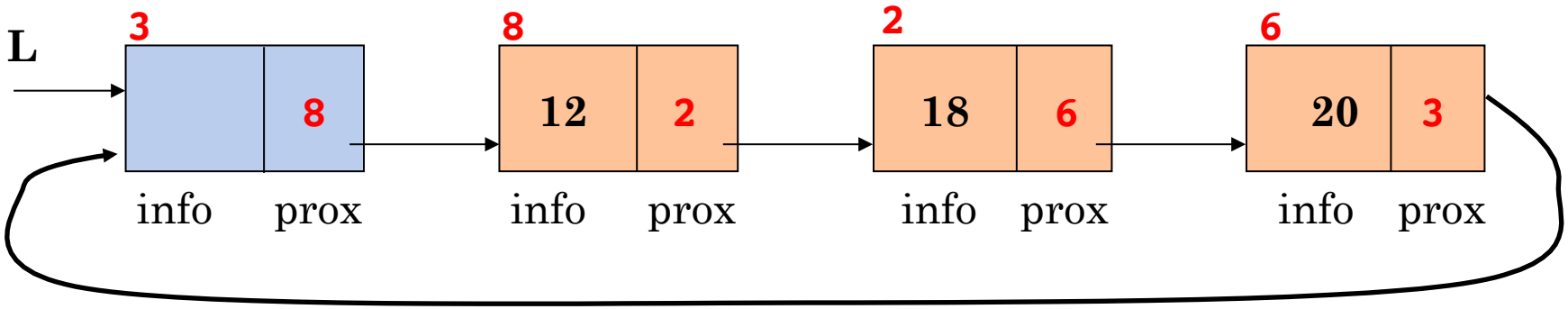
Função Busca Elemento em L, Circular e Com Nó Cabeça




```
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;
}
```

O que mudaria nesta função, sendo a lista circular?

Função Busca Elemento em L, Circular e Com Nó Cabeça




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

Função Insere Elemento em L, Circular e Com Nó Cabeça

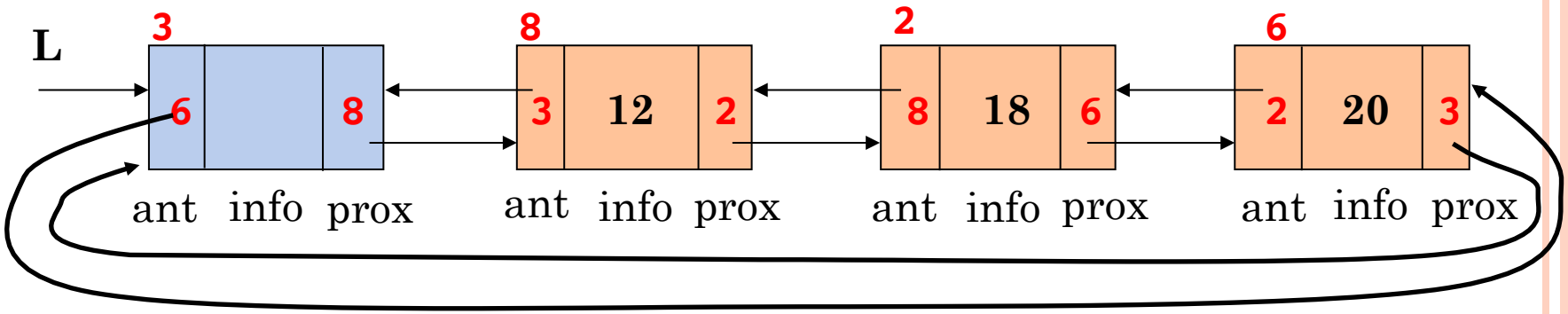
```
lista *insereElem(lista *L, int elem) {

   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;
}
```

Função Remove Elemento em L, Circular e Com Nó Cabeça

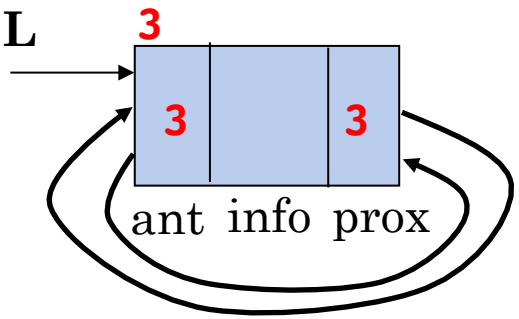
```
lista *removeElem(lista *L, int elem) {

   lista *pre, *lixo;

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

Listas Duplamente Encadeadas

- Em uma lista duplamente encadeada, cada nó guarda o endereço do próximo e do seu antecessor
  - Podem ser circulares e possuírem nós cabeça



  - Lista Vazia

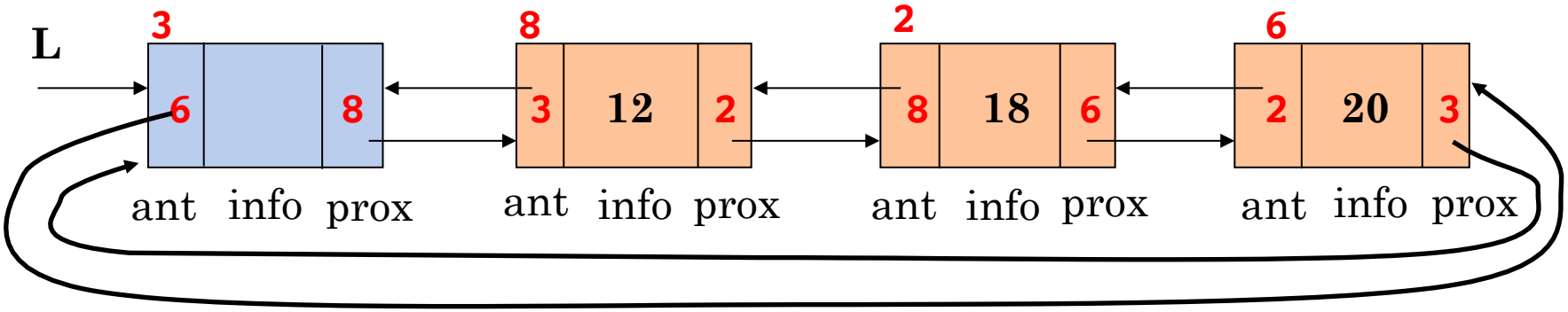


Listas Duplamente encadeadas Circulares com Nó Cabeça

- A estrutura de um nó da lista muda
- A inicialização também muda

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

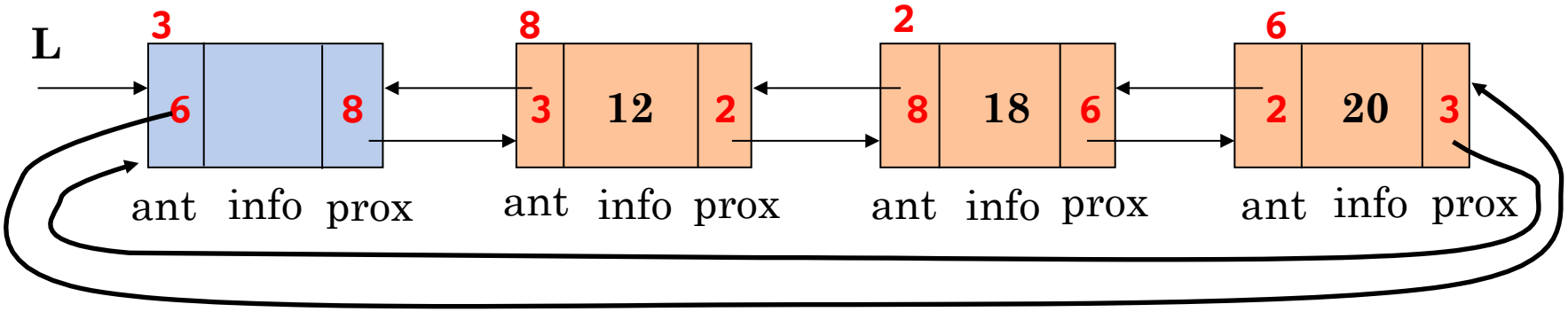
Função Busca Elemento em L, Duplamente encadeadas, Circulares com Nó Cabeça




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

O fato de ser duplamente encadeada mudaria a busca ?

Função Busca Elemento em L, Duplamente encadeadas, Circulares com Nó Cabeça

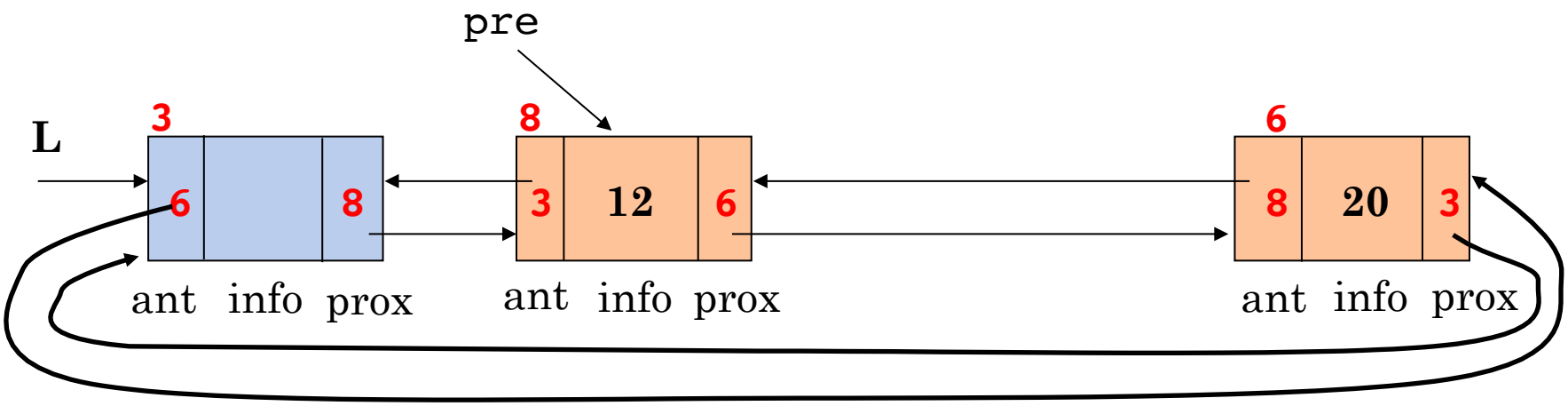



