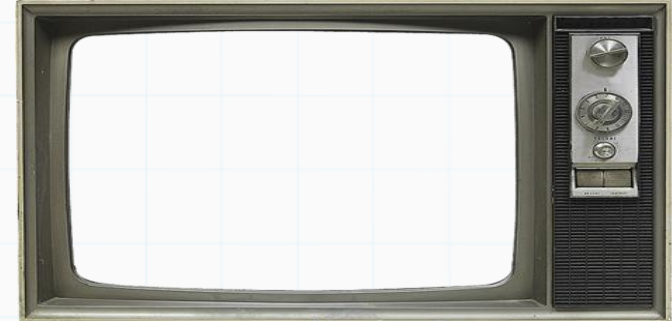


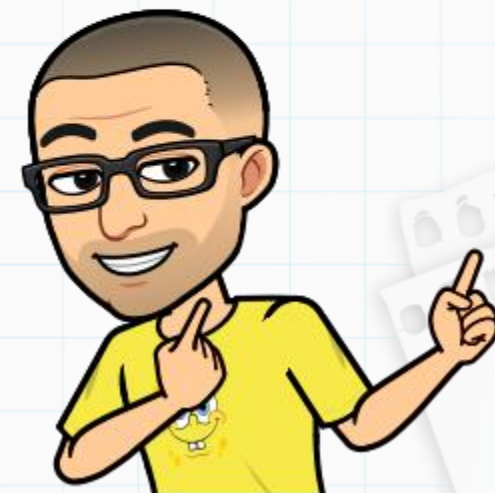
Programação Estruturada

Professor : Yuri Frota

yuri@ic.uff.br



HI



Listas encadeadas

Listas Lineares:

Uma lista agrupa um conjunto de elementos que se relacionam entre si

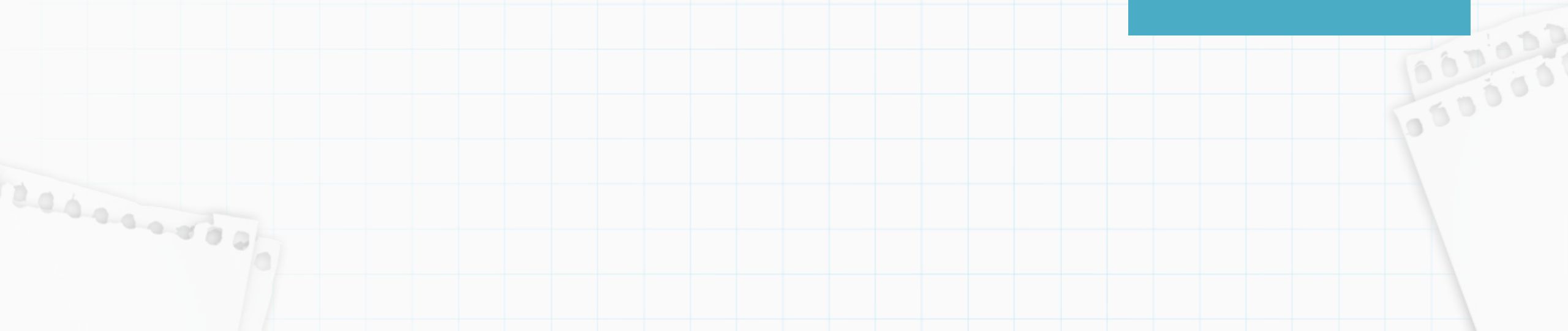
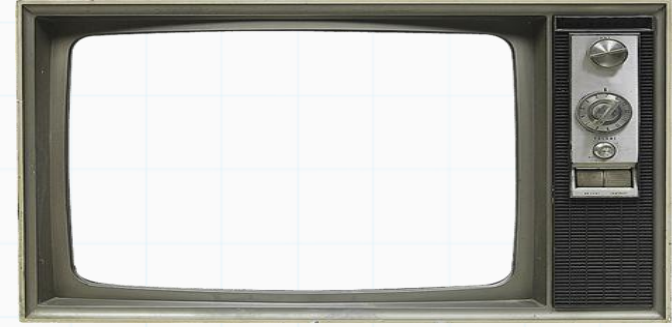
Uma lista linear é um conjunto de $n \geq 0$ elementos:

$$L[1], L[2], L[3], \dots, L[n]$$

Tais que:

Se $n > 0$, então $L[1]$ é o primeiro elemento

Para $1 < k \leq n$, o elemento $L[k]$ é precedido por $L[k-1]$



Listas encadeadas

Listas Lineares:

Uma lista agrupa um conjunto de elementos que se relacionam entre si

Uma lista linear é um conjunto de $n \geq 0$ elementos:

$$L[1], L[2], L[3], \dots, L[n]$$

Tais que:

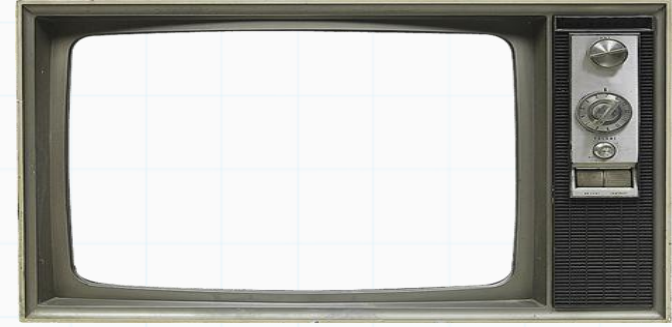
Se $n > 0$, então $L[1]$ é o primeiro elemento

Para $1 < k \leq n$, o elemento $L[k]$ é precedido por $L[k-1]$

Exemplo:



As operações principais de uma lista são: *inclusão, remoção e busca* de elementos



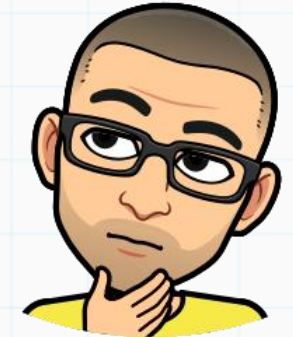
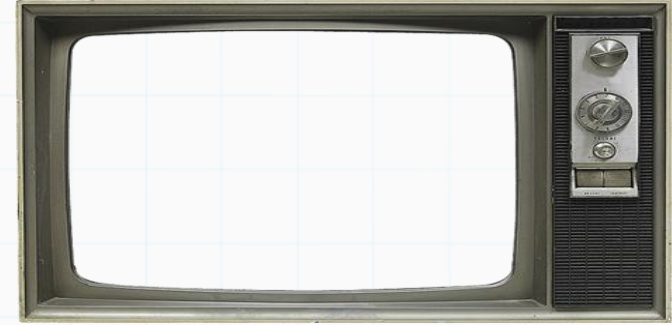
Listas encadeadas

Listas Lineares:

Podemos implementar as listas lineares utilizando **vetores**

Pontos positivos:

- É a maneira mais simples
- Os elementos são armazenados em posições consecutivas na memória
- é possível ter acesso direto a qualquer elemento da lista



65	82	4	33
----	----	---	----



Listas encadeadas

Listas Lineares:

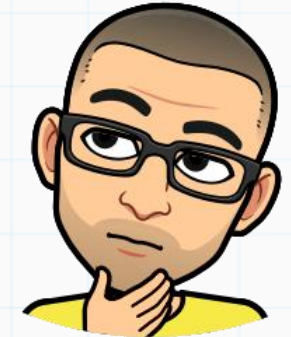
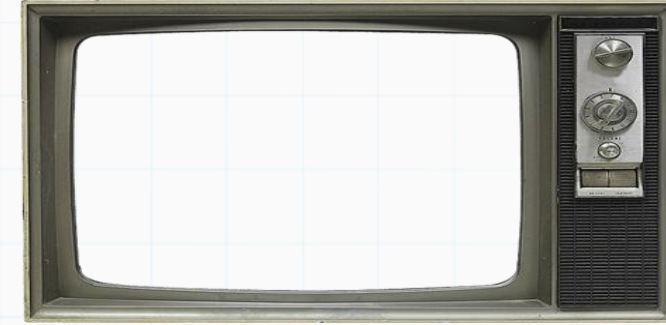
Podemos implementar as listas lineares utilizando **vetores**

Pontos positivos:

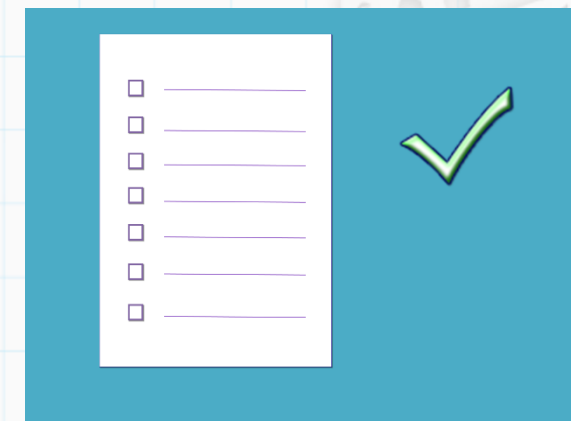
- É a maneira mais simples
- Os elementos são armazenados em posições consecutivas na memória
- é possível ter acesso direto a qualquer elemento da lista

Pontos negativos:

- Problema com dimensionamento (uma vez alocada a memória não pode diminuir ou aumentar)
- Inclusão e remoção de elementos não ocorrem de fato



65	82	4	33
----	----	---	----



Listas encadeadas

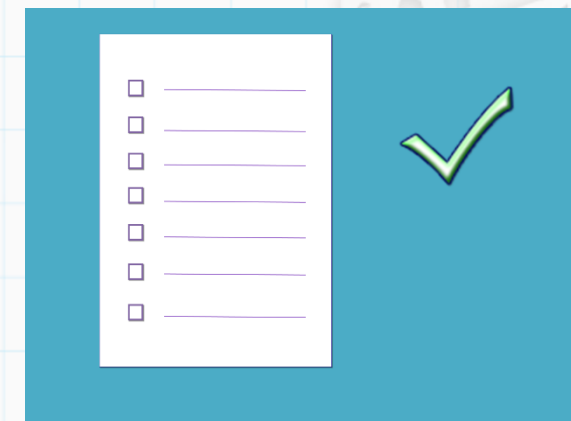
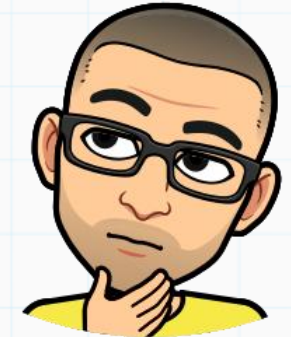
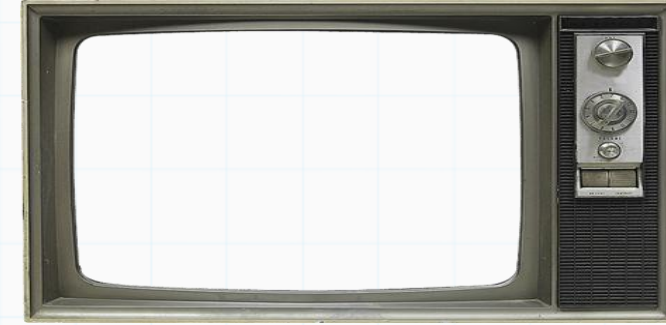
Listas Lineares:

Uma outra forma de implementar é utilizando **alocação encadeada dinâmica**.

Pontos positivos:

- As posições de memória para cada elemento da lista são alocadas a medida que se tornam necessárias, ou seja, a estrutura pode aumentar e diminuir em tempo de execução.

- As inclusões e remoções de elementos ocorrem de fato, com alocação e liberação de memória.



Listas encadeadas

Listas Lineares:

Uma outra forma de implementar é utilizando **alocação encadeada dinâmica**.

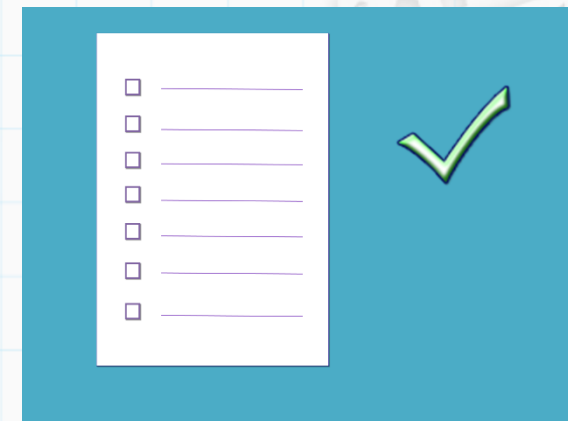
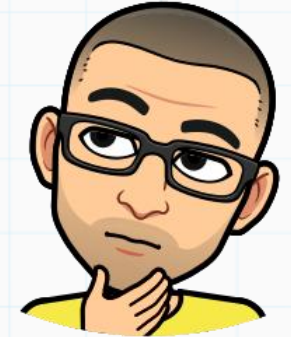
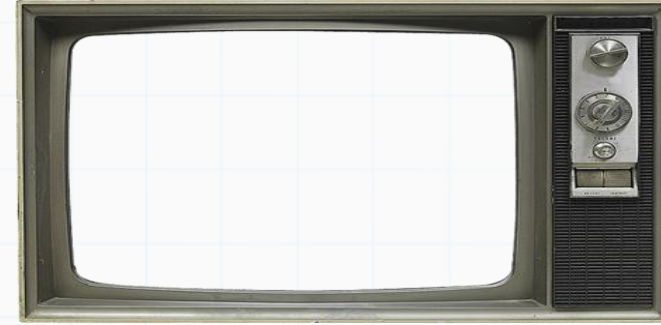
Pontos positivos:

- As posições de memória para cada elemento da lista são alocadas a medida que se tornam necessárias, ou seja, a estrutura pode aumentar e diminuir em tempo de execução.

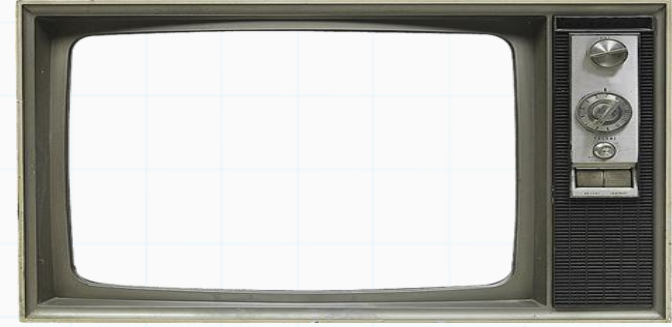
- As inclusões e remoções de elementos ocorrem de fato, com alocação e liberação de memória.

Pontos negativos:

- Não existe acesso direto a um elemento da lista (não é indexado).



Listas encadeadas

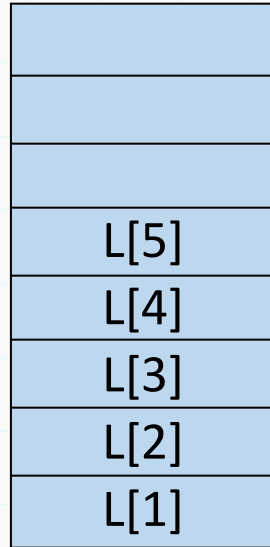


Listas Lineares:

Na memória, como se comportam:

VETORES

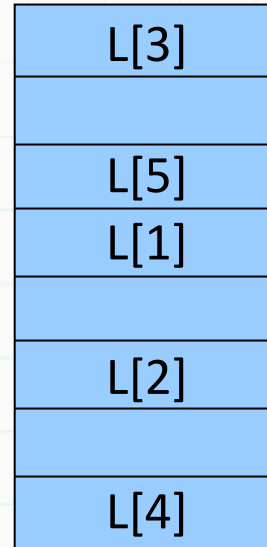
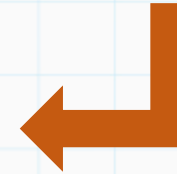
Armazenamento
Contíguo



Memória

LISTAS ENCADEADAS

Armazenamento
Não-Contíguo



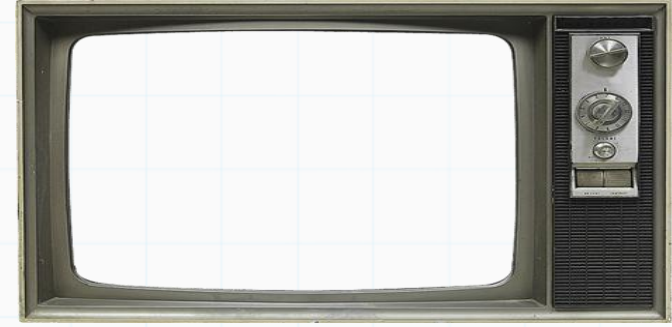
Memória



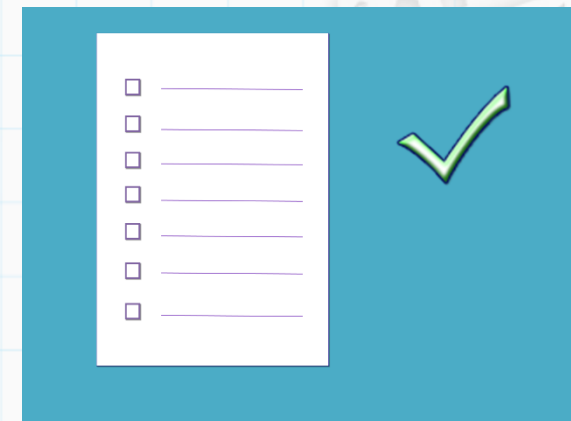
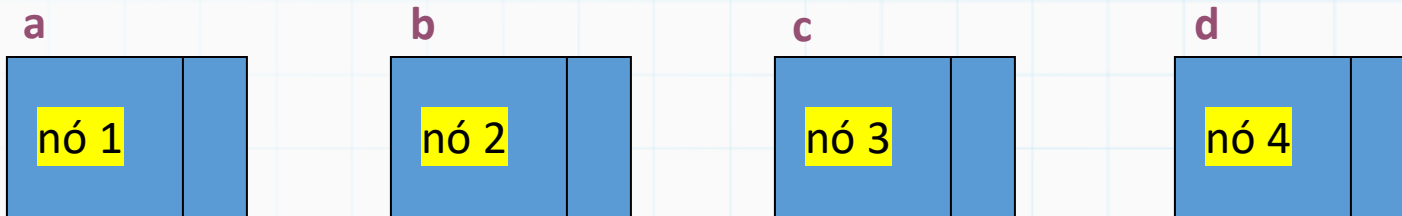
Listas encadeadas

Listas Encadeadas:

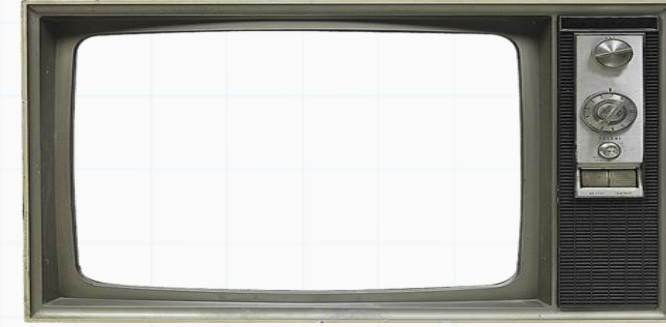
- O elementos são chamados de nós



endereço



Listas encadeadas

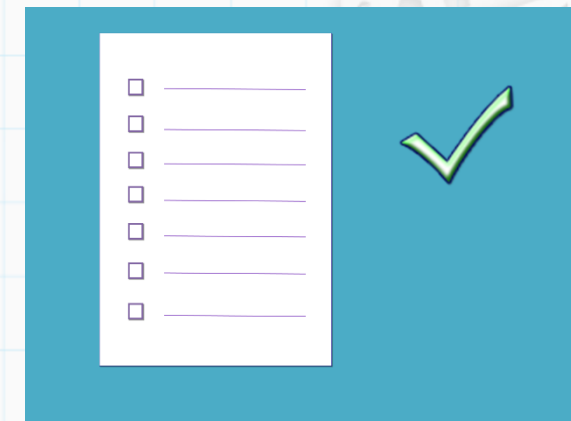
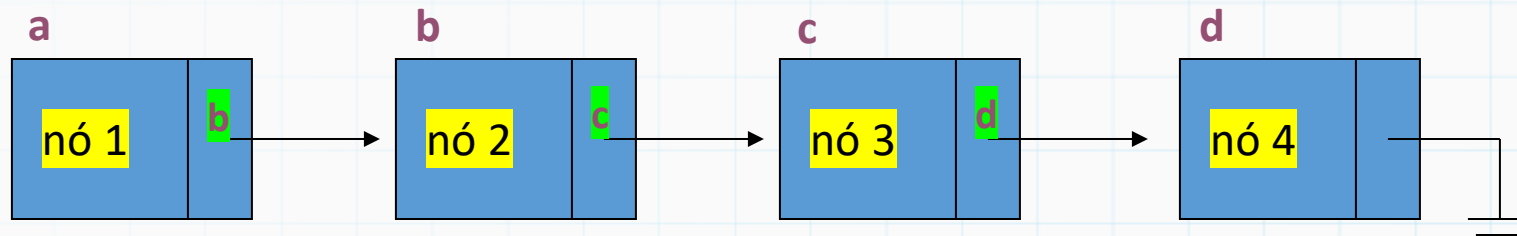


Listas Encadeadas:

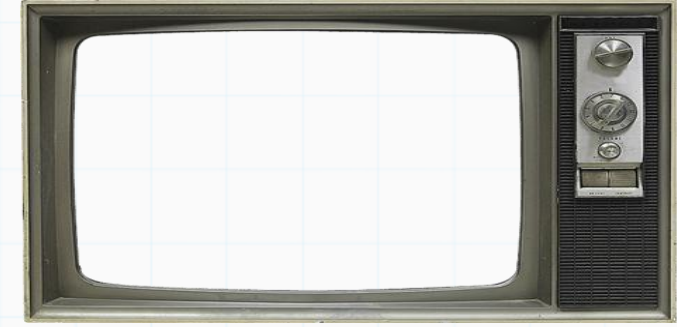
- Os elementos são chamados de nós
- Como os elementos não se encontram em posições contíguas de memória é necessário que cada elemento saiba o endereço do próximo
- Logo, cada nó é composto por duas partes:

endereço

A informação do elemento
O endereço do próximo nó → ponteiro



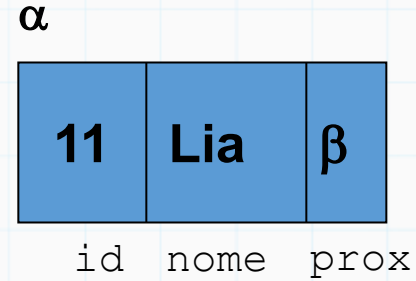
Listas encadeadas



Exemplo 1:

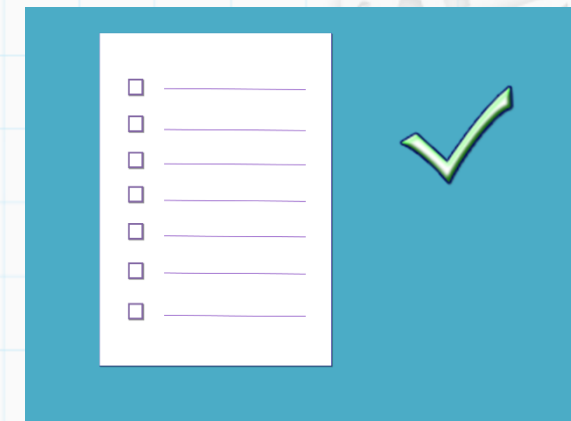
- Suponha uma turma com n alunos.
- Cada elemento possui uma matricula (id) e o nome de um aluno.

Representação gráfica do nó:

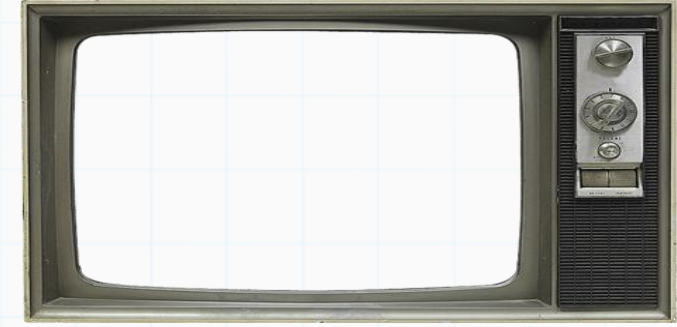


α é o endereço de memória de onde está armazenada a estrutura

β é o ponteiro para o próximo elemento



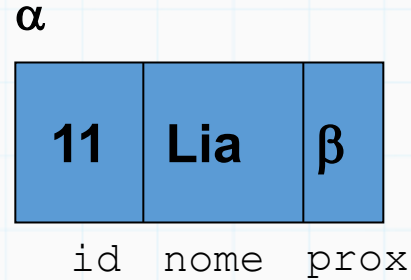
Listas encadeadas



Exemplo 1:

- Suponha uma turma com n alunos.
- Cada elemento possui uma matricula (id) e o nome de um aluno.

Representação gráfica do nó:



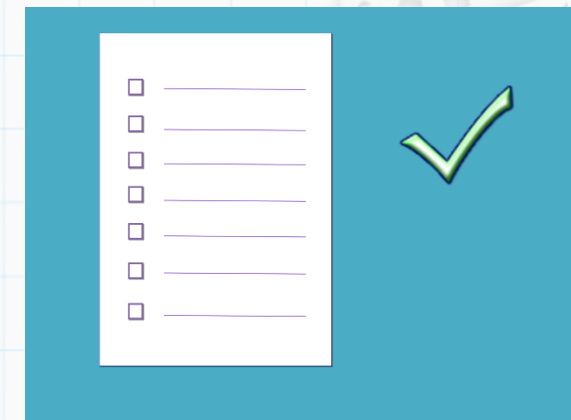
α é o endereço de memória de onde está armazenada a estrutura

β é o ponteiro para o próximo elemento

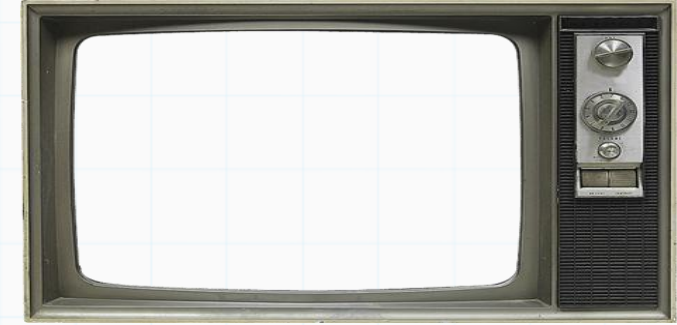
Estrutura :

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

...
lista *L;
```

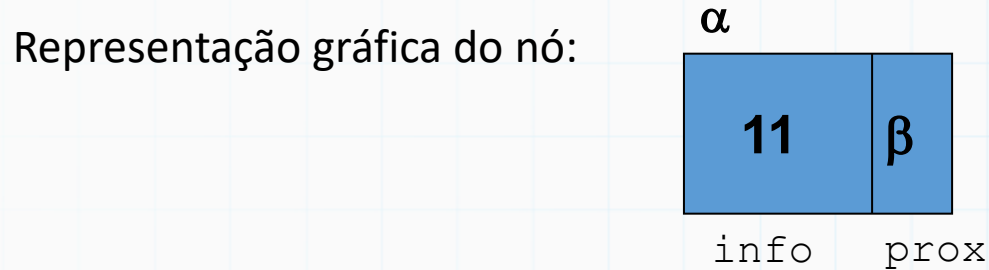


Listas encadeadas



Exemplo 2:

- Podemos também identificar o aluno apenas com a matrícula, como ficaria:



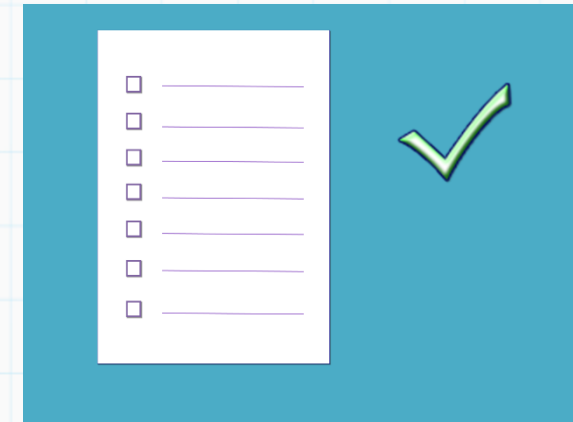
α é o endereço de memória de onde está armazenada a estrutura

β é o ponteiro para o próximo elemento

Estrutura :

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

...
lista *L = NULL;
```



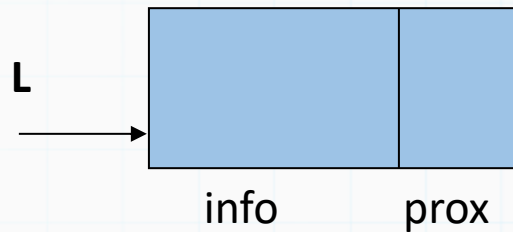
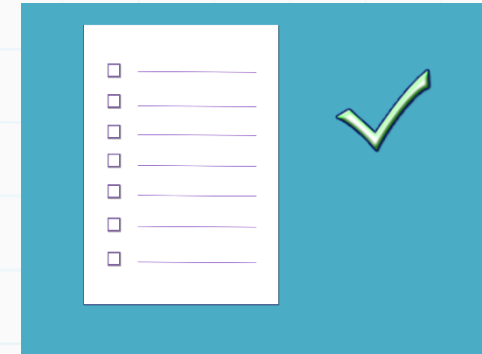
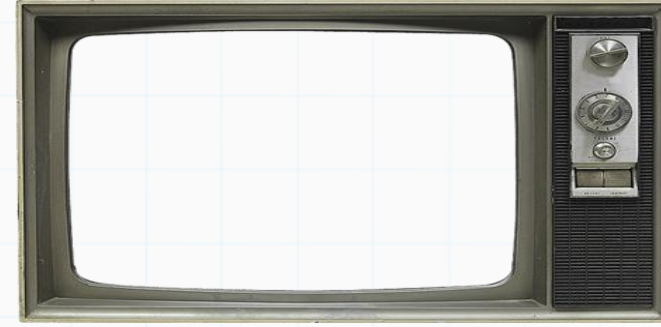
Iniciamos sempre a lista como NULL, que irá servir para marcar o fim da lista.

Listas encadeadas

Alocação: Para alocar 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));  
L = aux;
```



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

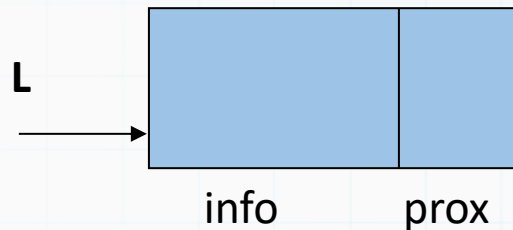
Listas encadeadas

Alocação: Para alocar um novo nó, deve-se especificar no comando de alocação, a variável ponteiro que irá guardar o novo endereço alocado.

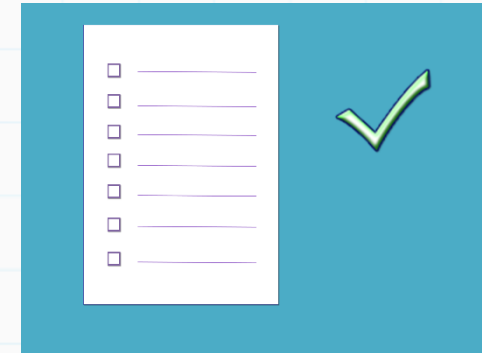
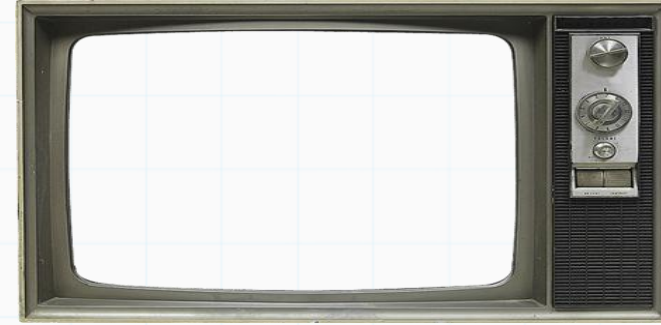
Exemplo:

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

- O endereço inicial da lista (L) deve sempre ser preservado, é a partir dele que é possível percorrer toda a lista (ele é o ponteiro para o início da lista)



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



Listas encadeadas

Alocação: Para alocar um novo nó, deve-se especificar no comando de alocação, a variável ponteiro que irá guardar o novo endereço alocado.

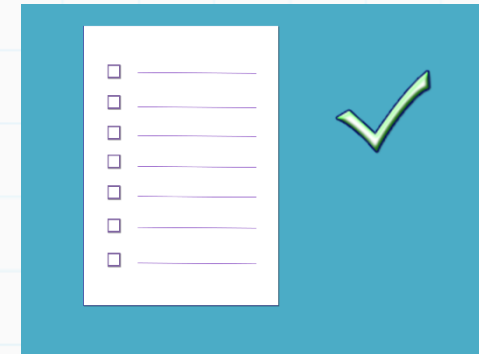
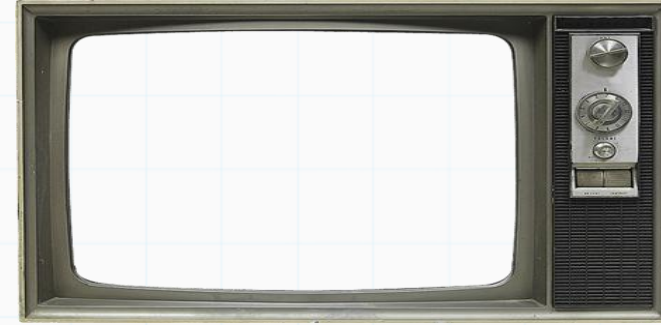
Exemplo:

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

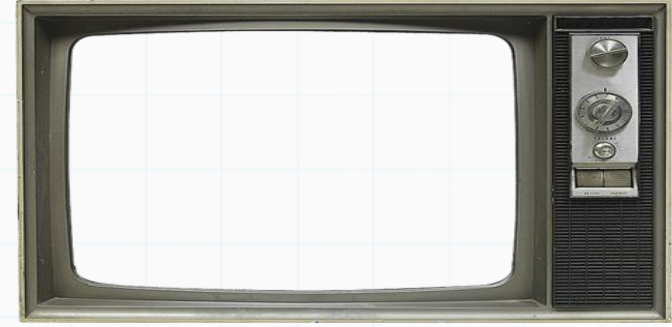
- O endereço inicial da lista (L) deve sempre ser preservado, é a partir dele que é possível percorrer toda a lista (ele é o ponteiro para o início da lista)
- A lista sempre termina apontando para nulo (NULL) para indicar o fim da lista



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



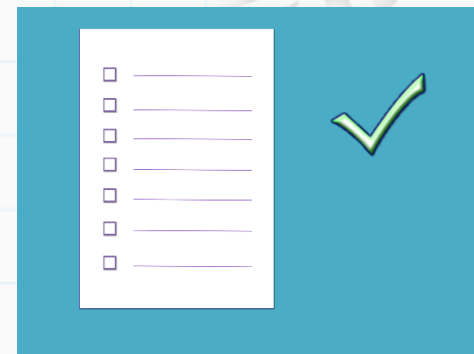
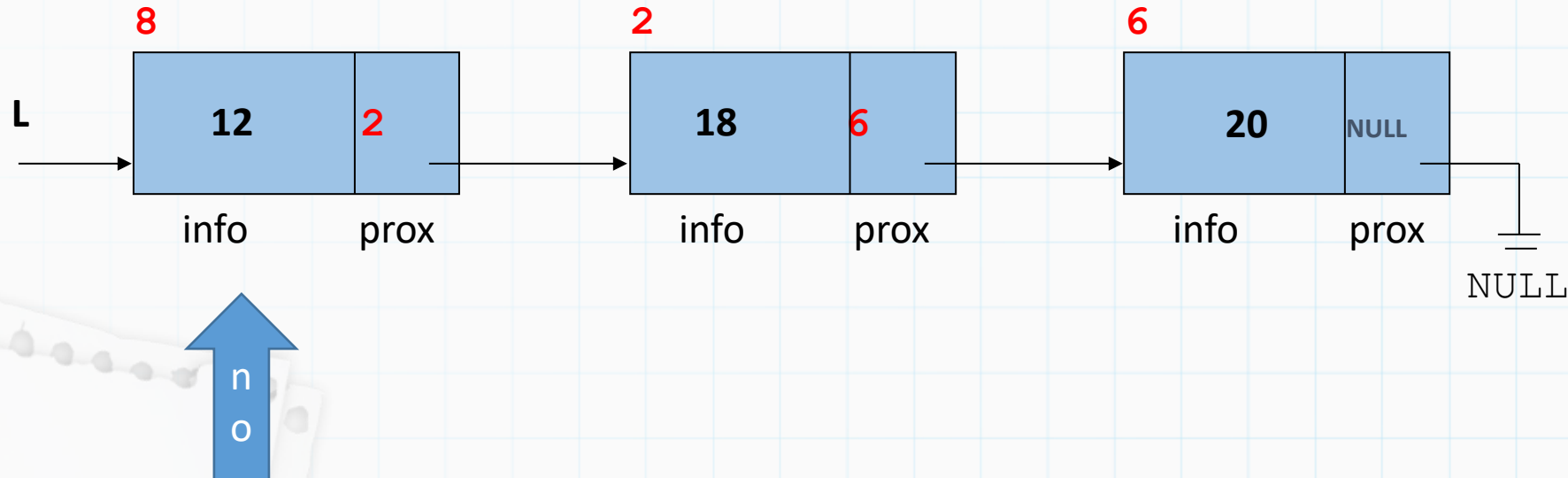
Listas encadenadas



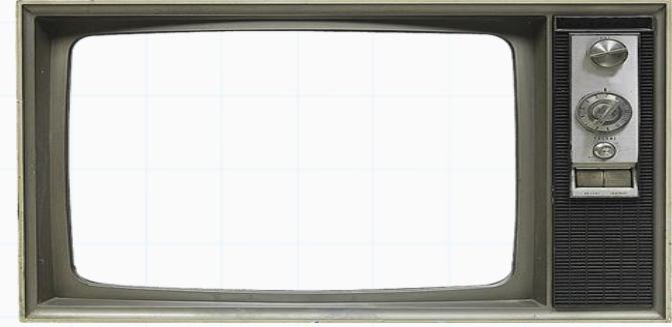
Percorrer:

```
lista *no;  
no =L;  
while (no != NULL)  
{  
    printf("%d, ", no->info);  
    no = no->prox;  
}
```

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



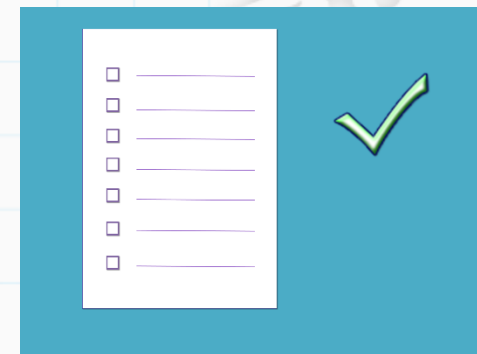
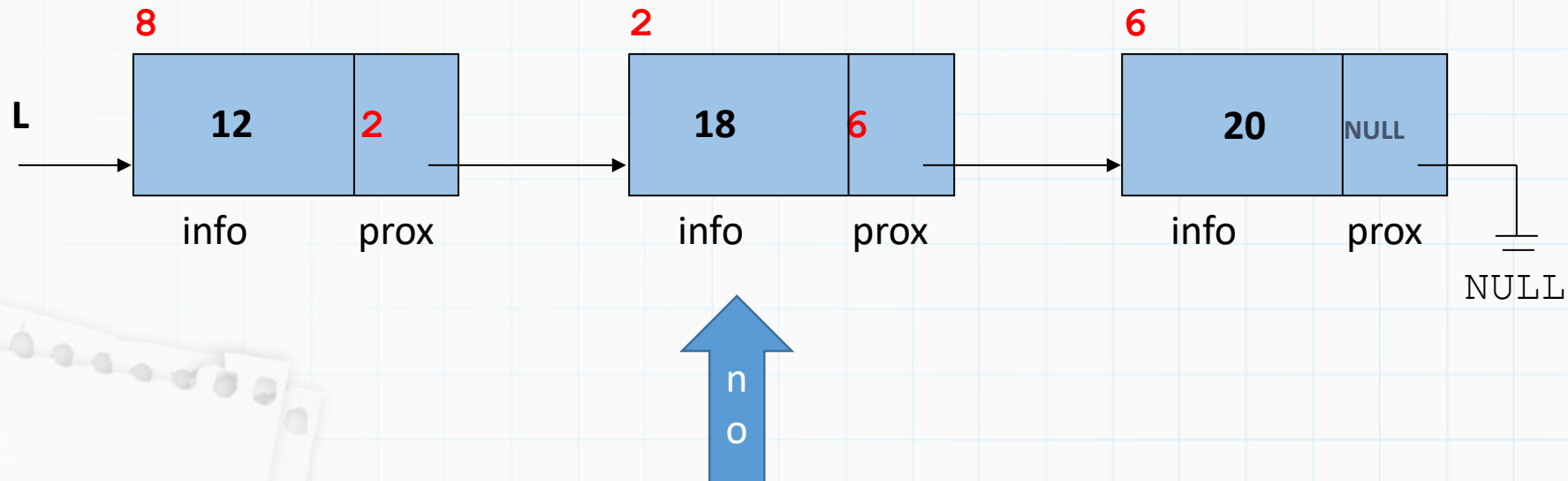
Listas encadenadas



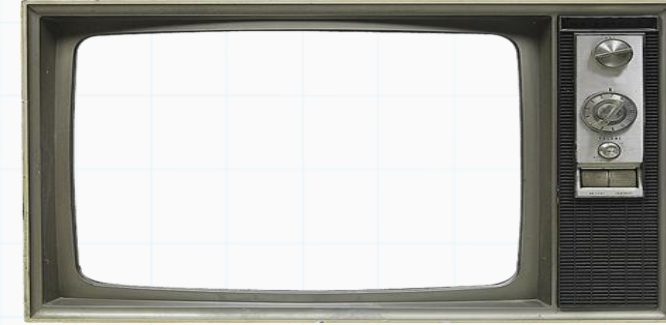
Percorrer:

```
lista *no;  
no =L;  
while (no != NULL)  
{  
    printf("%d, ", no->info);  
    no = no->prox;  
}
```

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



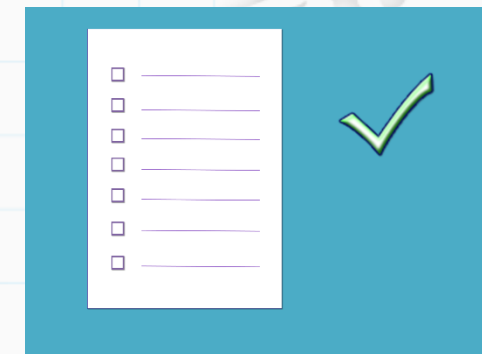
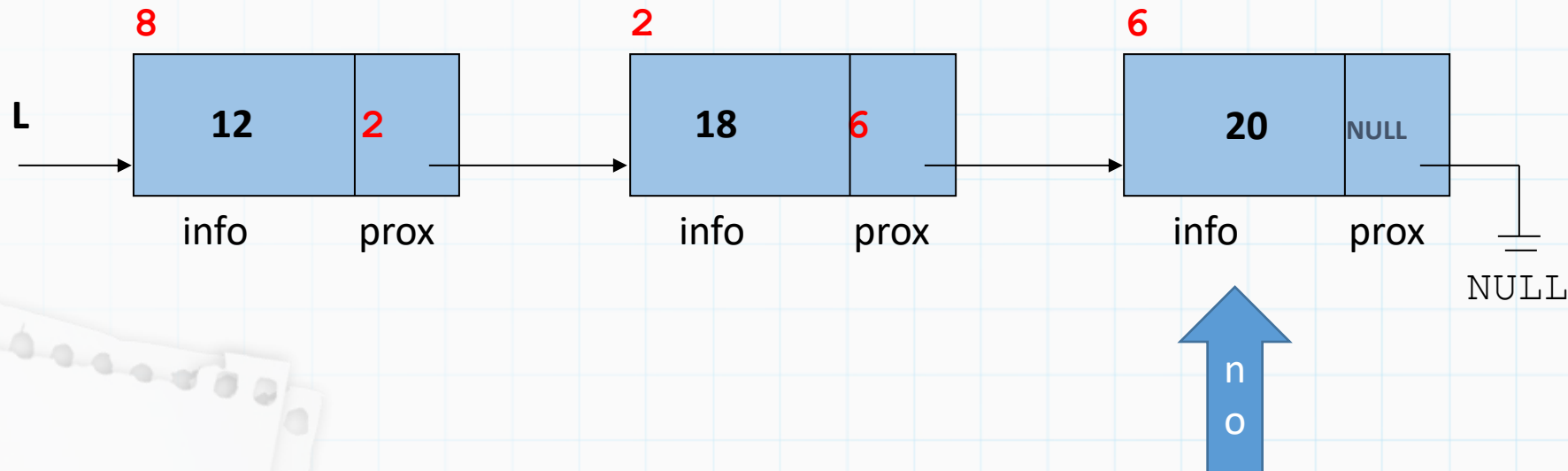
Listas encadenadas



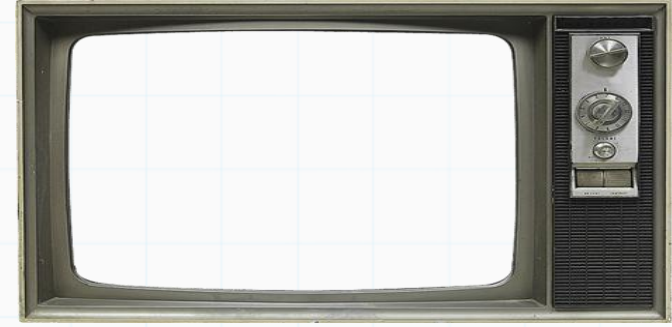
Percorrer:

```
lista *no;  
no =L;  
while (no != NULL)  
{  
    printf("%d, ", no->info);  
    no = no->prox;  
}
```

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



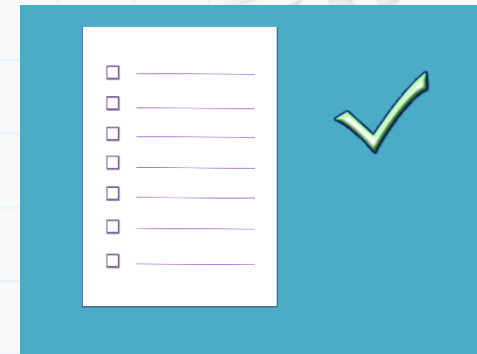
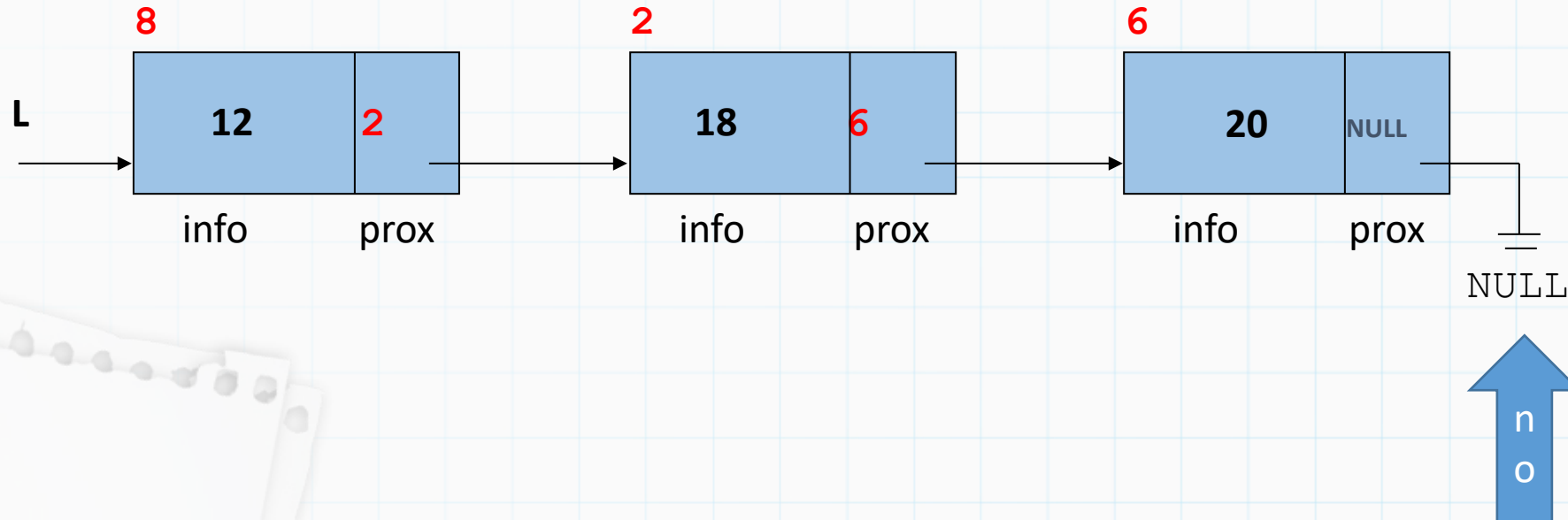
Listas encadenadas



Percorrer:

```
lista *no;  
no =L;  
while (no != NULL)  
{  
    printf("%d, ", no->info);  
    no = no->prox;  
}
```

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



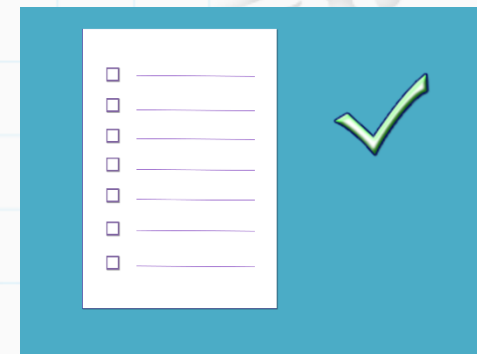
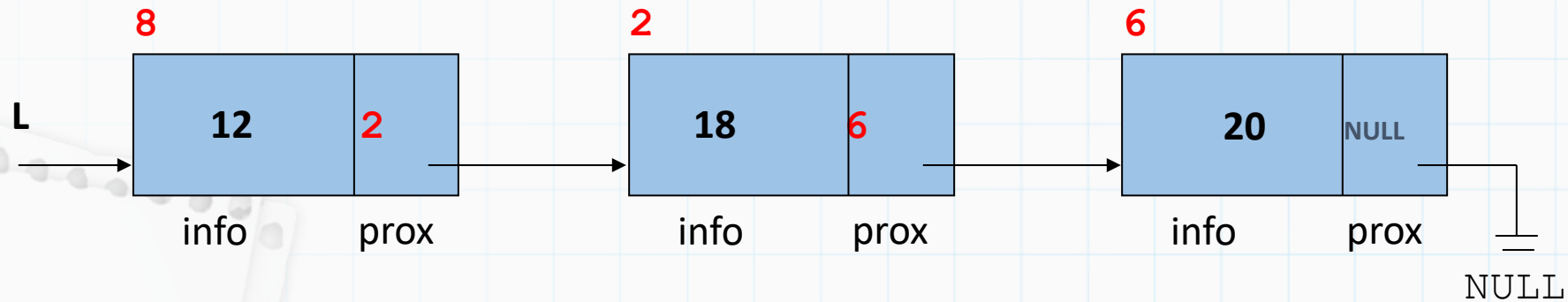
Listas encadeadas

Assim, a função de impressão ficaria assim:

ITERATIVO

```
void imprime_lista(lista* L)
{
    lista* no = L;

    while (no != NULL)
    {
        printf("%d, ", no->info);
        no = no->prox;
    }
}
```



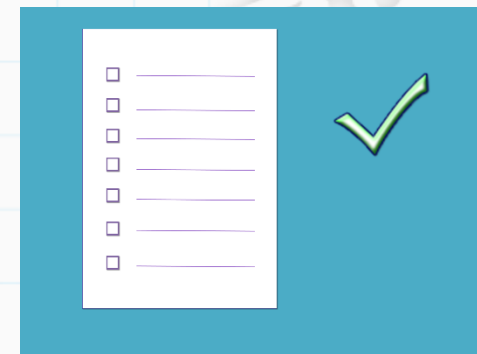
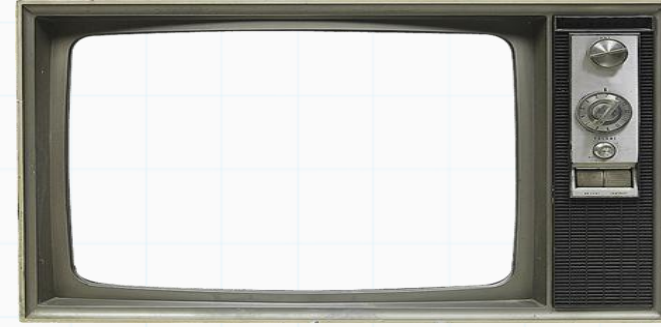
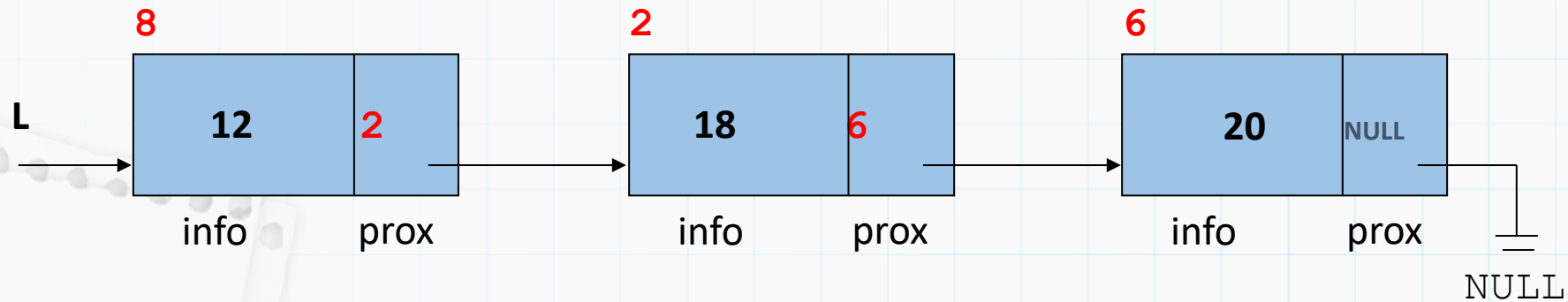
Listas encadeadas

Mas como ficaria a impressão recursiva ?

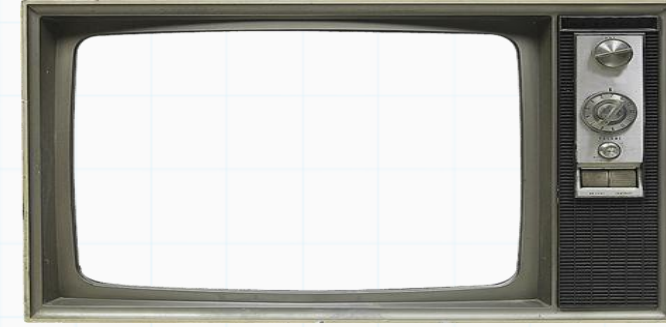
ITERATIVO

```
void imprime_lista(lista* L)
{
    lista* no = L;

    while (no != NULL)
    {
        printf("%d, ", no->info);
        no = no->prox;
    }
}
```



Listas encadeadas



Mas como ficaria a impressão recursiva ?

ITERATIVO

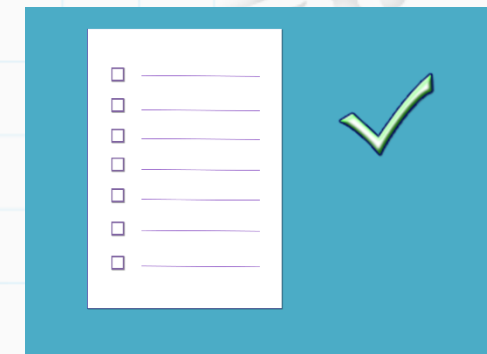
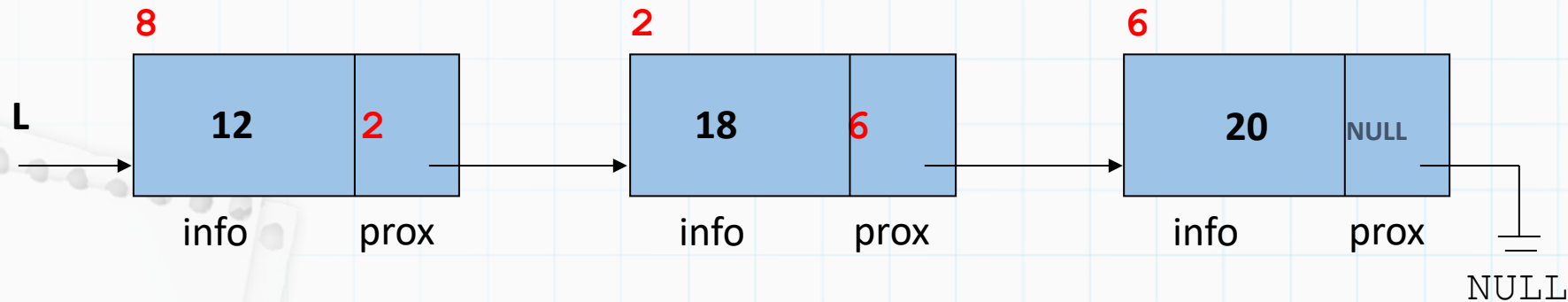
```
void imprime_lista(lista* L)
{
    lista* no = L;

    while (no != NULL)
    {
        printf("%d, ", no->info);
        no = no->prox;
    }
}
```

RECURSIVO

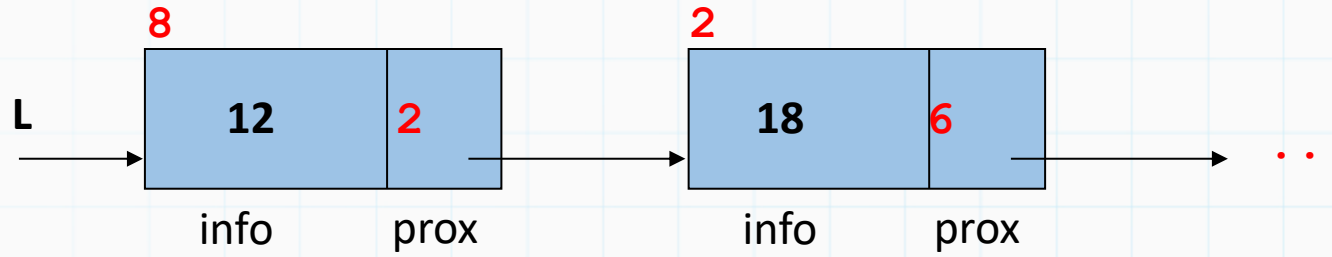
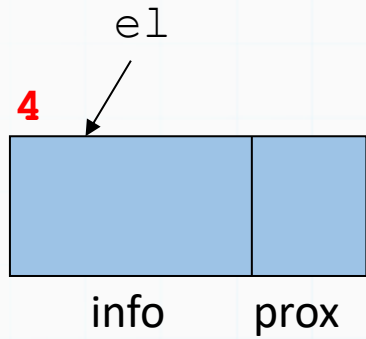
```
void imprime_lista_rec(lista* L)
{
    if (L == NULL)
        return;
    else
    {
        printf("%d, ", L->info);
        imprime_lista_rec(L->prox);
    }
}
```

← Caso base

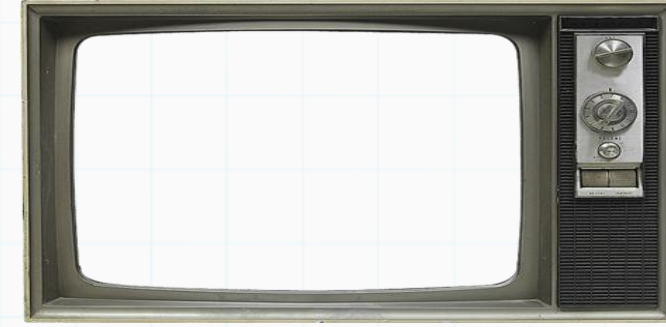


Listas encadeadas

Inserir: A inserção básica é no início da lista L

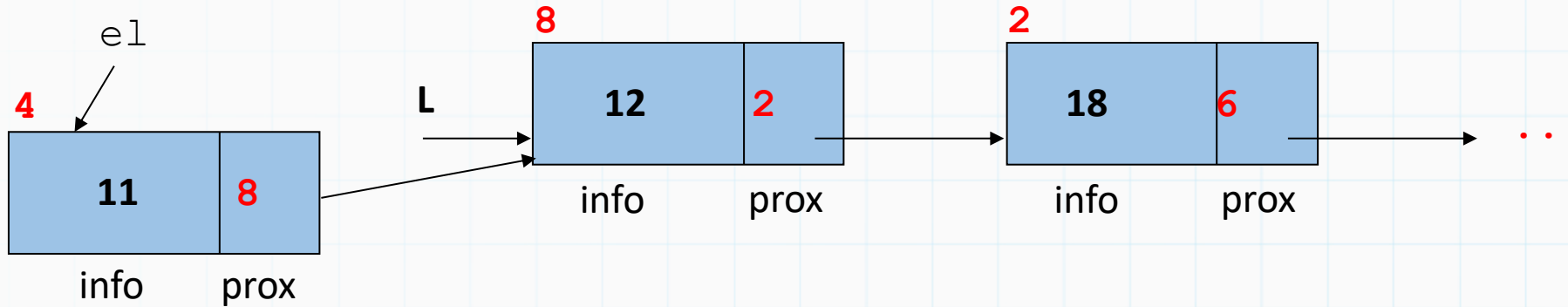