```
int buscaElem(lista *L, int elem, lista **pre){
      lista *aux, *preL;
      aux = L->prox;
      preL = L;
      while ((aux != L) && (elem > aux->info)) {
               preL = aux;
               aux = aux->next;
      }
      (*pre) = preL;
      if ((aux != L) && (elem == aux->info))
    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

O fato de ser duplamente encadeada mudaria a busca ?

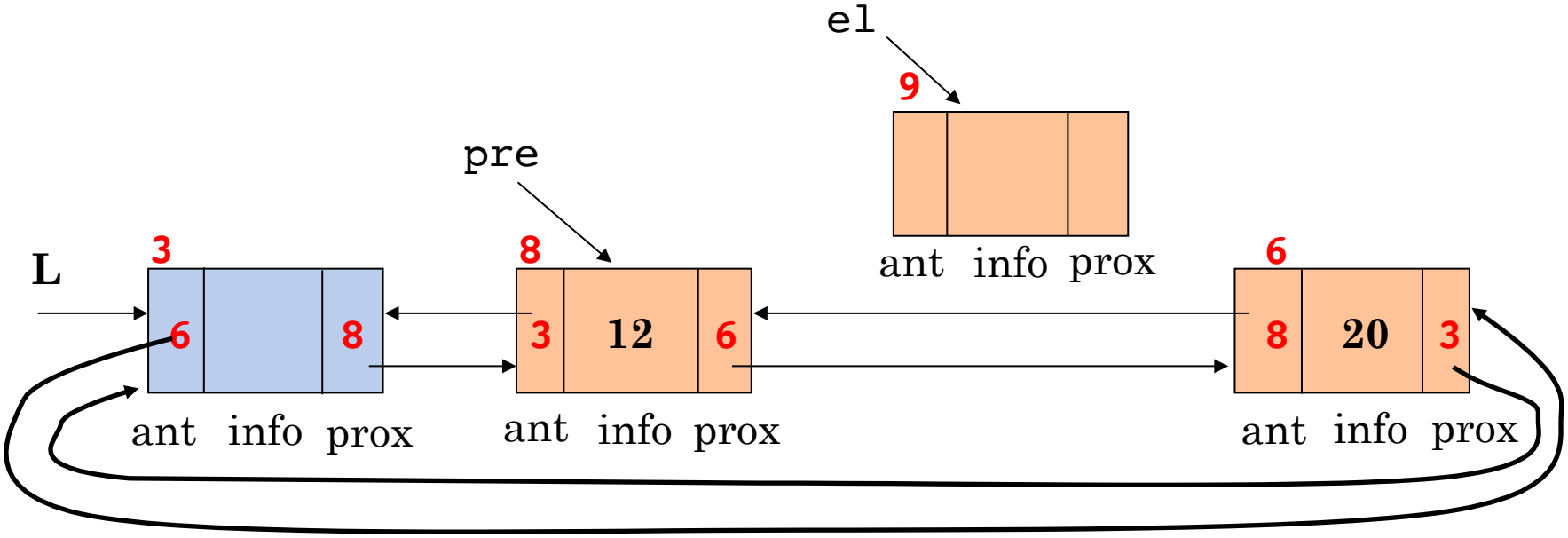
NÃO!

Inserindo um nó em L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 14)



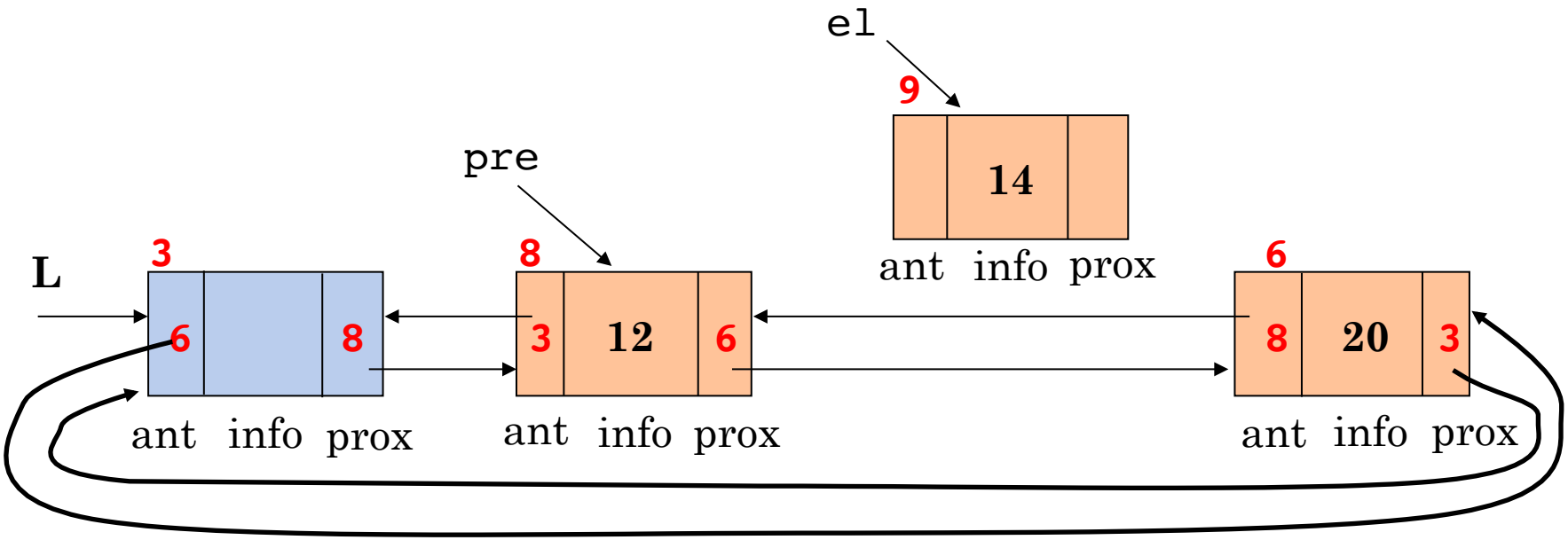

```
el = (lista *) malloc(sizeof(lista));
```

Inserindo um nó em L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 14)



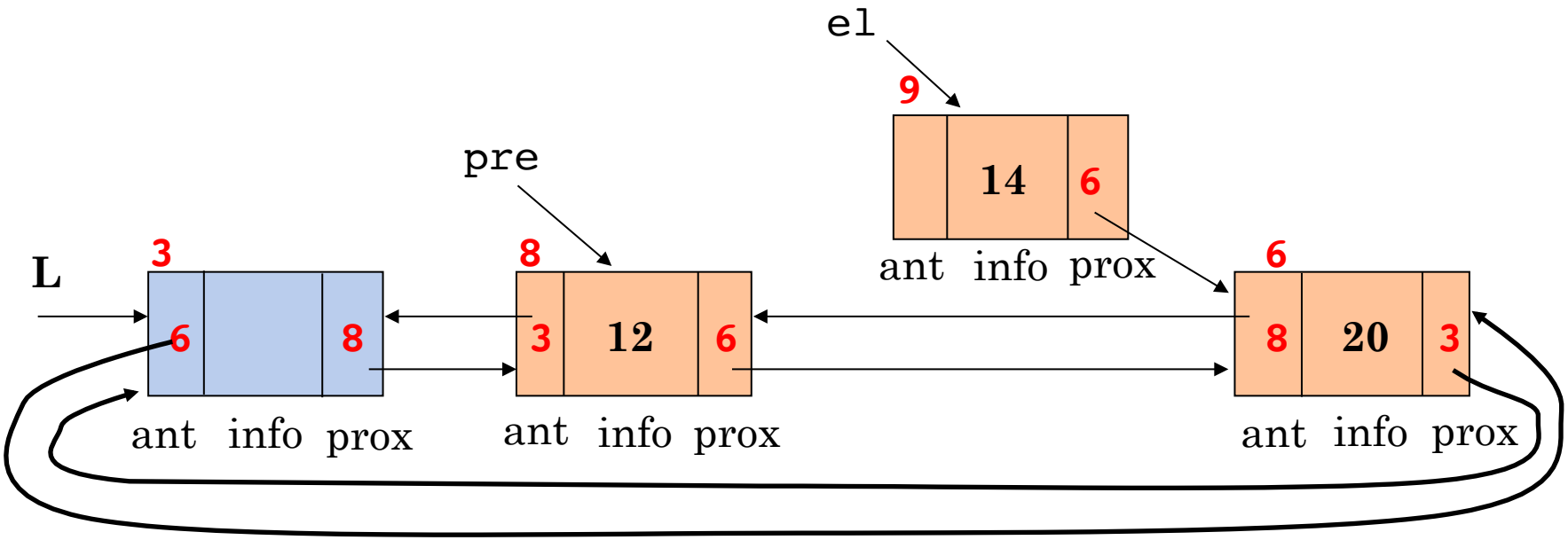

```
el = (lista *) malloc(sizeof(lista));
```

Inserindo um nó em L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 14)



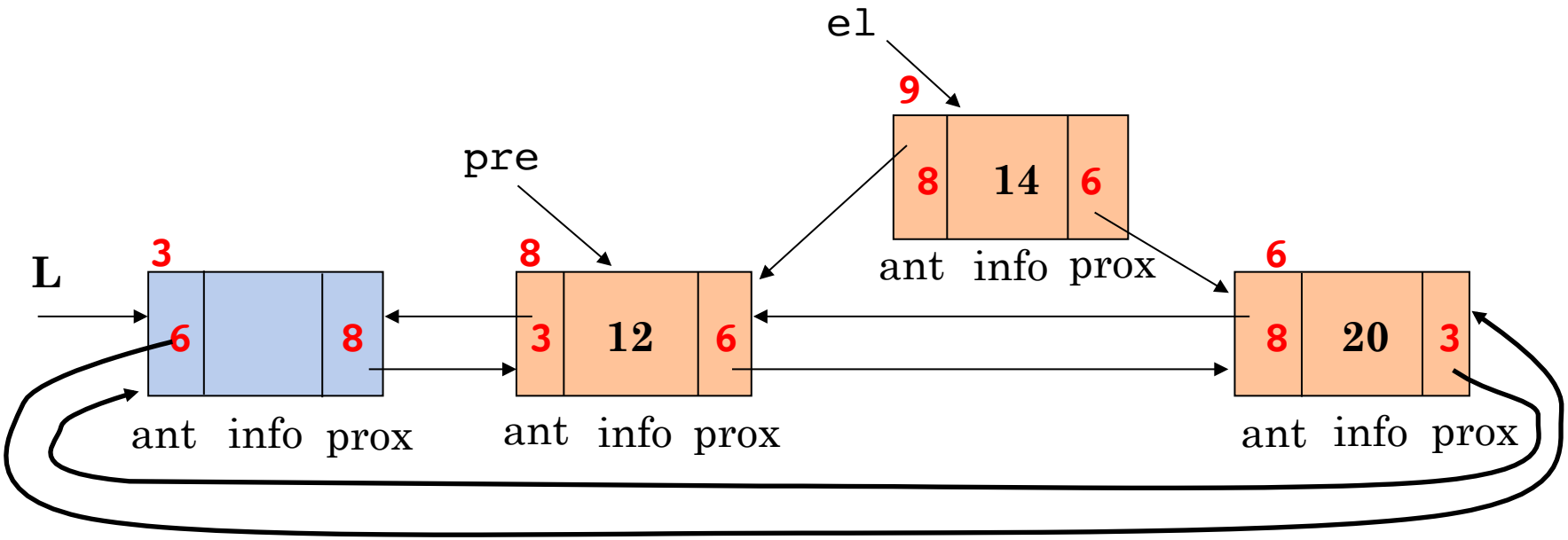

```
el = (lista *) malloc(sizeof(lista));
el->info = elem;
el->prox = ????;
el->ant = ????;
```

Inserindo um nó em L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 14)



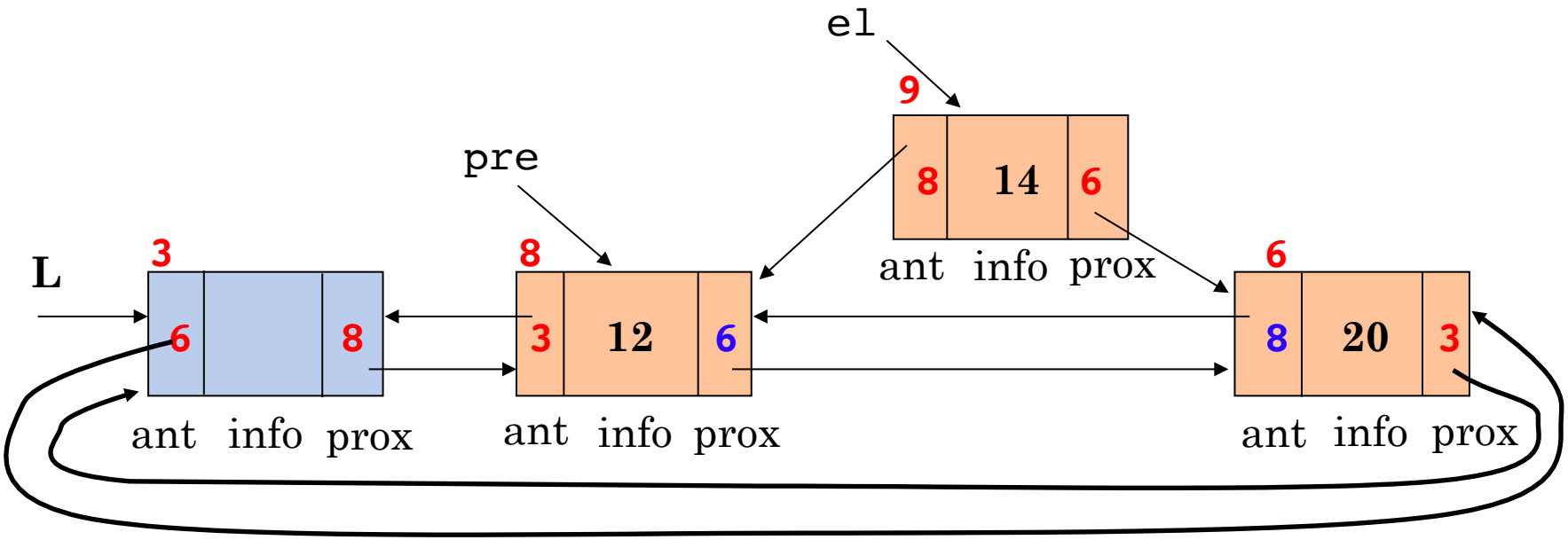

```
el = (lista *) malloc(sizeof(lista));
el->info = elem;
el->prox = pre->prox;
el->ant = ????;
```

Inserindo um nó em L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 14)



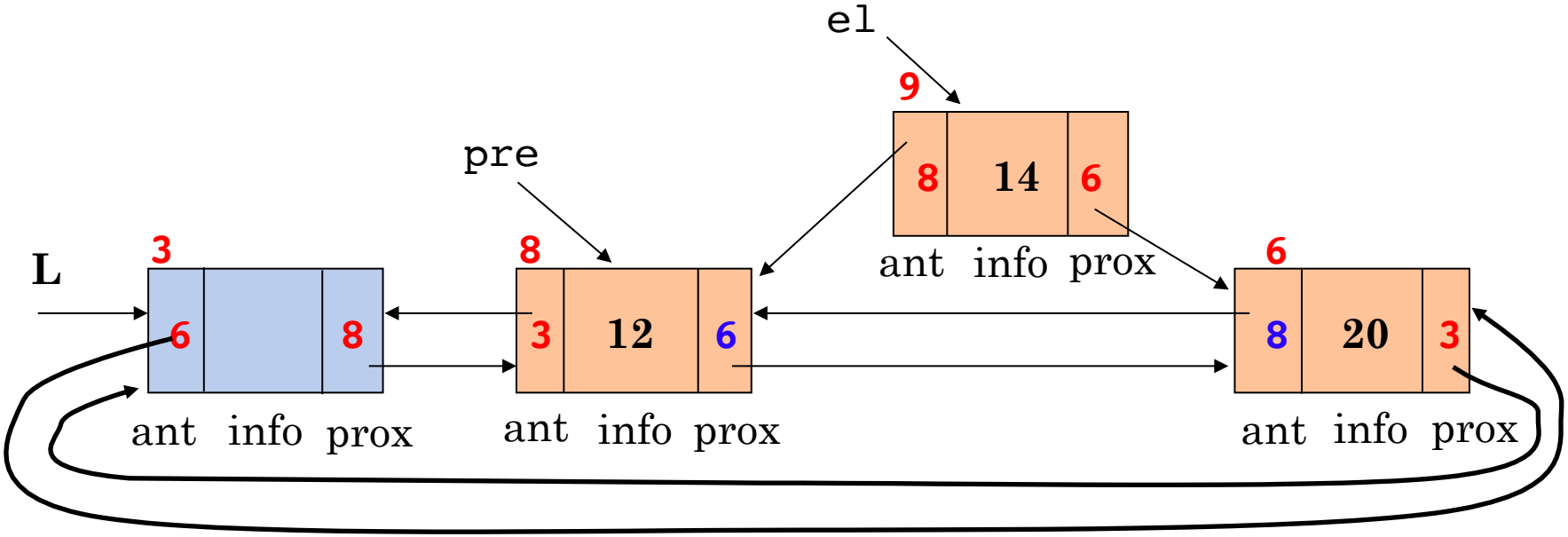

```
el = (lista *) malloc(sizeof(lista));
el->info = elem;
el->prox = pre->prox;
el->ant = pre;
```