```
e1 = (lista *) malloc(sizeof(lista));
```

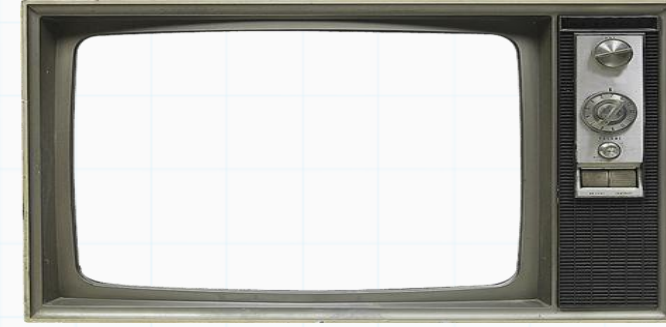


Listas encadeadas

Inserir: A inserção básica é no início da lista L

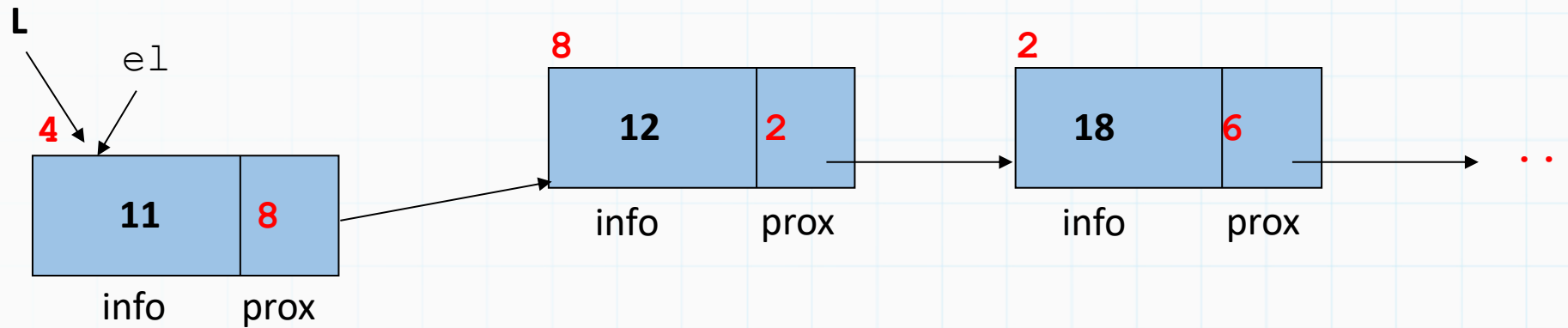


```
el = (lista *) malloc(sizeof(lista));  
el->info = 11;  
el->prox = L;
```



Listas encadeadas

Inserir: A inserção básica é no início da lista L

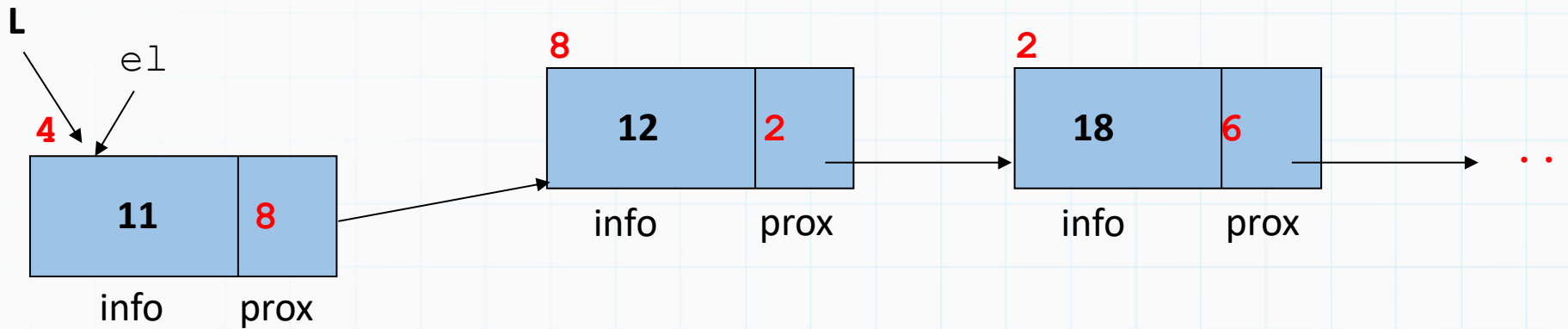


```
el = (lista *) malloc(sizeof(lista));  
el->info = 11;  
el->prox = L;  
L = el;
```



Listas encadeadas

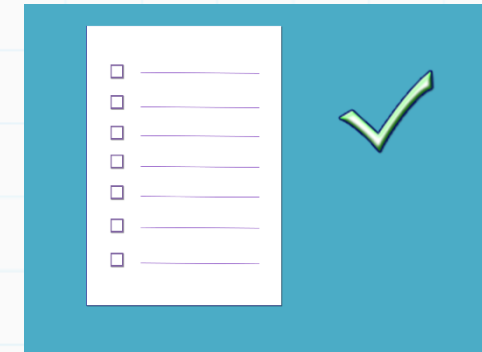
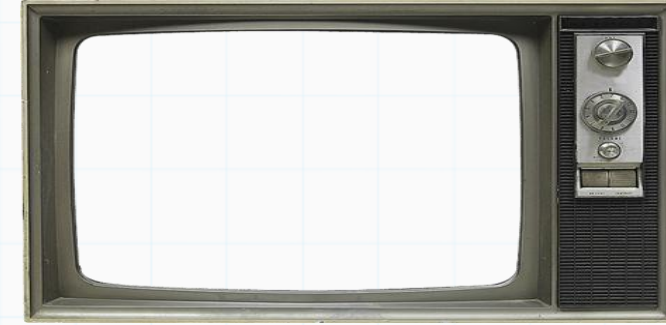
Inserir: A inserção básica é no início da lista L



```
el = (lista *) malloc(sizeof(lista));  
el->info = 11;  
el->prox = L;  
L = el;
```

Se a inserção for feita dentro de uma função, não esqueça de sempre retornar o início da lista para não perder o identificador da lista.

```
L = insere_lista(L, 11);
```



Listas encadeadas

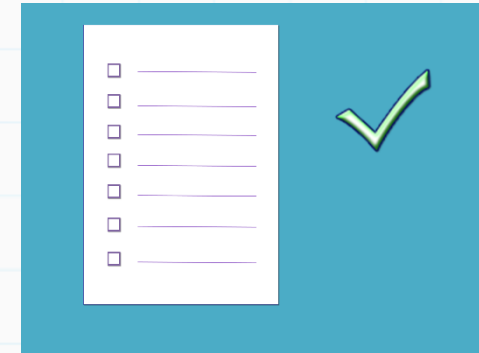
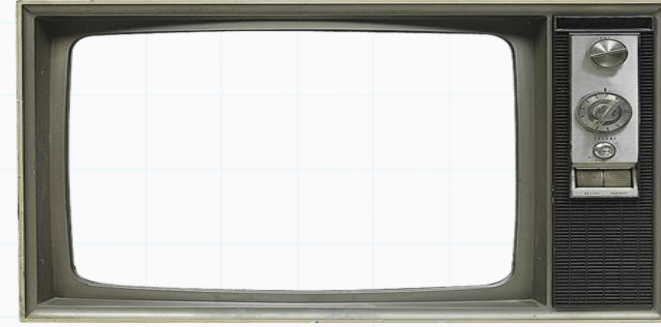
Inserir: A inserção básica é no início da lista L

```
lista * aloca_no(void)
{
    lista *aux;
    aux      = (lista *) malloc (sizeof(lista));
    aux->prox = NULL;

    return aux;
}
```

```
lista * insere_lista(lista* L, int el)
{
    lista *no;
    no      = aloca_no();
    no->info = el;
    no->prox = L;
    L        = no;

    return L;
}
```



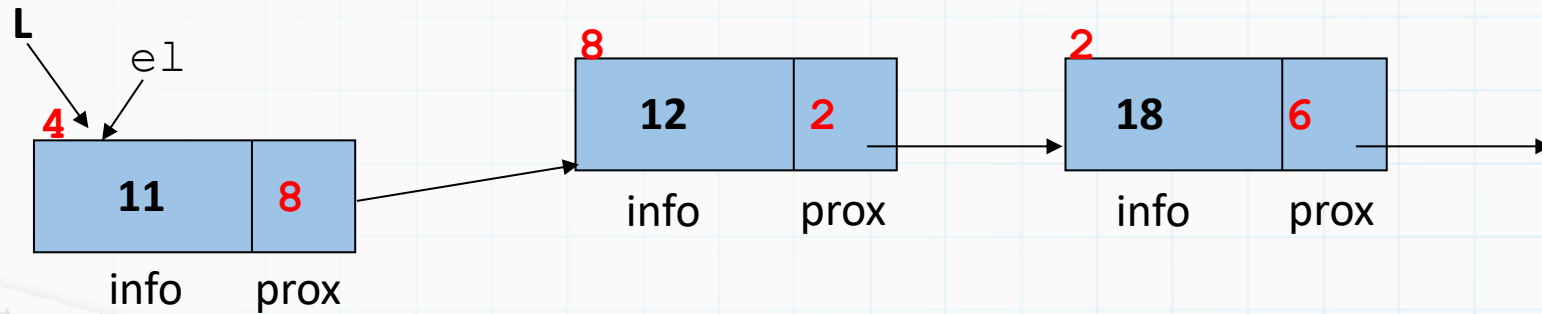
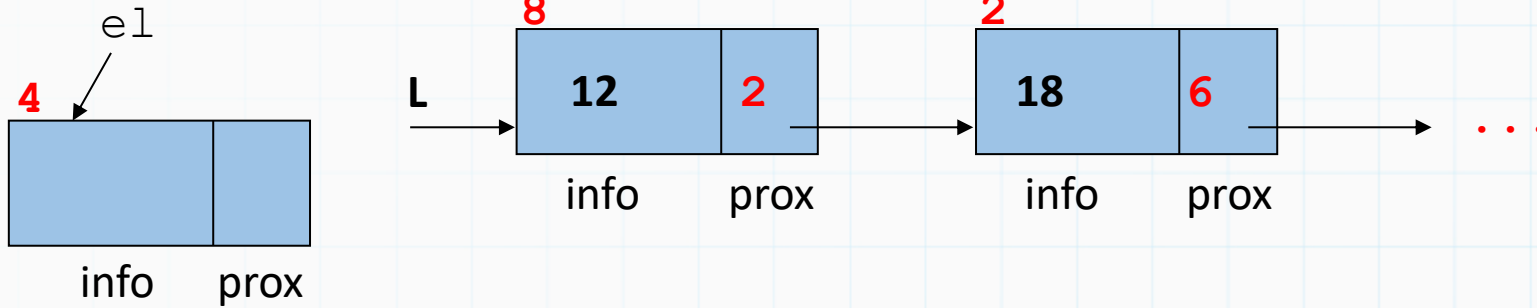
Se a inserção for feita dentro de uma função, não esqueça de sempre retornar o início da lista para não perder o identificador da lista.

```
L = insere_lista(L, 11);
```

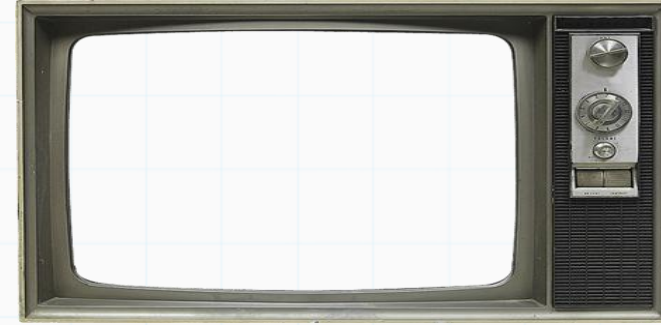
Listas encadeadas

Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

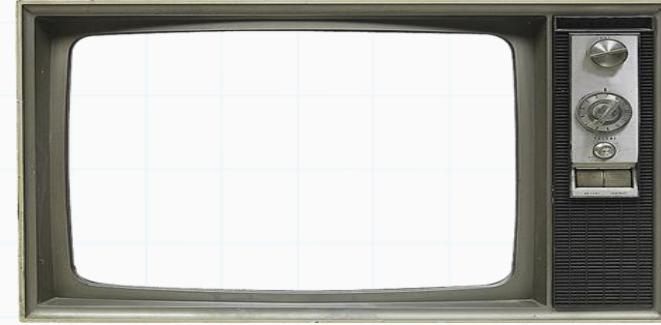
$pos=0$ (igual inserção no início)



```
... if (pos == 0)
    {
        lista *el = aloca_no();
        el->info = el;
        el->prox = L;
        return el;
    }
```

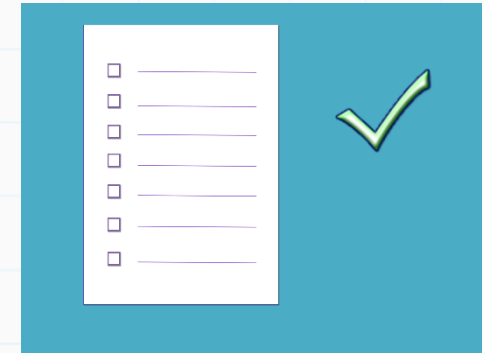
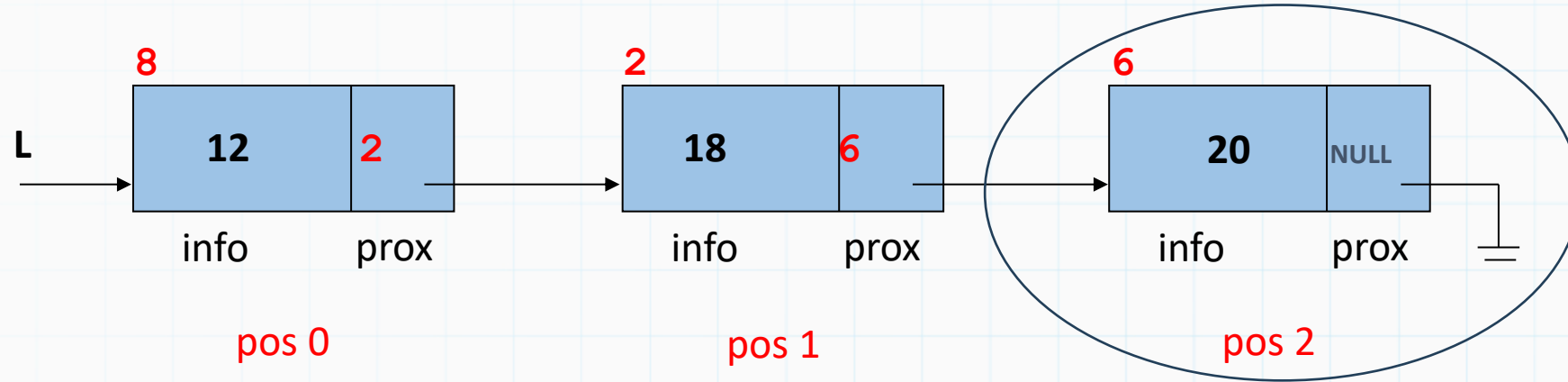


Listas encadeadas

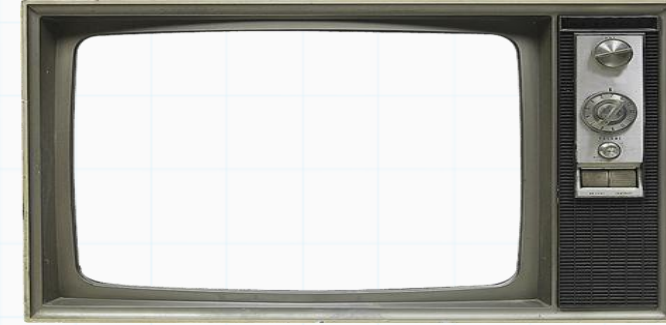


Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

$pos > 0$ (encontra o anterior a posição) Ex: Queremos inserir na $pos = 2$ elemento 15

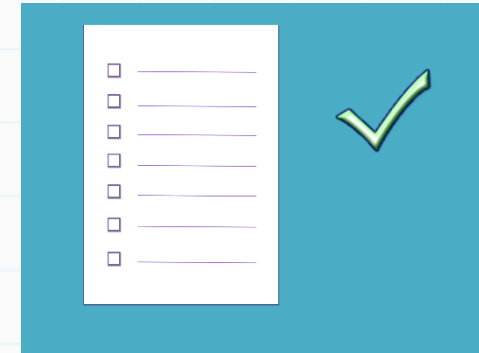
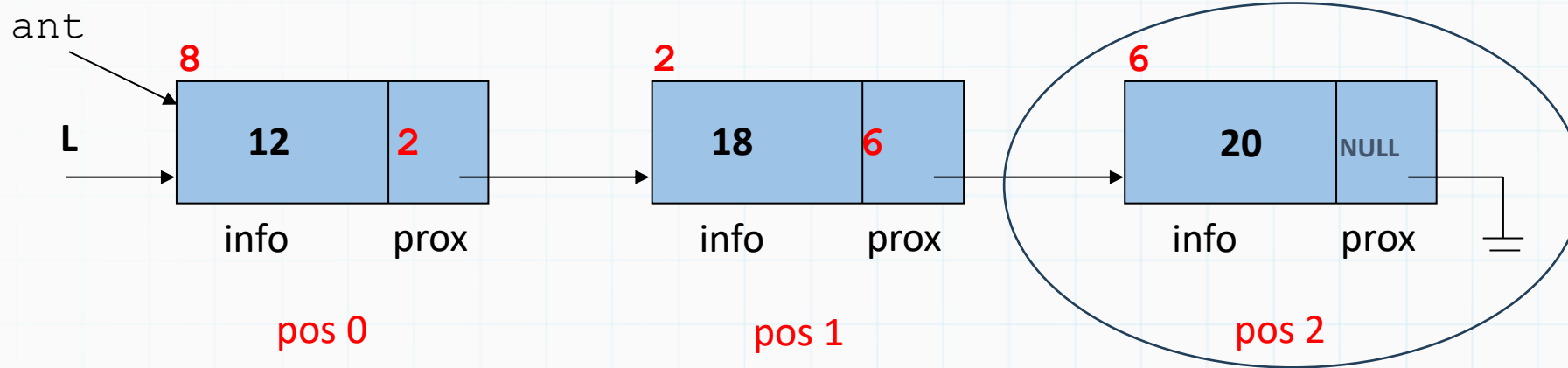


Listas encadeadas



Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

$pos > 0$ (encontra o anterior a posição) Ex: Queremos inserir na $pos = 2$ elemento 15