Inserindo um nó em L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 14)



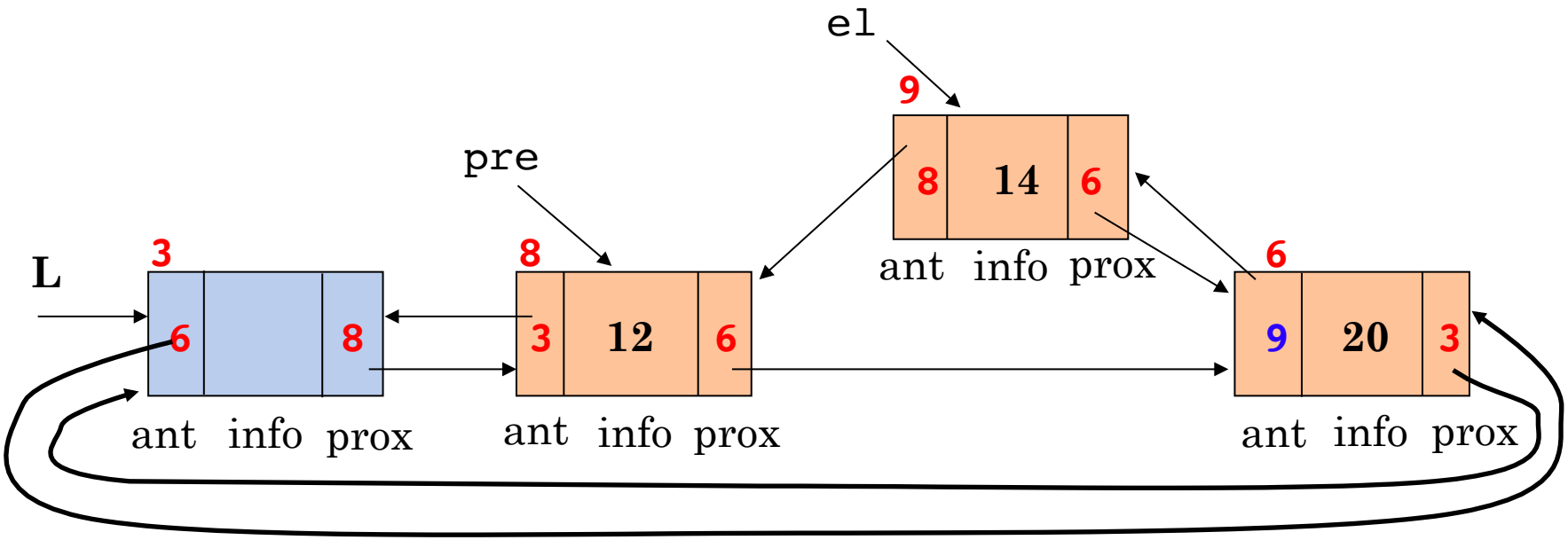

```
el = (lista *) malloc(sizeof(lista));
el->info = elem;
el->prox = pre->prox;
el->ant = pre;
```

Inserindo um nó em L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 14)



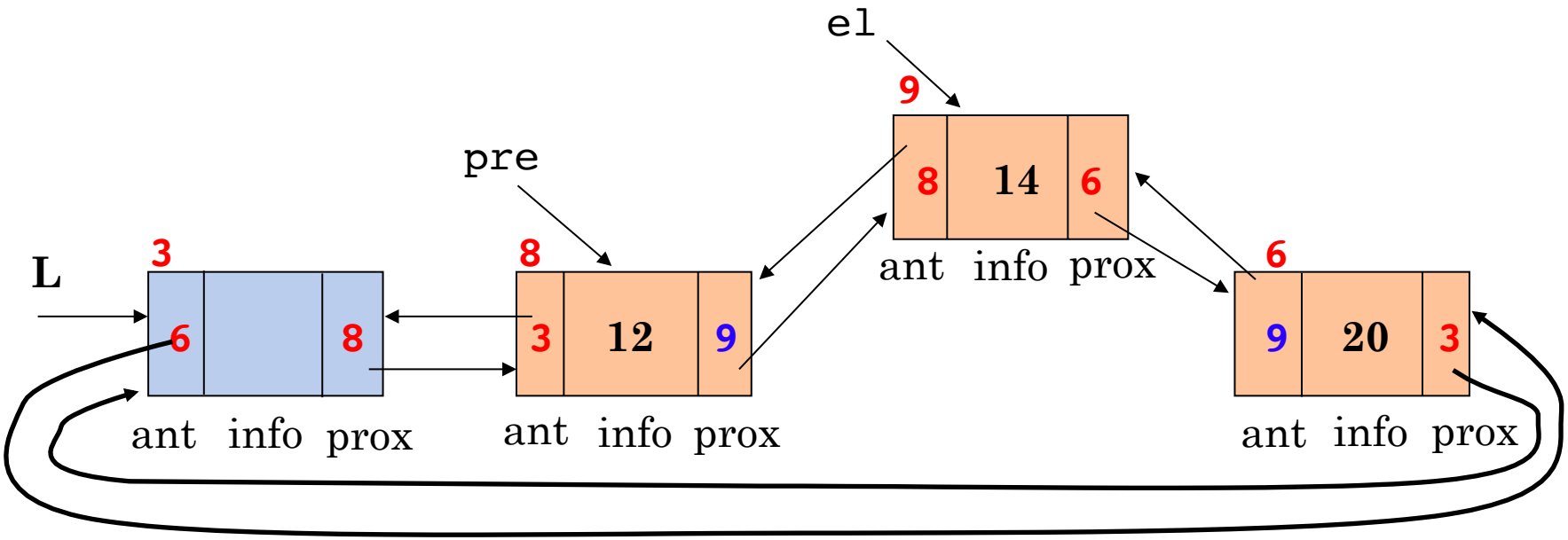

```
el = (lista *) malloc(sizeof(lista));
el->info = elem;
el->prox = pre->prox;
el->ant = pre;
pre->prox->ant = ????;
pre->prox = ????;
```

Inserindo um nó em L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 14)




```
el = (lista *) malloc(sizeof(lista));
el->info = elem;
el->prox = pre->prox;
el->ant = pre;
pre->prox->ant = el;
pre->prox = ????;
```

Inserindo um nó em L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 14)

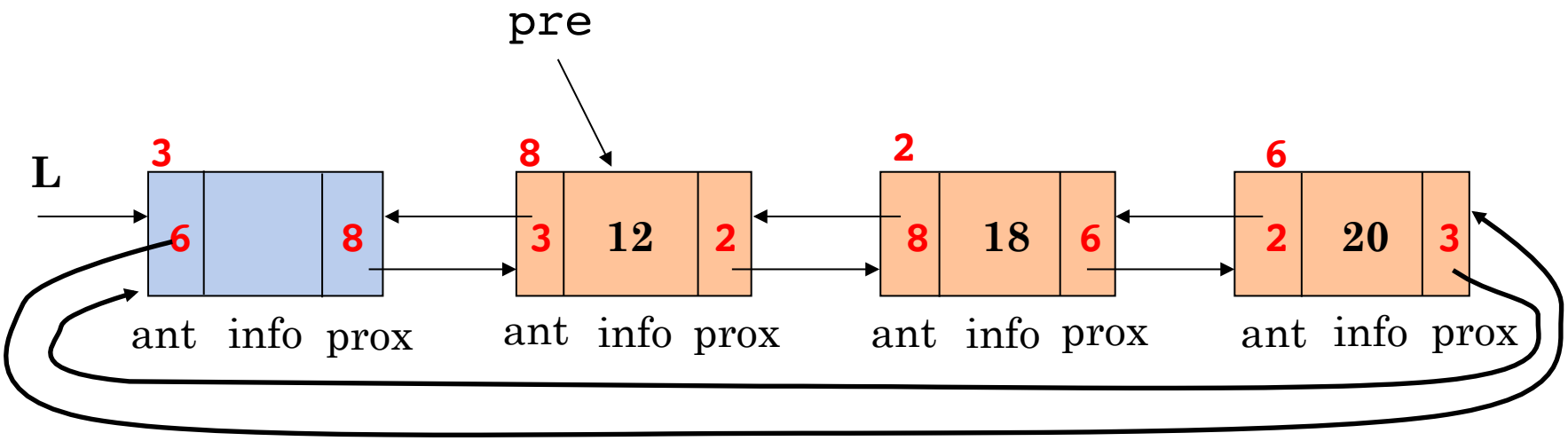



```
el = (lista *) malloc(sizeof(lista));
el->info = elem;
el->prox = pre->prox;
el->ant = pre;
pre->prox->ant = el;
pre->prox = el;
```

Função Insere Elemento em L, Duplamente Encadeada, Circular e Com Nó Cabeça

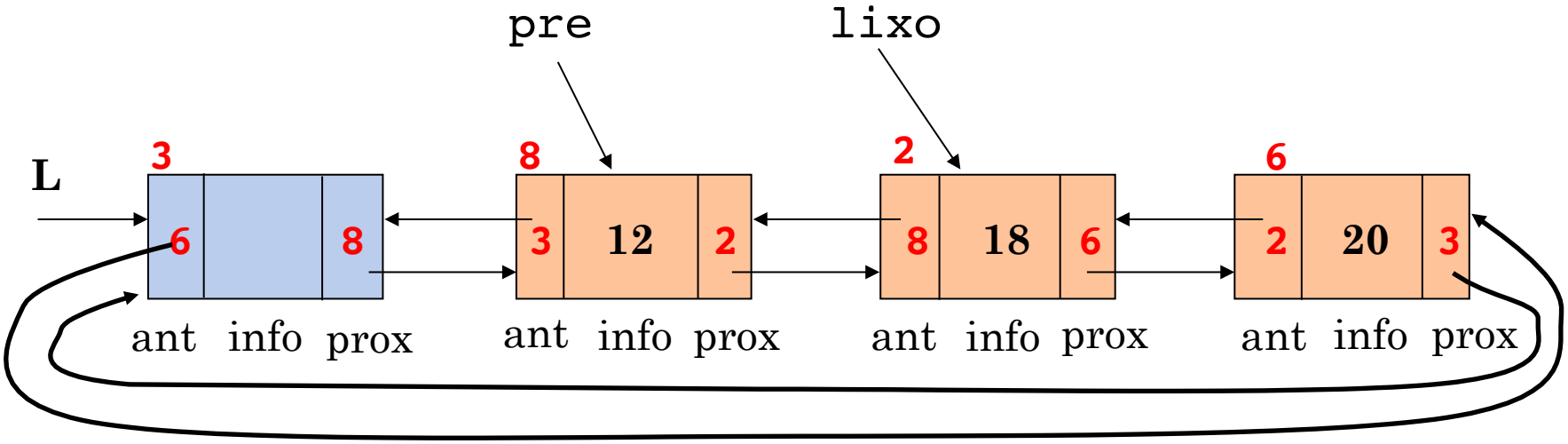
```
lista *insereElem(lista *L, int elem) {

    lista *pre, *el;

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

Removendo um nó de L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 18)



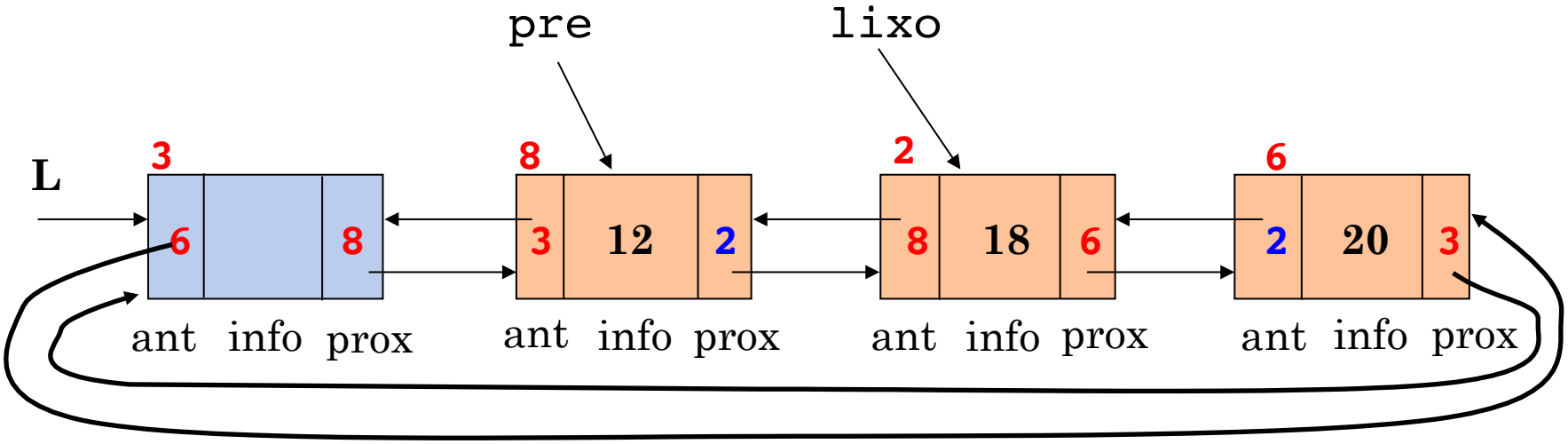

```
lixo = pre->prox;
```

Removendo um nó de L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 18)



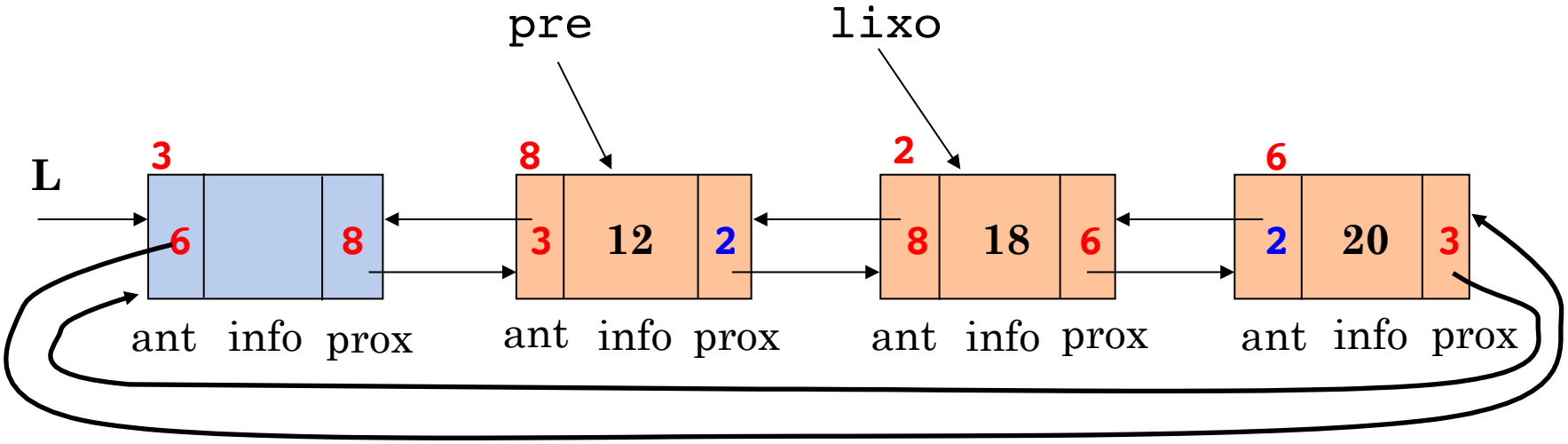

```
lixo = pre->prox;
```

Removendo um nó de L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 18)



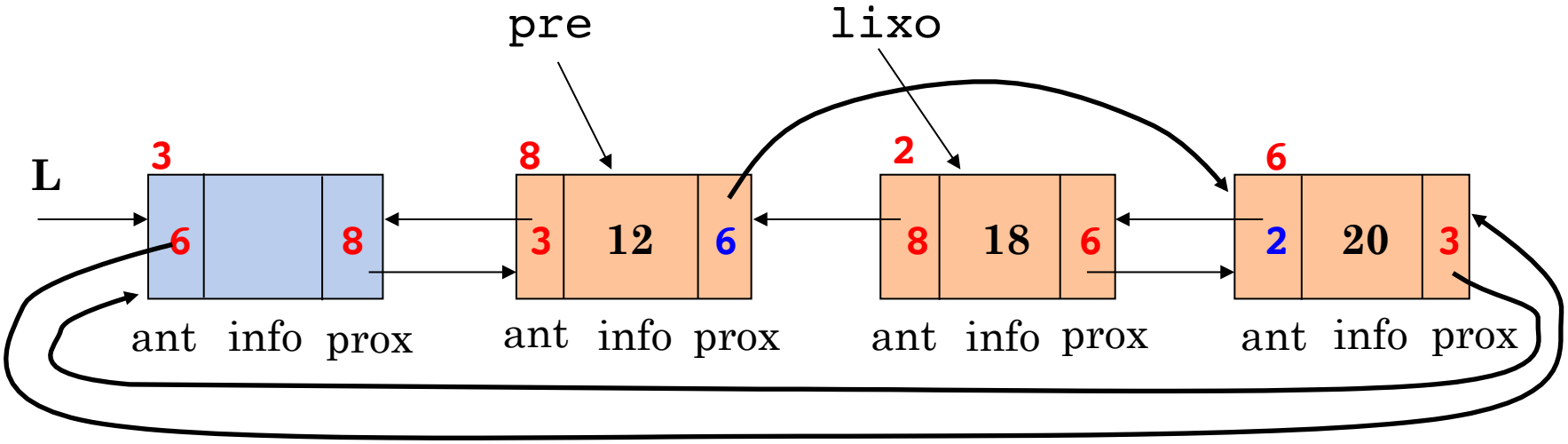