```
lista * anterior = L;
```

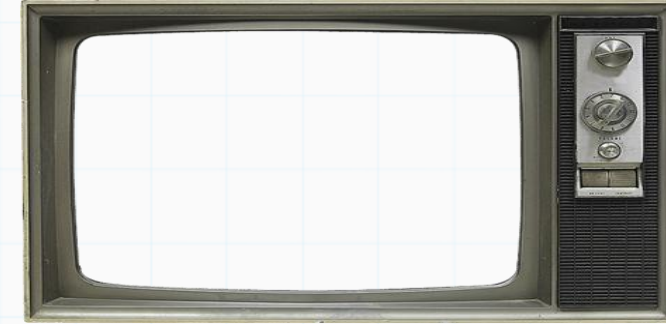
```
// encontra anterior do elemento (se existir)  
for(int i=0; i<pos-1; i++)  
{  
    anterior=anterior->prox;  
}
```

OU

```
lista * anterior = L;
```

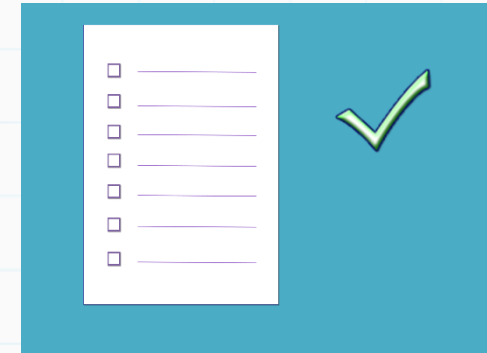
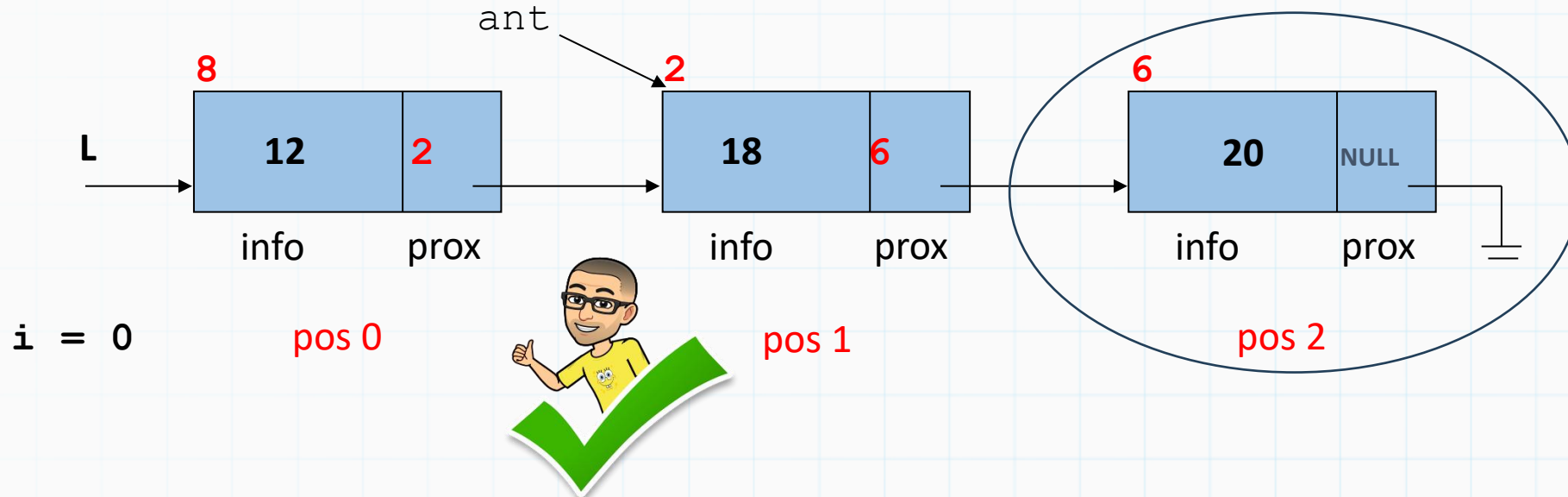
```
// encontra anterior do elemento (se existir)  
for(int i=1; i<=pos-1; i++)  
{  
    anterior=anterior->prox;  
}
```

Listas encadeadas



Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

$pos > 0$ (encontra o anterior a posição) Ex: Queremos inserir na $pos = 2$ elemento 15

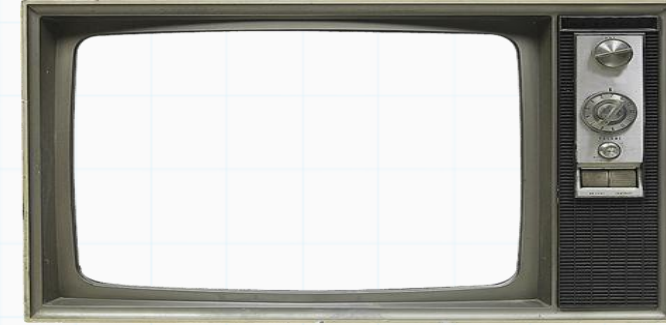


```
lista * anterior = L;  
  
// encontra anterior do elemento (se existir)  
for(int i=0; i<pos-1; i++)  
{  
    anterior=anterior->prox;  
}
```

OU

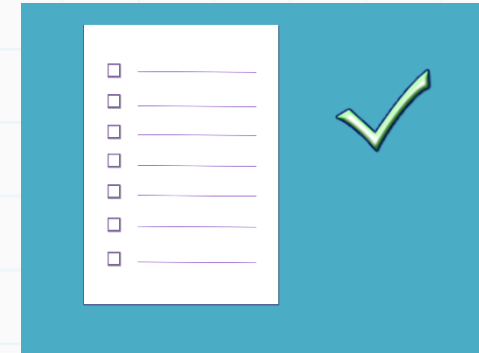
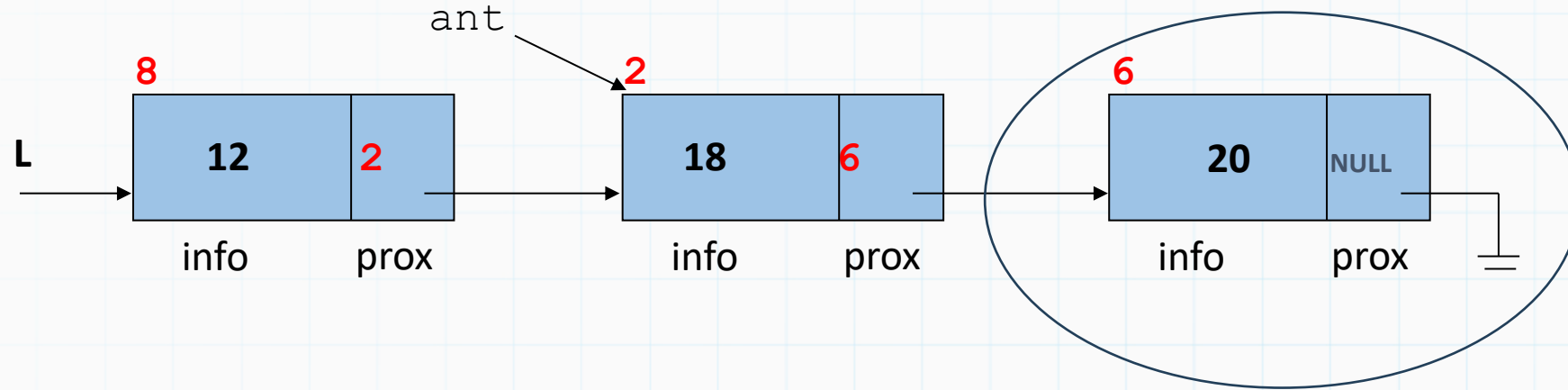
```
lista * anterior = L;  
  
// encontra anterior do elemento (se existir)  
for(int i=1; i<=pos-1; i++)  
{  
    anterior=anterior->prox;  
}
```

Listas encadeadas

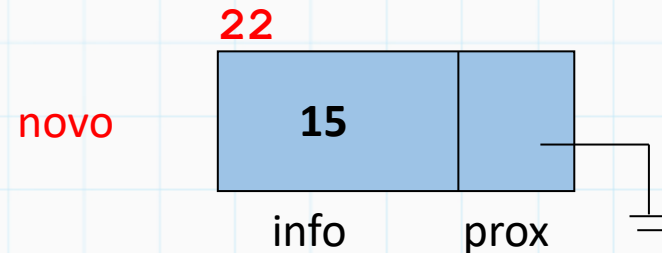


Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

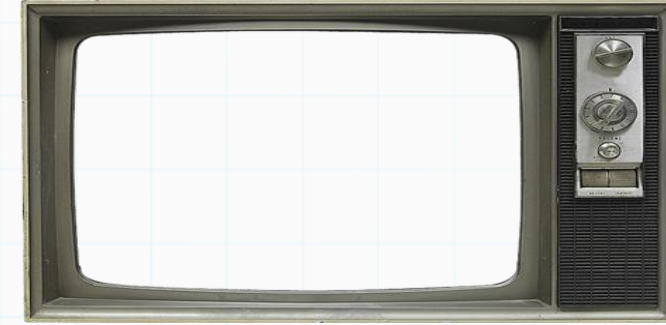
$pos > 0$ (encontra o anterior a posição) Ex: Queremos inserir na $pos = 2$ elemento 15



```
lista *novo = aloca_no();  
novo->info = el;
```

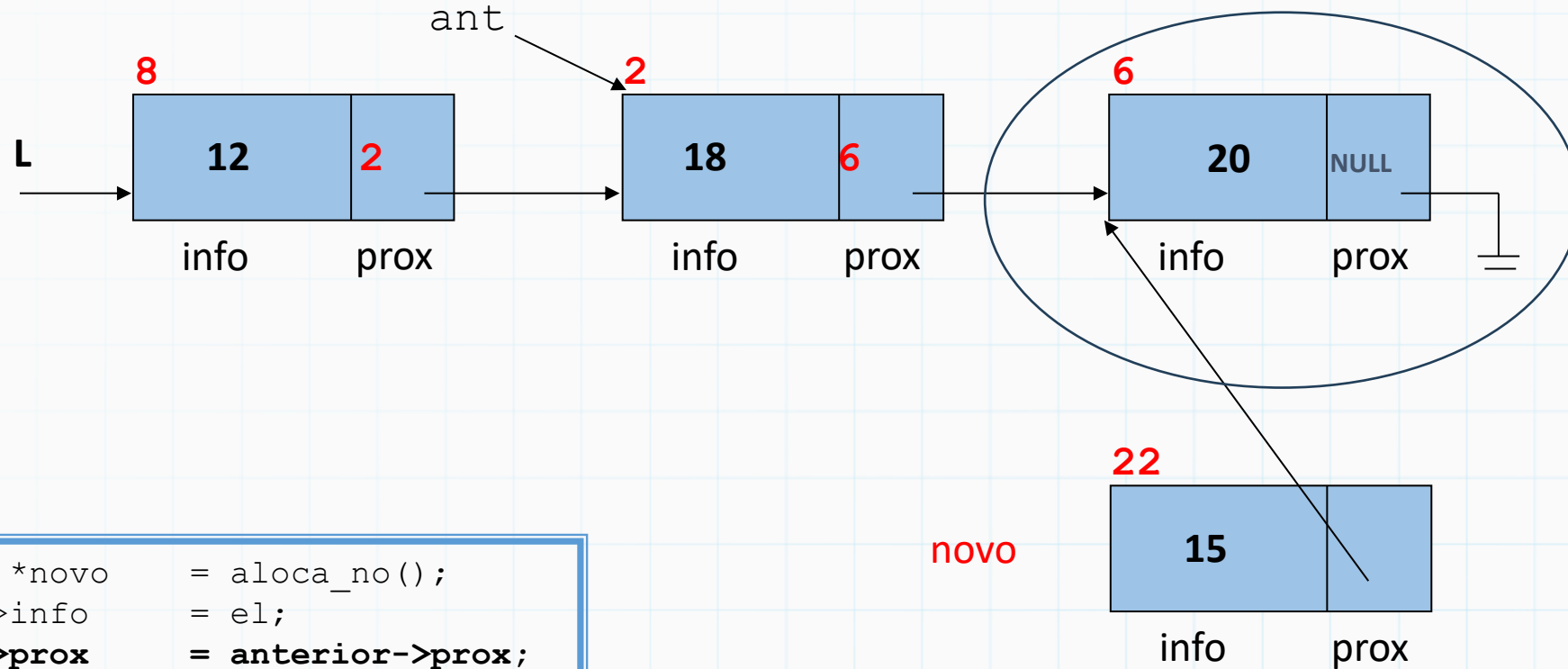


Listas encadeadas



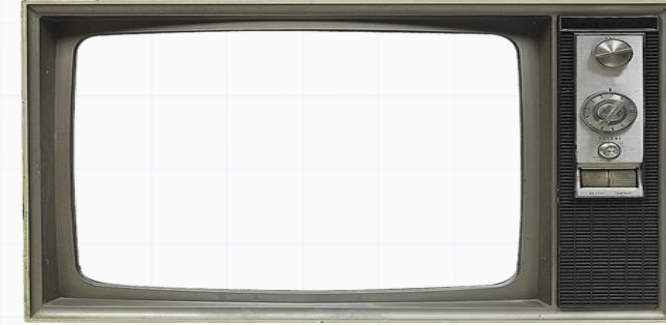
Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

$pos > 0$ (encontra o anterior a posição) Ex: Queremos inserir na $pos = 2$ elemento 15



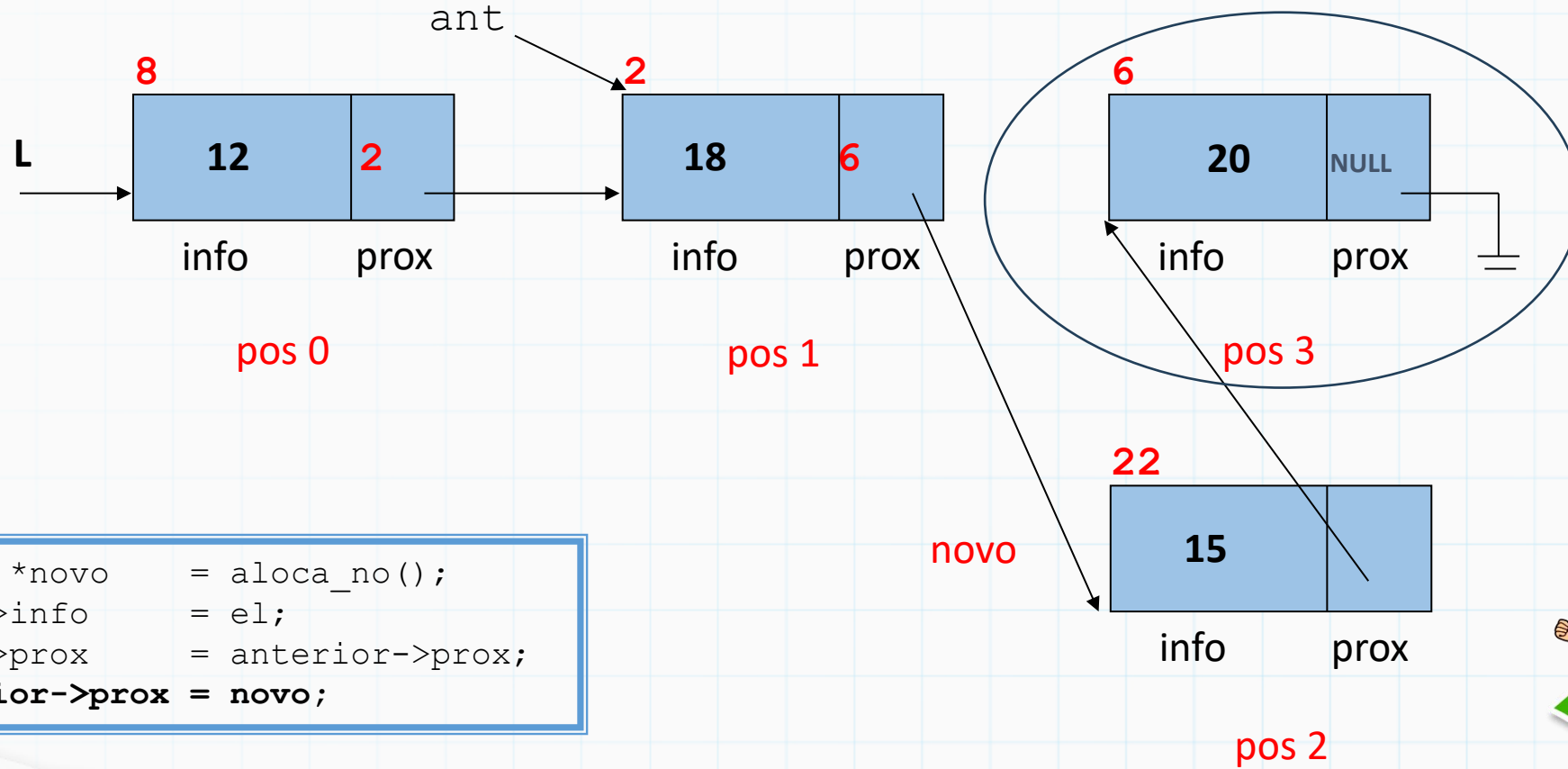
```
lista *novo = aloca_no();  
novo->info = el;  
novo->prox = anterior->prox;
```

Listas encadeadas

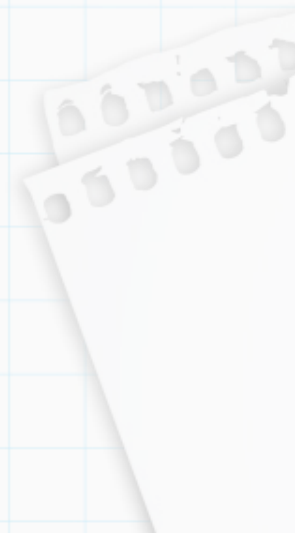
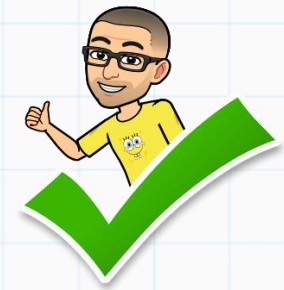


Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

$pos > 0$ (encontra o anterior a posição) Ex: Queremos inserir na $pos = 2$ elemento 15



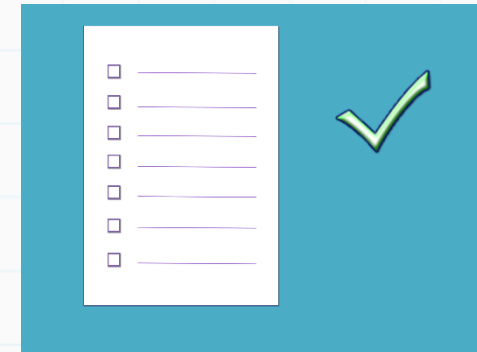
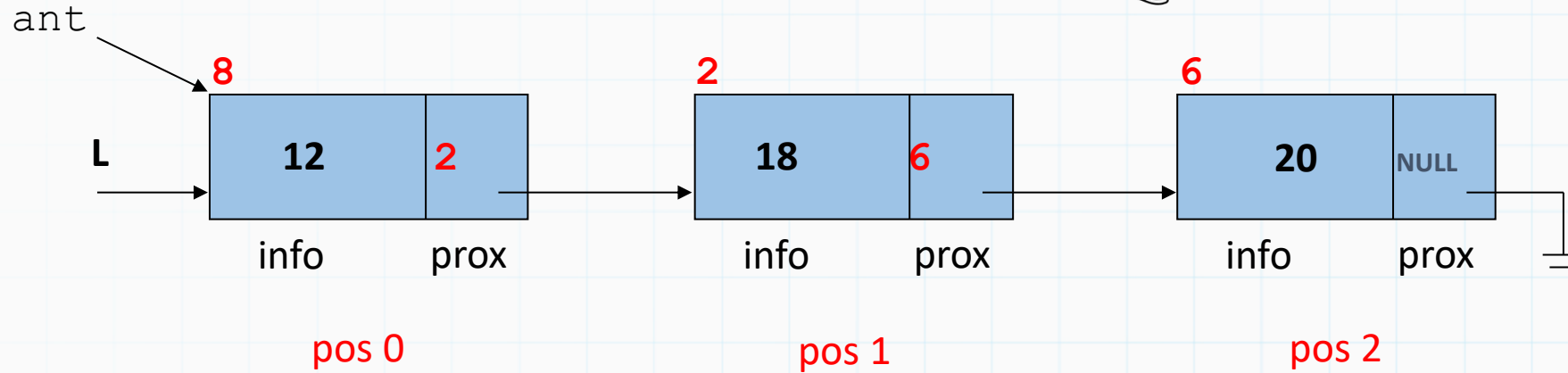
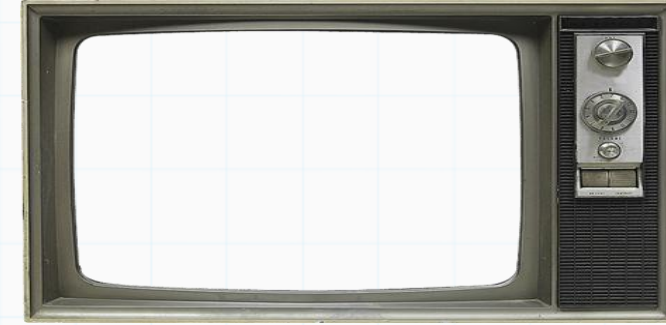
```
lista *novo = aloca_no();
novo->info = el;
novo->prox = anterior->prox;
anterior->prox = novo;
```



Listas encadeadas

Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

e se pos fosse um elemento novo no fim da lista, isto é, $pos=3$