```
lixo = pre->prox;
```

Removendo um nó de L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 18)



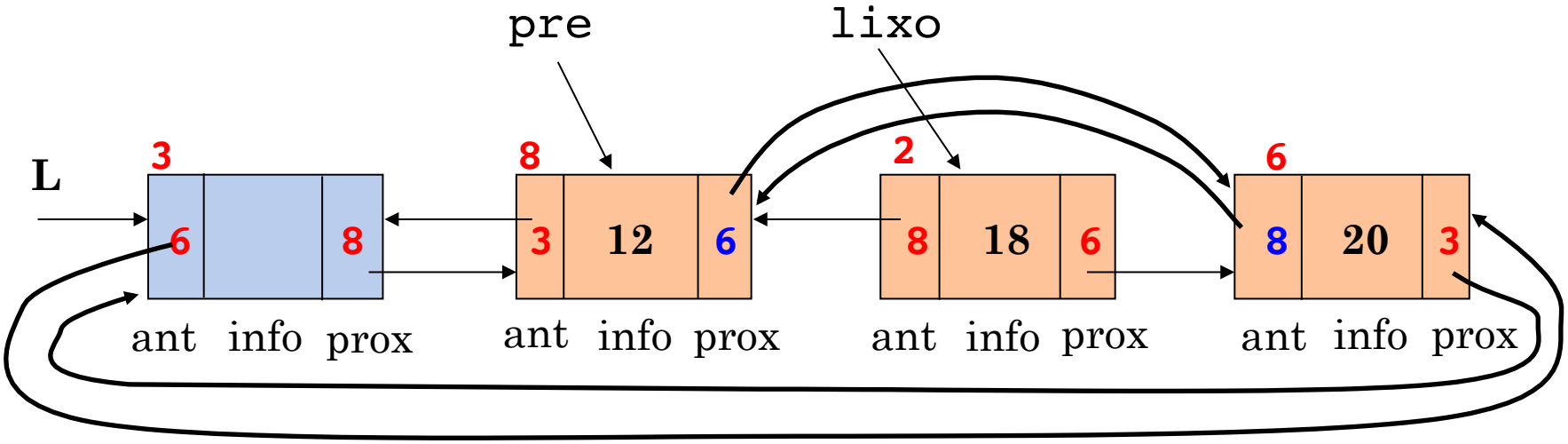

```
lixo = pre->prox;

lixo->ant->prox = ????;

lixo->prox->ant = ????;
```

Removendo um nó de L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 18)



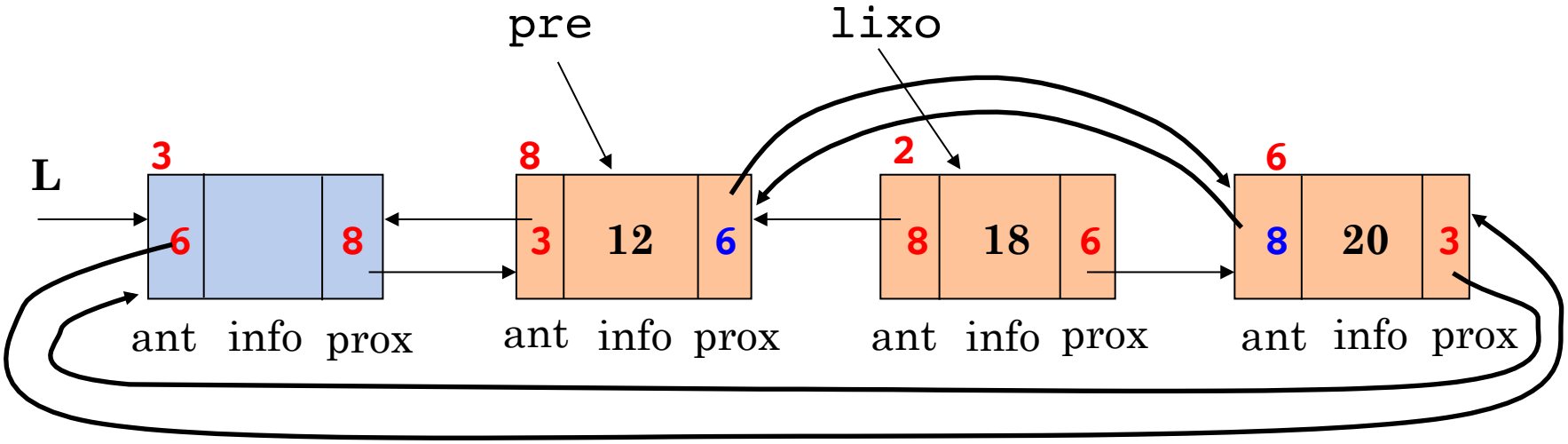

```
lixo = pre->prox;

lixo->ant->prox = lixo->prox;

lixo->prox->ant = ????;
```

Removendo um nó de L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 18)



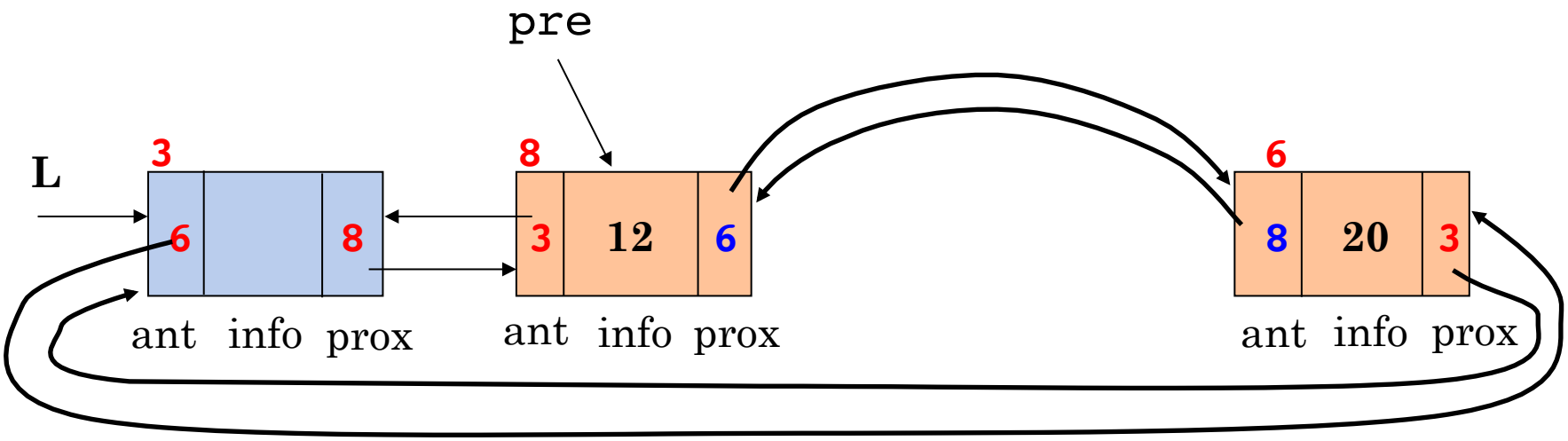

```
lixo = pre->prox;
lixo->ant->prox = lixo->prox;
lixo->prox->ant = lixo->ant;
```

Removendo um nó de L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 18)




```
lixo = pre->prox;
lixo->ant->prox = lixo->prox;
lixo->prox->ant = lixo->ant;
free(lixo);
```

Removendo um nó de L, Duplamente Encadeada, Circular e Com Nó Cabeça (elem = 18)




```
lixo = pre->prox;
lixo->ant->prox = lixo->prox;
lixo->prox->ant = lixo->ant;
free(lixo);
```

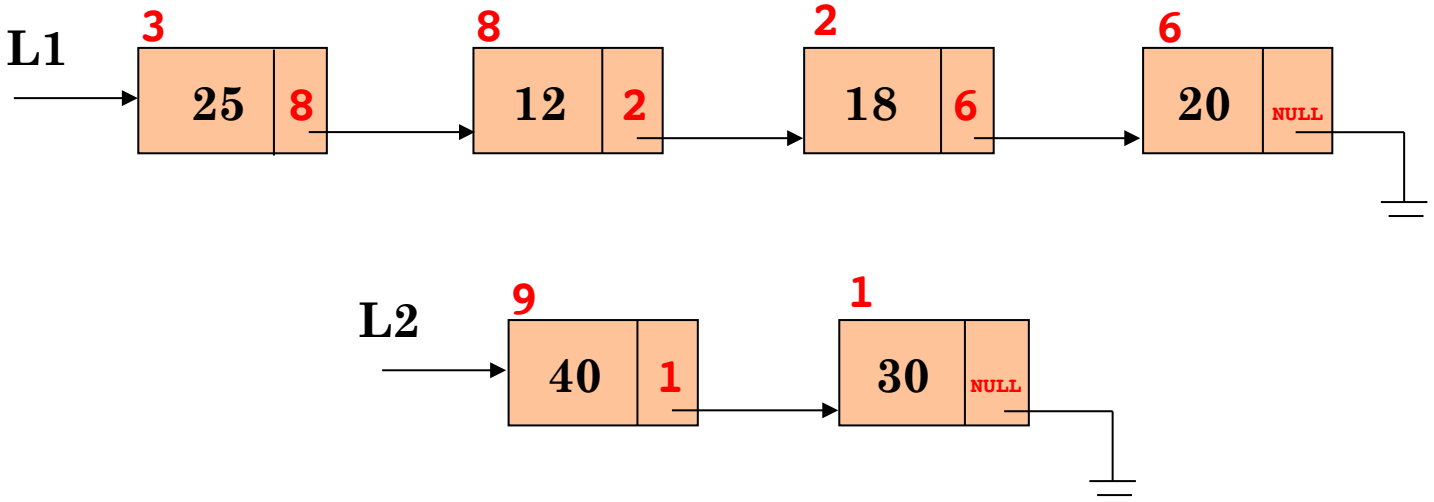
Função Remove Elemento em L, Duplamente Encadeada, Circular e Com Nó Cabeça

```
lista *removeElem(lista *L, int elem) {

    lista *pre, *lixo;

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

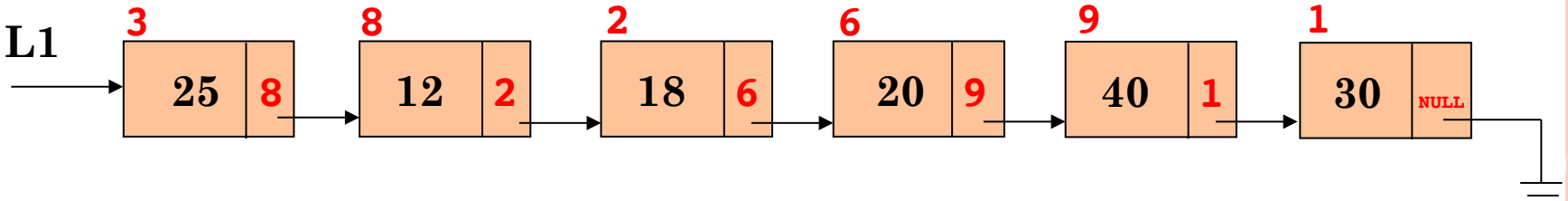
Exercício

- Considere duas listas encadeadas L1 e L2, onde cada nó da lista possui duas partes: dado e prox. Para cada item abaixo, faça uma função que receba as duas listas como parâmetros e as concatene de acordo com as características especificadas.
- Para concatenar L1 com L2 basta juntar o final de L1 com o começo de L2, formando uma única lista com endereço inicial em L1.
  a) L1 e L2 são simplesmente encadeadas, não circulares e sem nós cabeça
  b) L1 e L2 são simplesmente encadeadas, circulares e sem nós cabeça → NÃO
  c) L1 e L2 são simplesmente encadeadas, circulares e com nós cabeça
  d) L1 e L2 são simplesmente encadeadas, não circulares e com nós cabeça
  e) L1 e L2 são duplamente encadeadas, circulares e com nós cabeça

A) L1 e L2 são simplesmente encadeadas, não circulares e sem nós cabeça

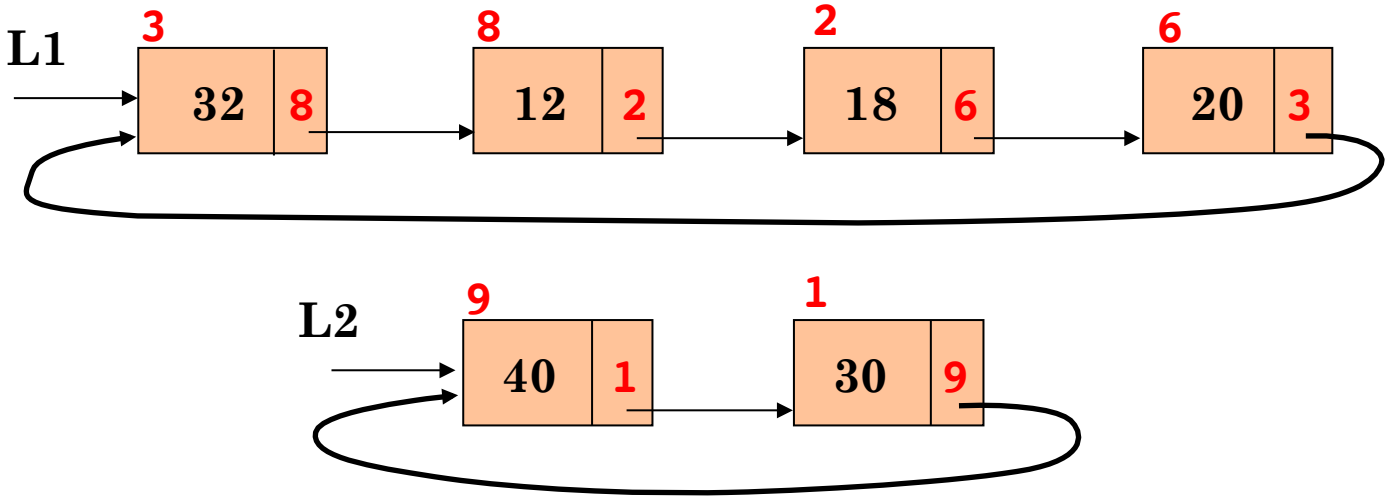


Resultado:

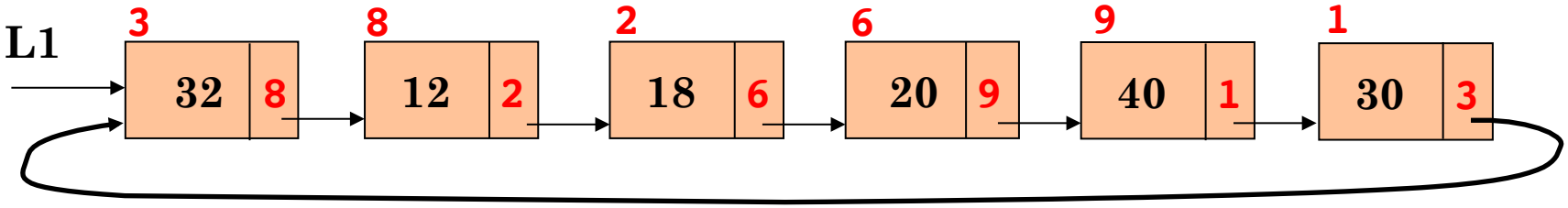


Obs: Também deve ser testado se alguma das listas está vazia

B) L1 e L2 são simplesmente encadeadas, circulares e sem nós cabeça

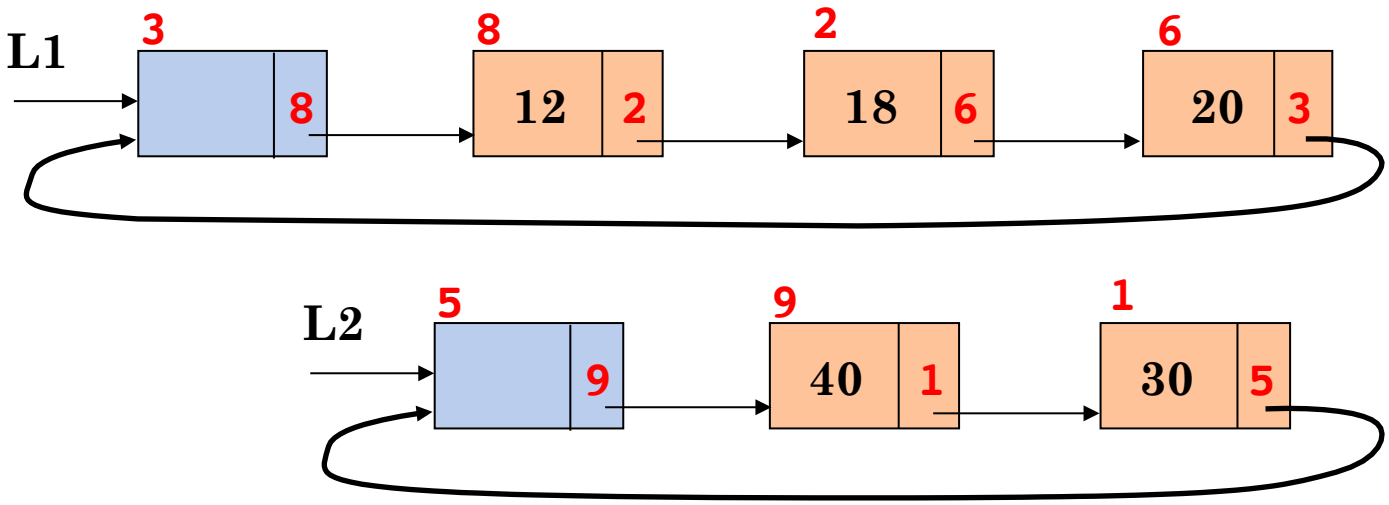


Resultado:

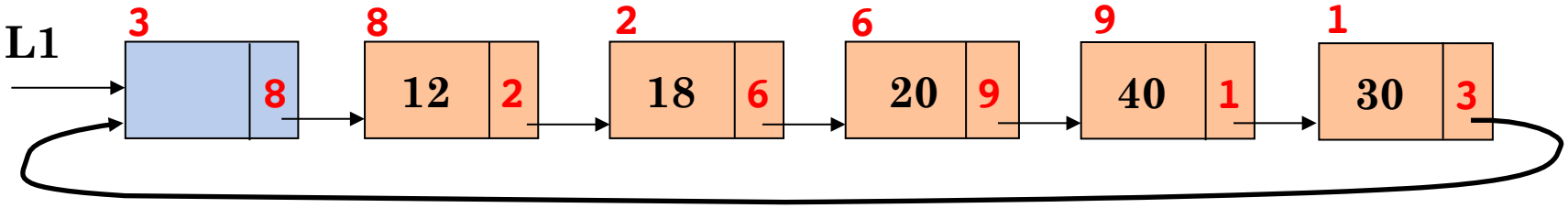


Obs: Também deve ser testado se alguma das listas está vazia

C) L1 e L2 são simplesmente encadeadas, circulares e com nós cabeça

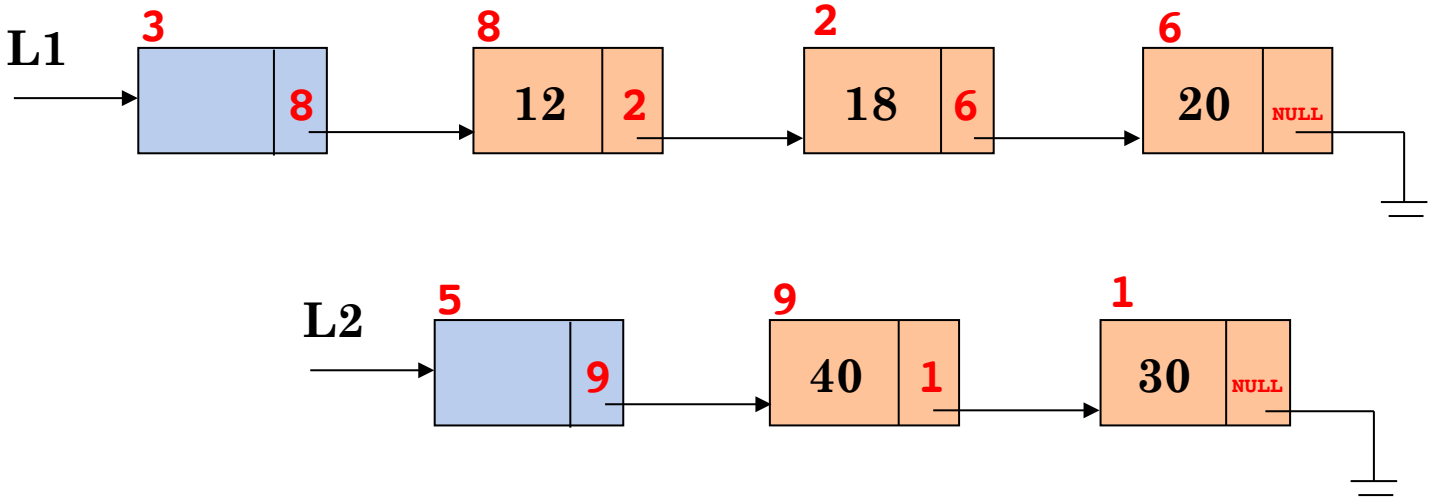


Resultado:

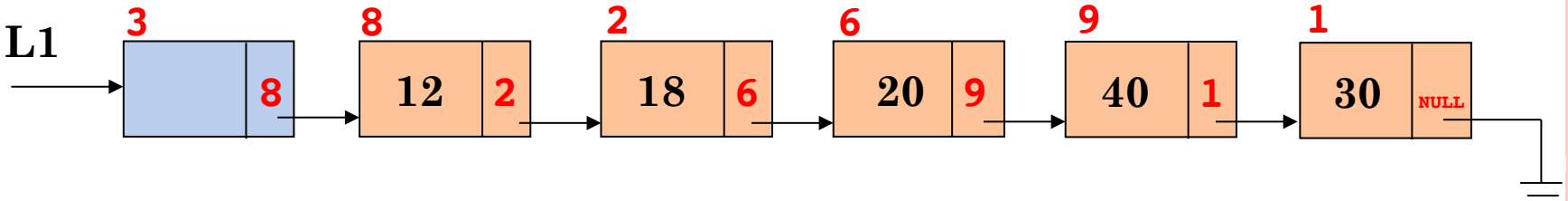


Obs: Também deve ser testado se alguma das listas está vazia

D) L1 e L2 são simplesmente encadeadas, não circulares e com nós cabeça

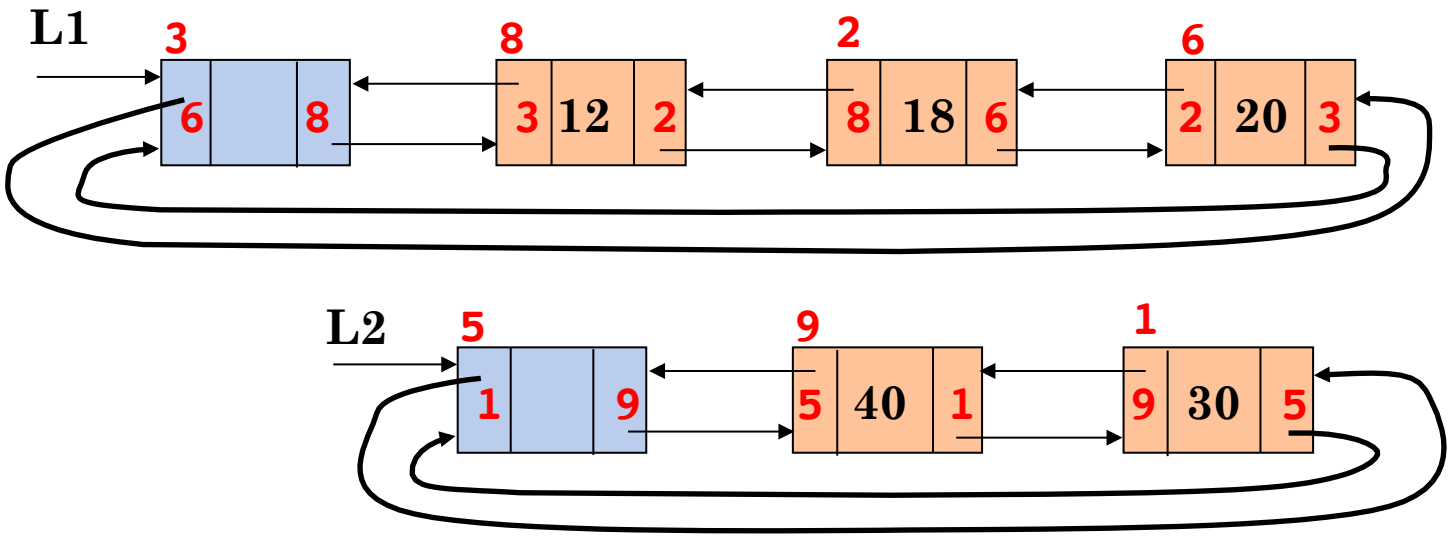


Resultado:

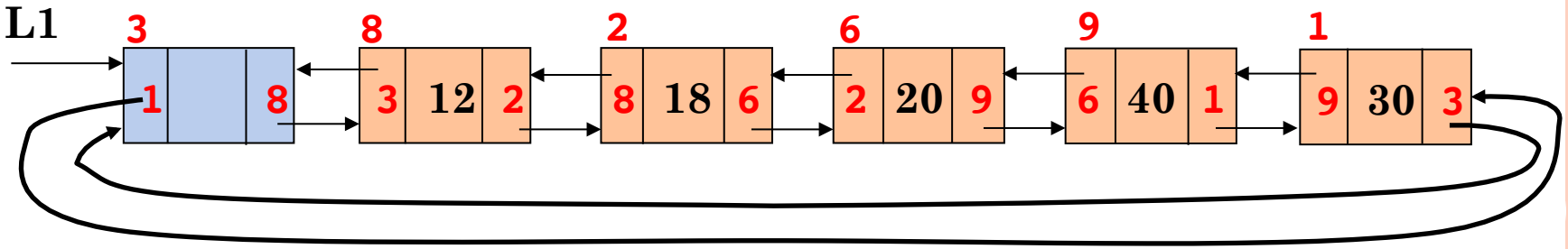


Obs: Também deve ser testado se alguma das listas está vazia

E) L1 e L2 são duplamente encadeadas, circulares e com nós cabeça



Resultado:



Obs: Também deve ser testado se alguma das listas está vazia