```
lista * anterior = L;
```

```
// encontra anterior do elemento (se existir)  
for(int i=0; i<pos-1; i++)  
{  
    anterior=anterior->prox;  
}
```

OU

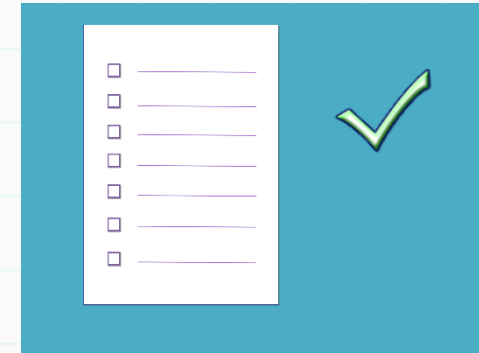
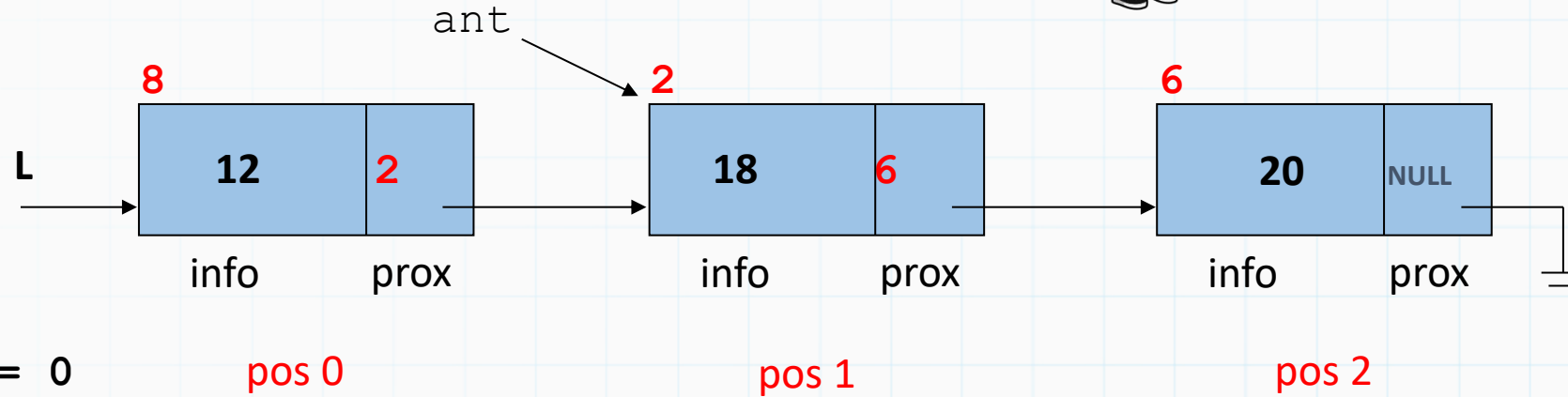
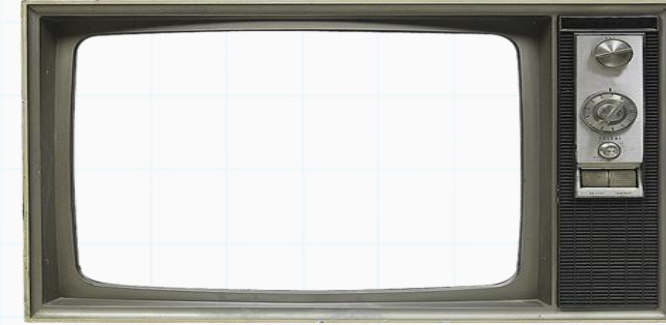
```
lista * anterior = L;
```

```
// encontra anterior do elemento (se existir)  
for(int i=1; i<=pos-1; i++)  
{  
    anterior=anterior->prox;  
}
```

Listas encadeadas

Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

e se pos fosse um elemento novo no fim da lista, isto é, $pos=3$



```
lista * anterior = L;

// encontra anterior do elemento (se existir)
for(int i=0; i<pos-1; i++)
{
    anterior=anterior->prox;
}
```

OU

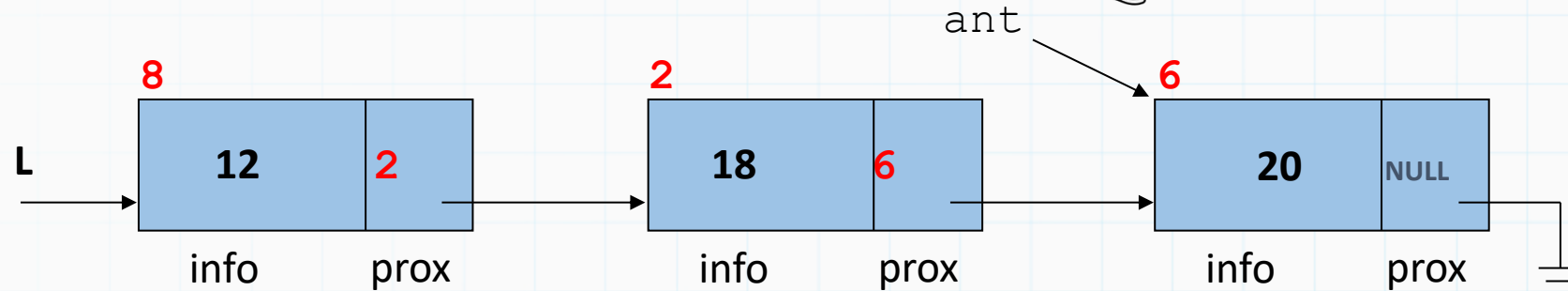
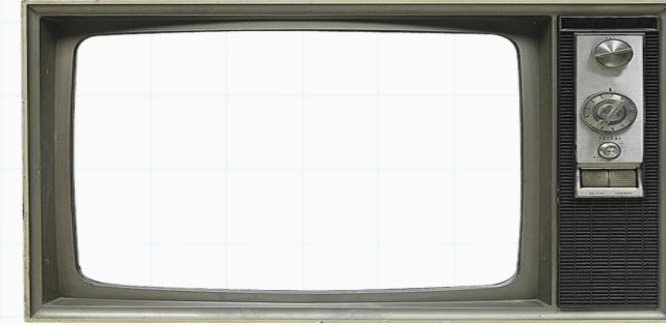
```
lista * anterior = L;

// encontra anterior do elemento (se existir)
for(int i=1; i<=pos-1; i++)
{
    anterior=anterior->prox;
}
```

Listas encadeadas

Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

e se pos fosse um elemento novo no fim da lista, isto é, $pos=3$



$i = 1$

pos 0



pos 1

pos 2

```
lista * anterior = L;  
  
// encontra anterior do elemento (se existir)  
for(int i=0; i<pos-1; i++)  
{  
    anterior=anterior->prox;  
}
```

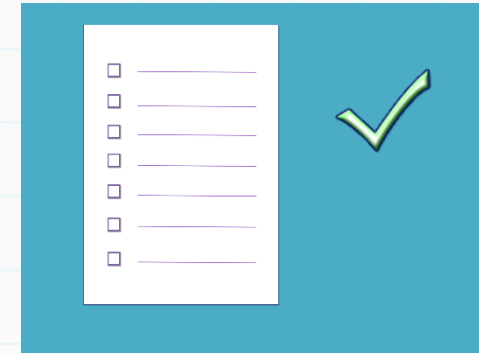
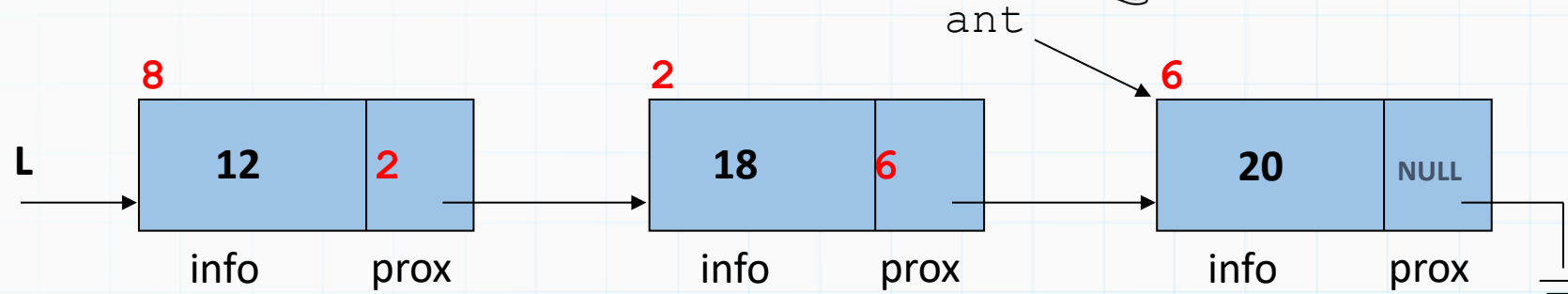
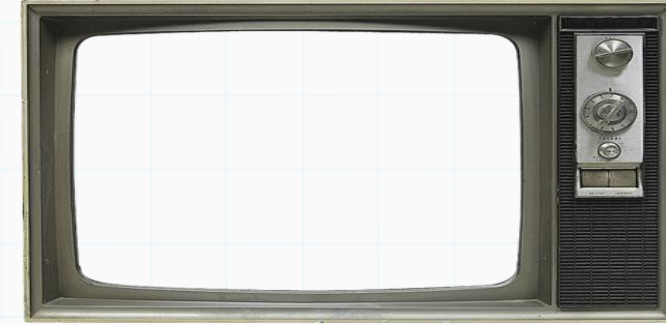
OU

```
lista * anterior = L;  
  
// encontra anterior do elemento (se existir)  
for(int i=1; i<=pos-1; i++)  
{  
    anterior=anterior->prox;  
}
```

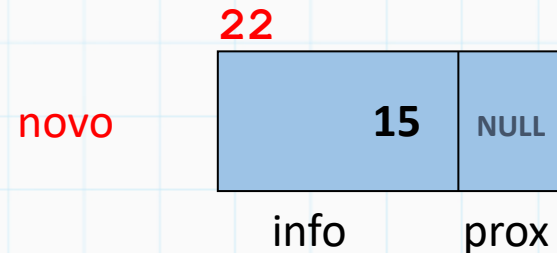
Listas encadeadas

Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

e se pos fosse um elemento novo no fim da lista, isto é, $pos=3$



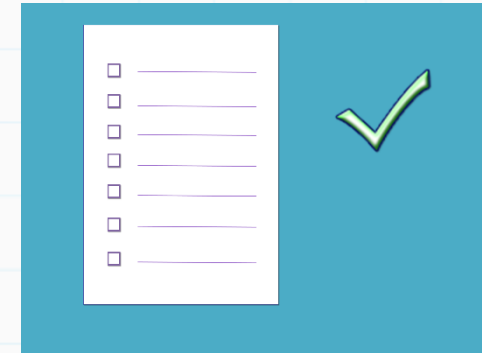
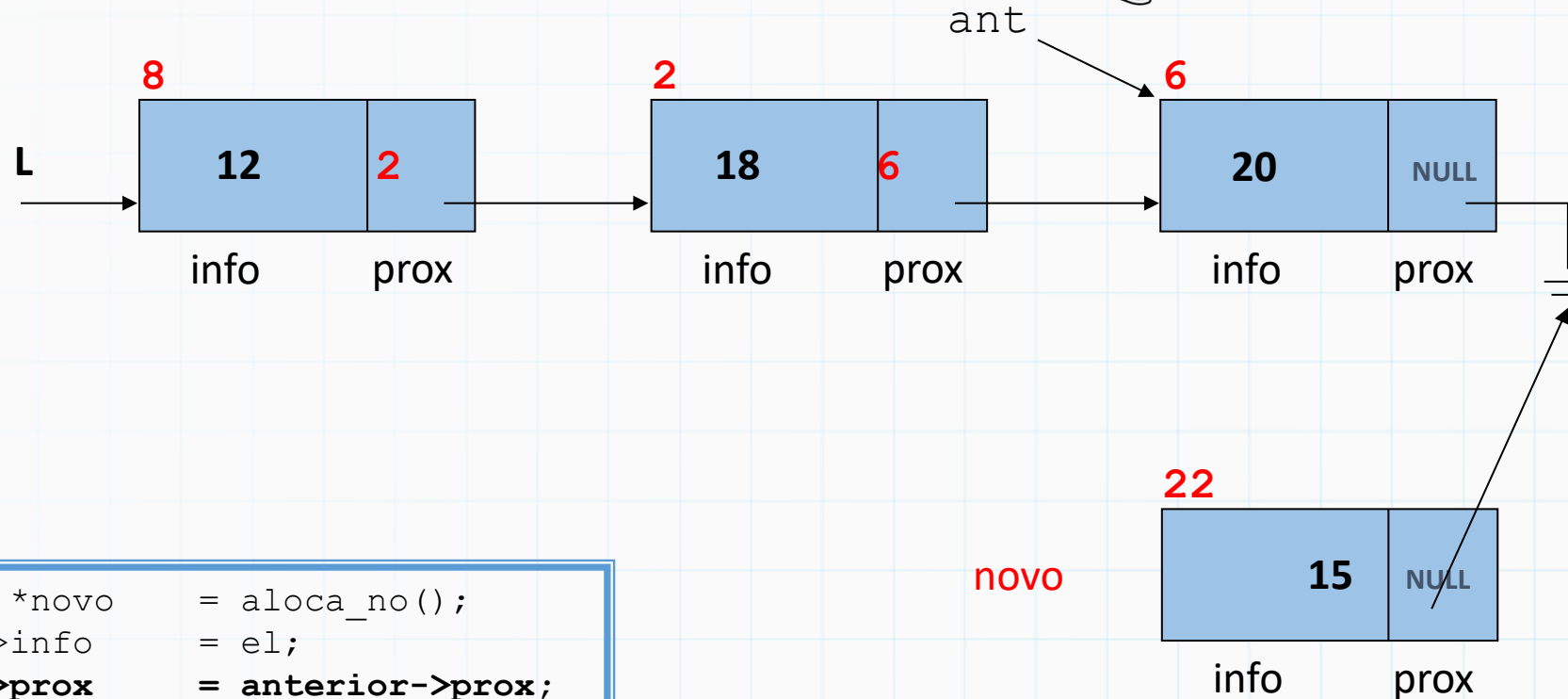
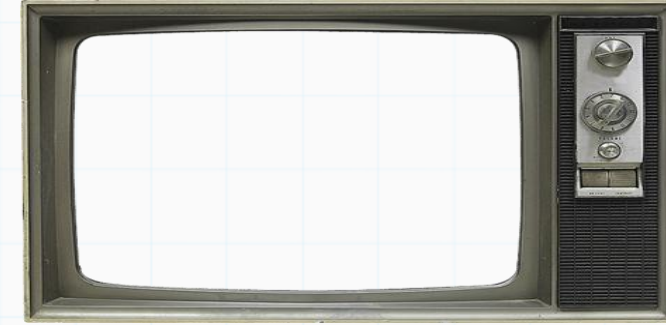
```
lista *novo = aloca_no();  
novo->info = el;  
novo->prox = anterior->prox;  
anterior->prox = novo;  
return L;
```



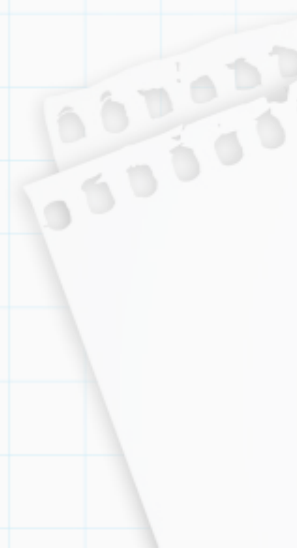
Listas encadeadas

Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

e se pos fosse um elemento novo no fim da lista, isto é, $pos=3$



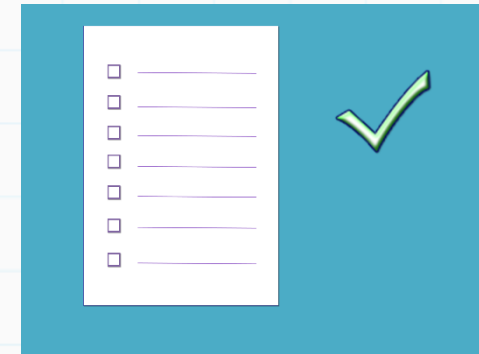
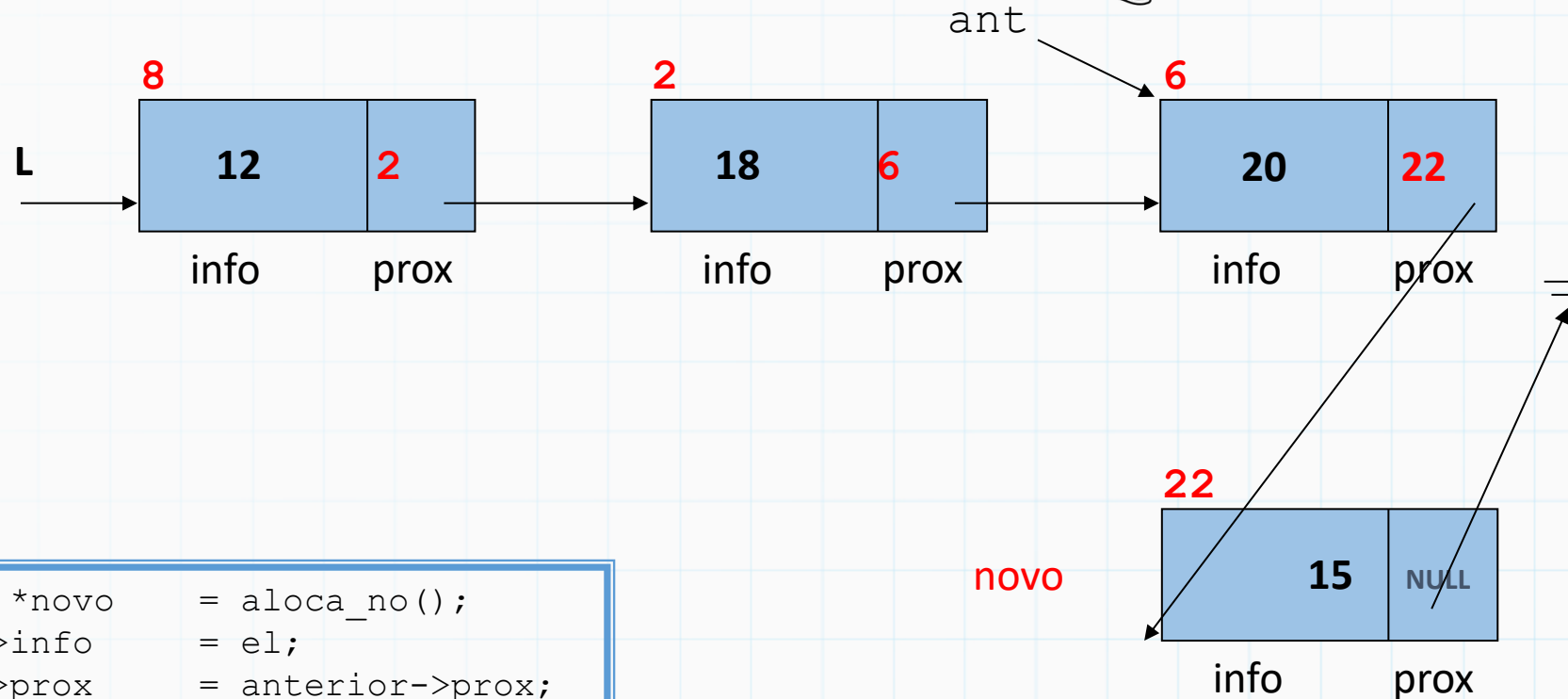
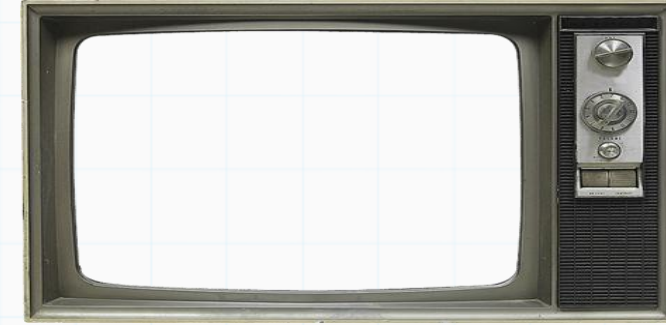
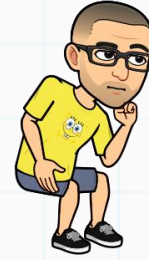
```
lista *novo = aloca_no();
novo->info = el;
novo->prox = anterior->prox;
anterior->prox = novo;
return L;
```



Listas encadeadas

Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

e se pos fosse um elemento novo no fim da lista, isto é, $pos=3$



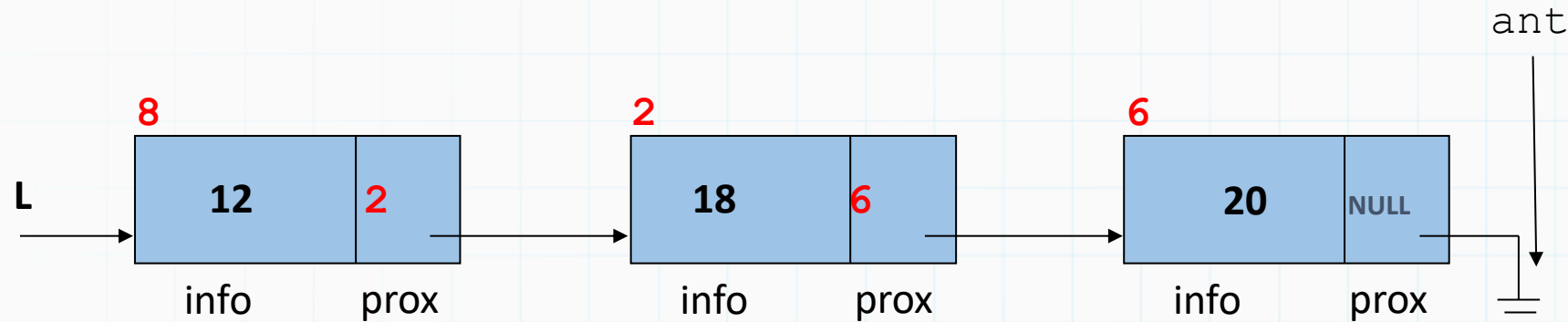
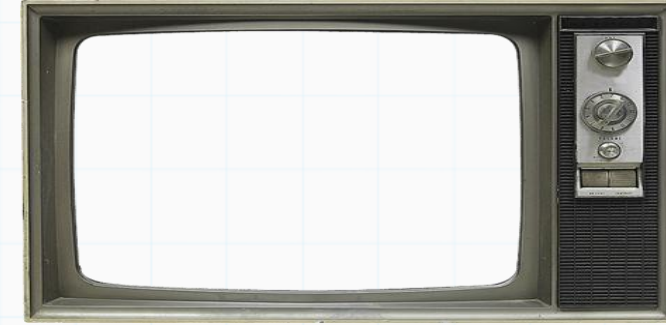
```
lista *novo = aloca_no();
novo->info = el;
novo->prox = anterior->prox;
anterior->prox = novo;
return L;
```



Listas encadeadas

Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

e se pos fosse um valor maior que n , isto é, $pos=4$

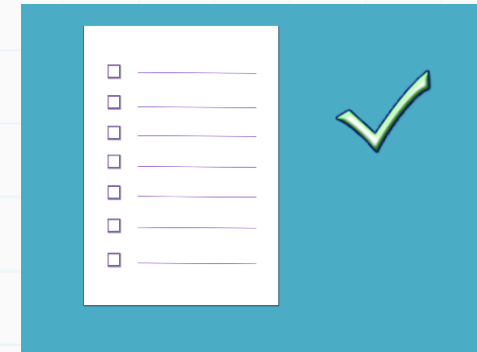


$i = 2$

pos 0

pos 1

pos 2



```
lista * anterior = L;  
  
// encontra anterior do elemento (se existir)  
for(int i=0; i<pos-1; i++)  
{  
    anterior=anterior->prox;  
}
```

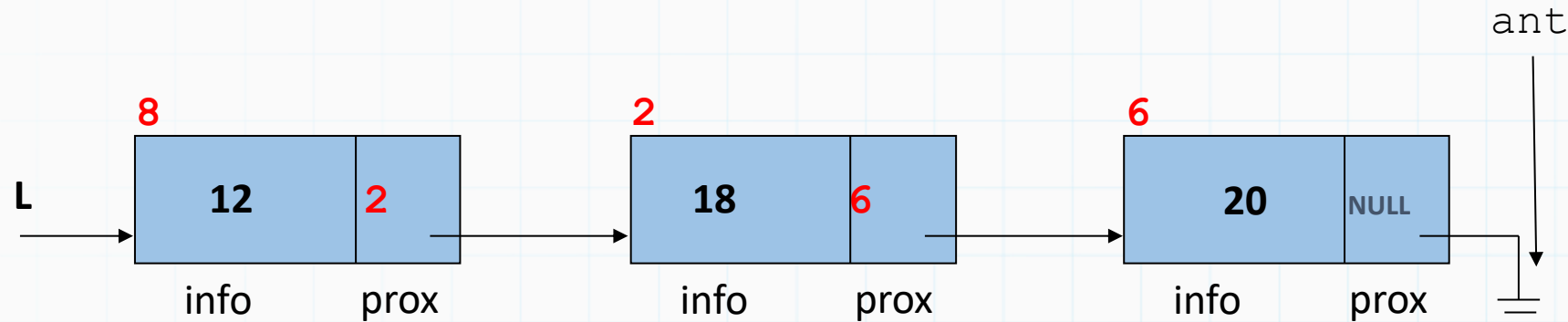
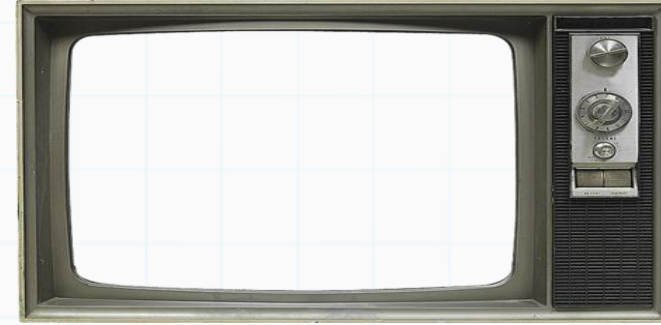
OU

```
lista * anterior = L;  
  
// encontra anterior do elemento (se existir)  
for(int i=1; i<=pos-1; i++)  
{  
    anterior=anterior->prox;  
}
```

Listas encadeadas

Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

e se pos fosse um valor maior que n , isto é, $pos=4$

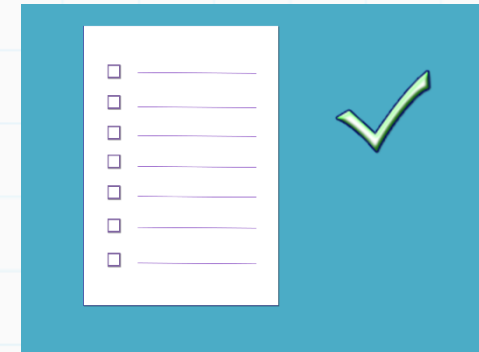


$i = 2$

pos 0

pos 1

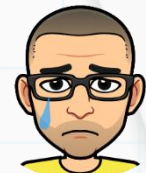
pos 2



```
lista * anterior = L;  
  
// encontra anterior do elemento (se existir)  
for(int i=0; i<pos-1; i++)  
{  
    anterior=anterior->prox;  
}
```

```
lista *novo      = aloca_no();  
novo->info       = el;  
novo->prox       = anterior->prox;  
anterior->prox  = novo;  
return L;
```

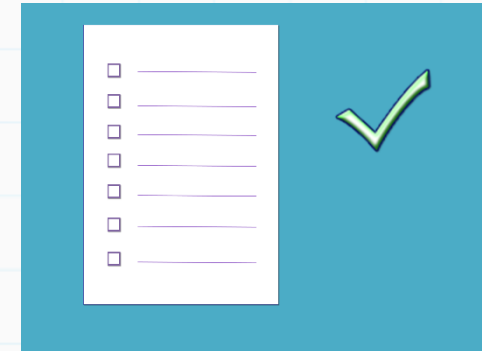
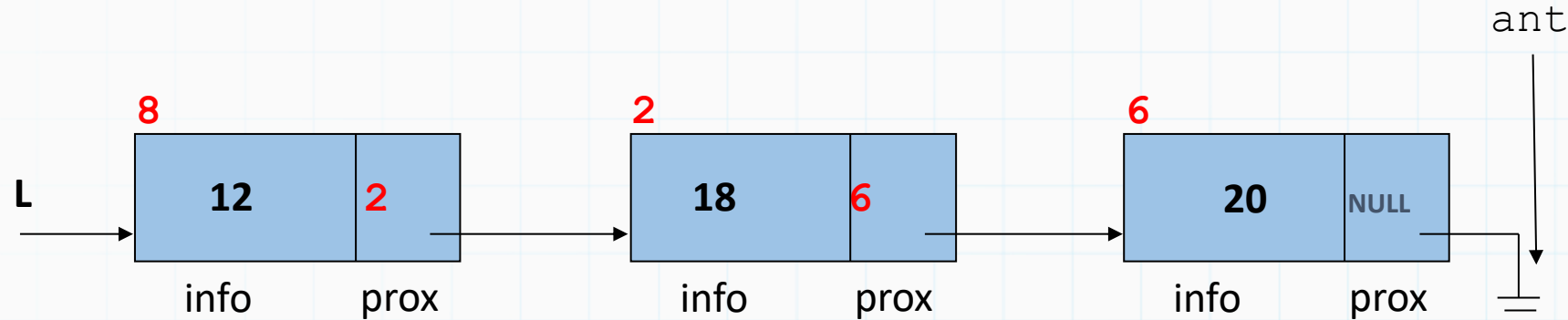
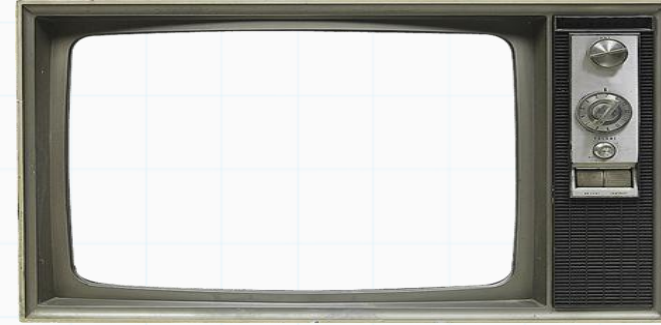
ERRO !



Listas encadeadas

Inserir: A inserção numa posição específica $pos=\{0, 1, 2, \dots, n\}$

e se pos fosse um valor maior que n , isto é, $pos=4$



$i = 2$

pos 0

pos 1

pos 2

```
lista * anterior = L;

// encontra anterior do elemento (se existir)
for(int i=0; i<pos-1; i++)
{
    anterior=anterior->prox;
}
```



```
lista * anterior = L;

// encontra anterior do elemento (se existir)
for(int i=0; i<pos-1; i++)
{
    anterior=anterior->prox;

    // passou do fim
    if (anterior == NULL)
    {
        printf("Posicao invalida\n");
        return L;
    }
}
```

Listas encadeadas

```
lista* insere_lista_pos(lista* L, int el, int pos)
{
    if(pos < 0)
    {
        printf("Posicao invalida\n");
        return L;
    }

    // insercao no inicio
    if(pos == 0)
    {
        lista *novo = aloca_no();
        novo->info = el;
        novo->prox = L;
        return novo;
    }
}
```

```
    // demais posicoes
    else
    {
        lista * anterior = L;

        // encontra anterior do elemento (se existir)
        for(int i=0; i<pos-1; i++)
        {
            anterior=anterior->prox;

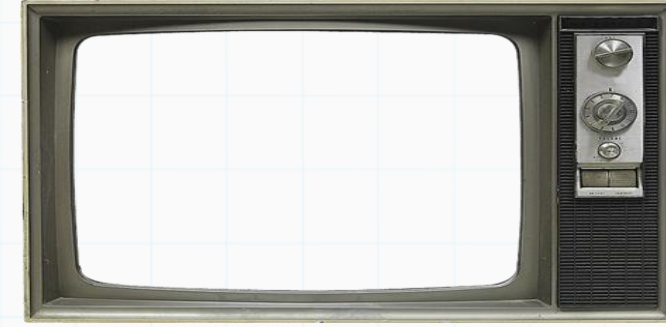
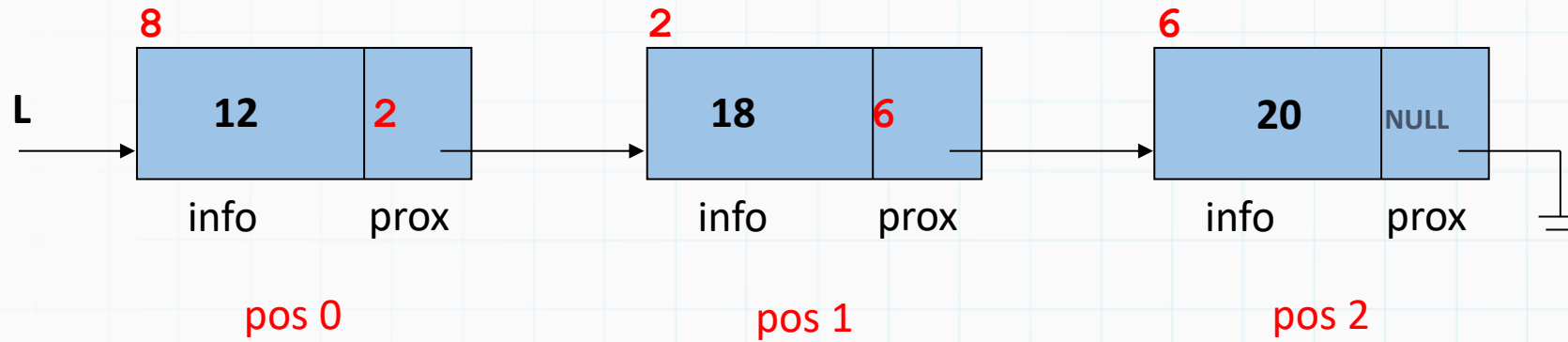
            // passou do fim
            if (anterior == NULL)
            {
                printf("Posicao invalida\n");
                return L;
            }
        }

        lista *novo = aloca_no();
        novo->info = el;
        novo->prox = anterior->prox;
        anterior->prox = novo;
        return L;
    }
}
```

Listas encadeadas

Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

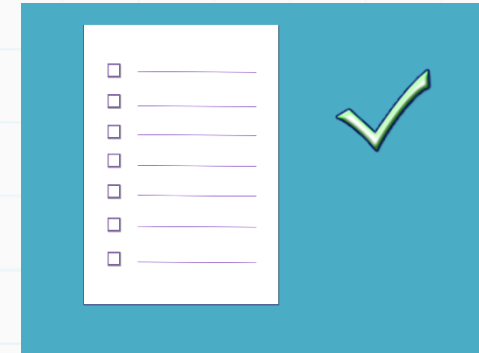
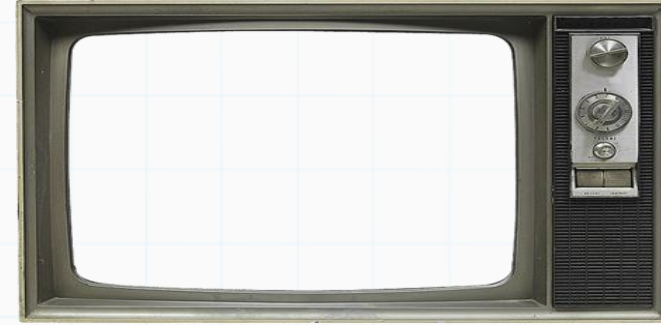
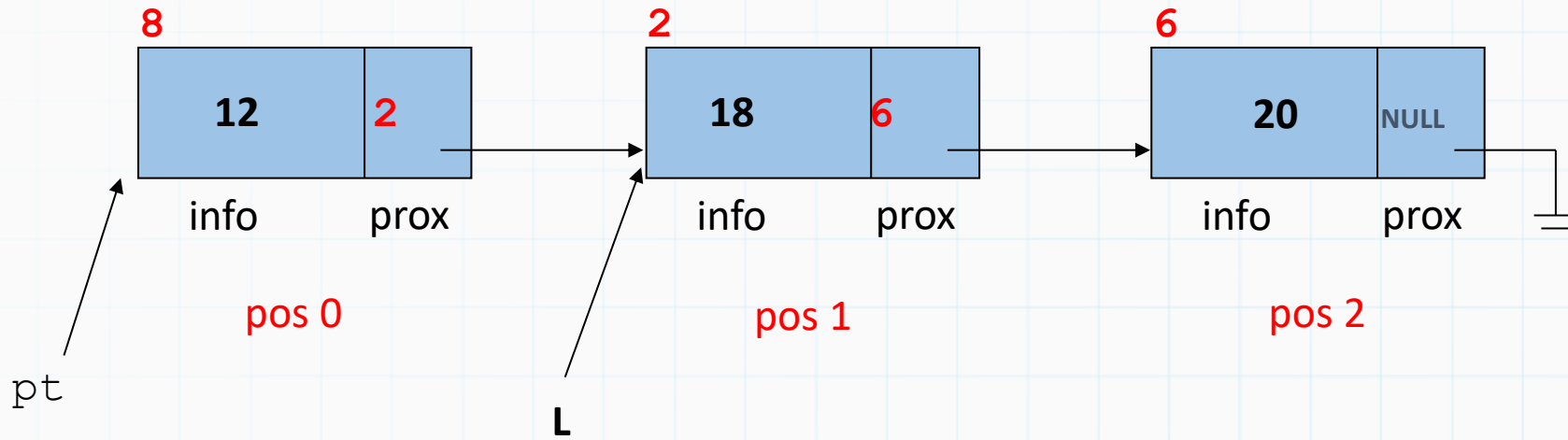
pos=0



Listas encadeadas

Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos=0$



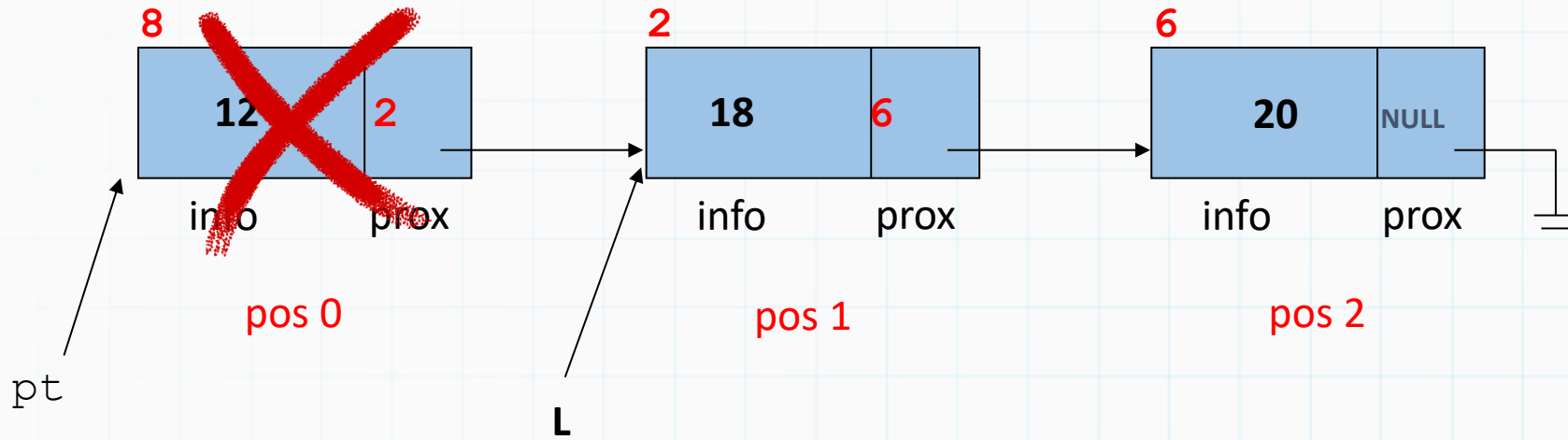
```
if(pos == 0)
{
    pt    = L;
    L     = L->prox;
}
```



Listas encadeadas

Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

pos=0



```
if(pos == 0)
{
    pt    = L;
    L    = L->prox;
    free(pt);
    return L;
}
```

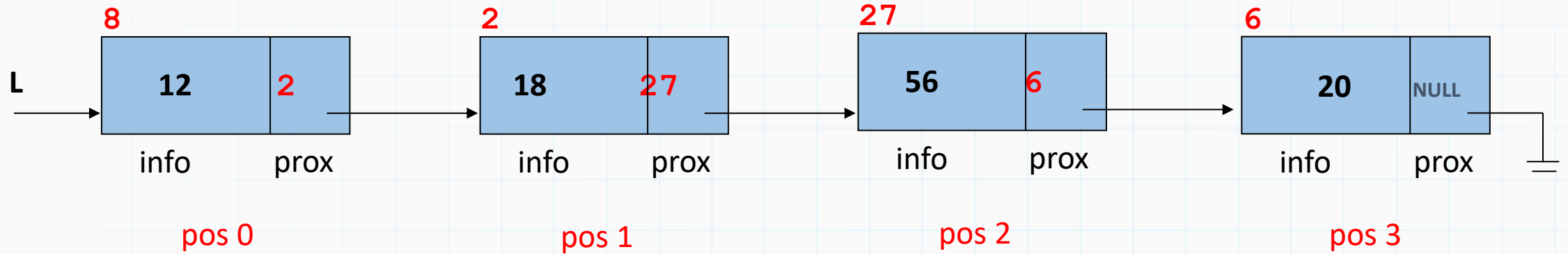


Listas encadeadas

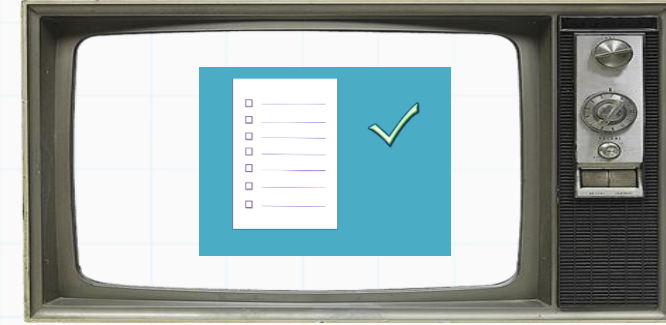


Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos > 0$, (encontra a posição e o anterior a posição) Ex: Queremos remover na $pos = 2$

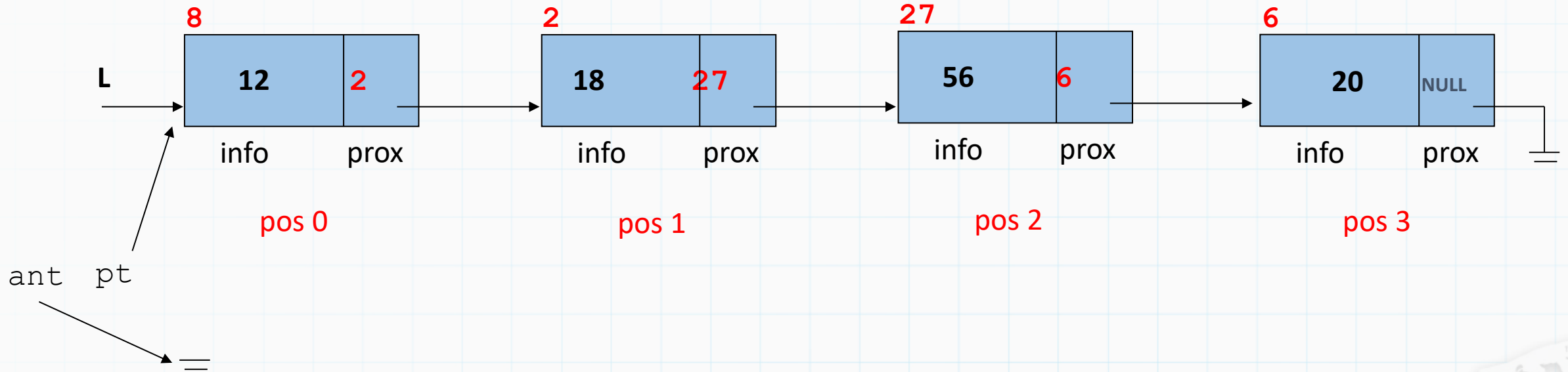


Listas encadeadas



Remover: Remover numa posição específica $pos = \{0, 1, 2, \dots, n-1\}$

$pos > 0$, (encontra a posição e o anterior a posição) Ex: Queremos remover na $pos = 2$



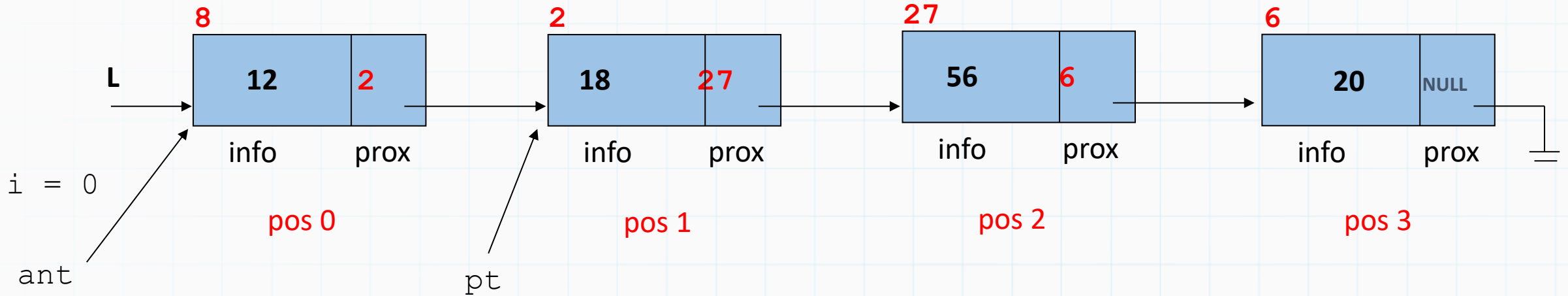
```
pt = L; anterior = NULL;
```

Listas encadeadas



Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos > 0$, (encontra a posição e o anterior a posição) Ex: Queremos remover na $pos = 2$



```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt      = pt->prox;
}
```

OU

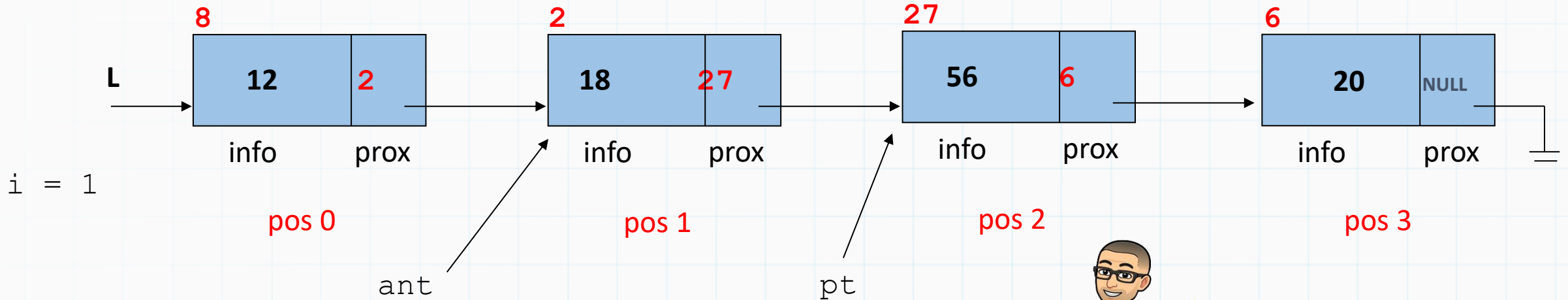
```
pt = L; anterior = NULL;
for(int i=1; i<=pos; i++)
{
    anterior = pt;
    pt      = pt->prox;
}
```

Listas encadeadas



Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos > 0$, (encontra a posição e o anterior a posição) Ex: Queremos remover na $pos = 2$



```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt      = pt->prox;
}
```

OU

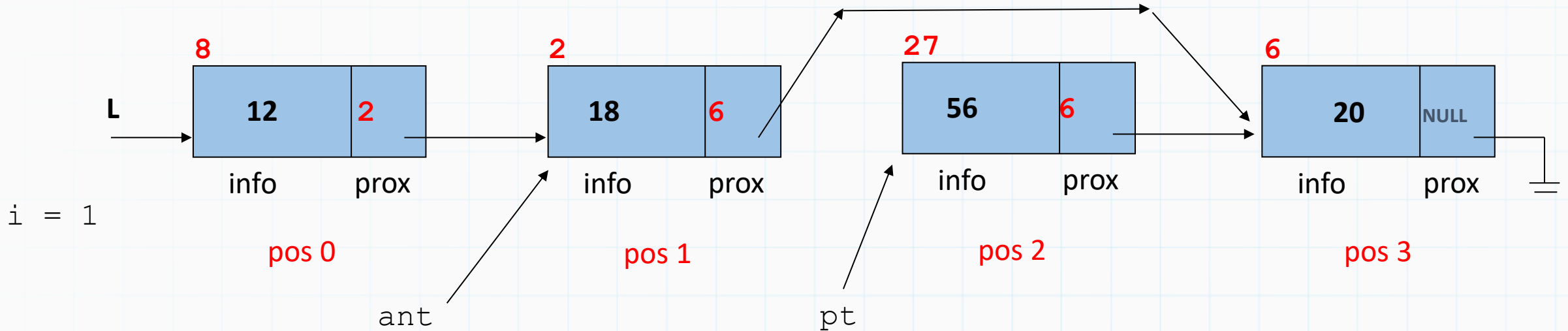
```
pt = L; anterior = NULL;
for(int i=1; i<=pos; i++)
{
    anterior = pt;
    pt      = pt->prox;
}
```

Listas encadeadas



Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos > 0$, (encontra a posição e o anterior a posição) Ex: Queremos remover na $pos = 2$



```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt = pt->prox;
}
```

```
anterior->prox = pt->prox;
```

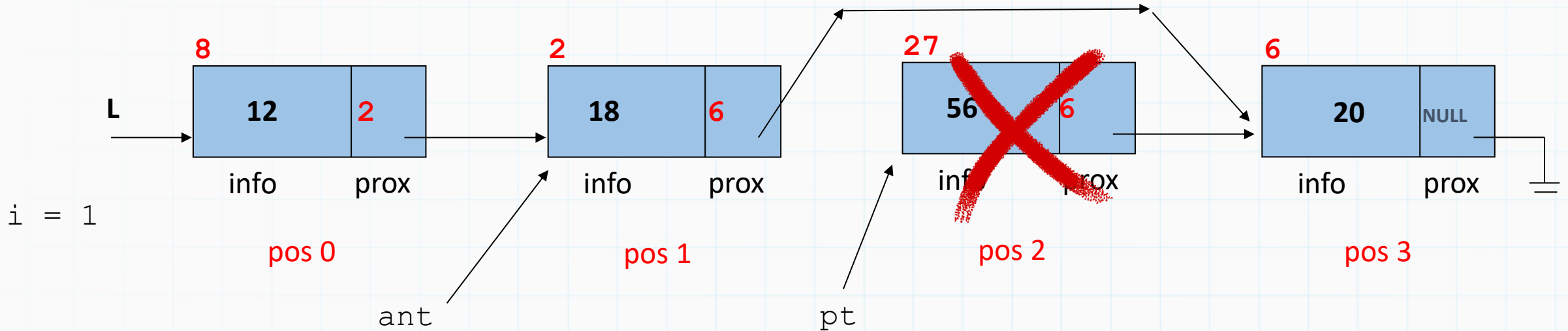


Listas encadeadas



Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos > 0$, (encontra a posição e o anterior a posição) Ex: Queremos remover na $pos = 2$



```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt = pt->prox;
}
```

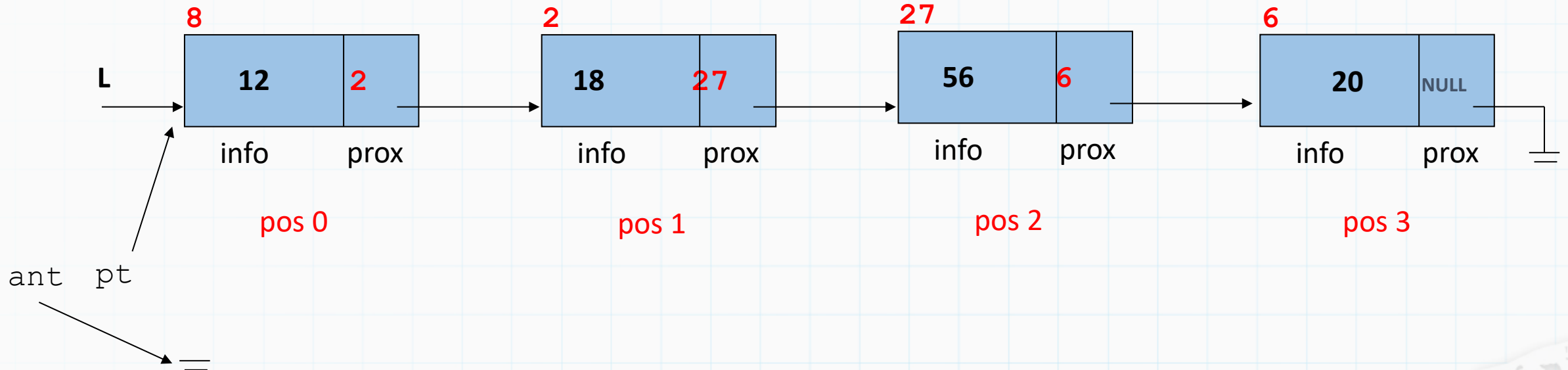
```
anterior->prox = pt->prox;
free (pt) ;
return L;
```

Listas encadeadas



Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos=n-1$, (e se for a ultima posição) Ex: Queremos remover na $pos = 3$



```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt = pt->prox;
}
```

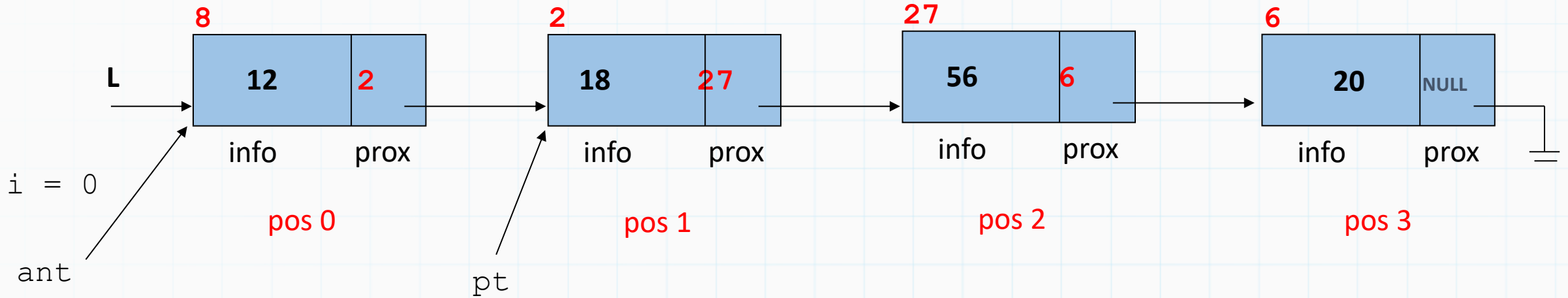
```
anterior->prox = pt->prox;
free(pt);
return L;
```

Listas encadeadas



Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos=n-1$, (e se for a ultima posição) Ex: Queremos remover na $pos = 3$



```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt      = pt->prox;
}
```

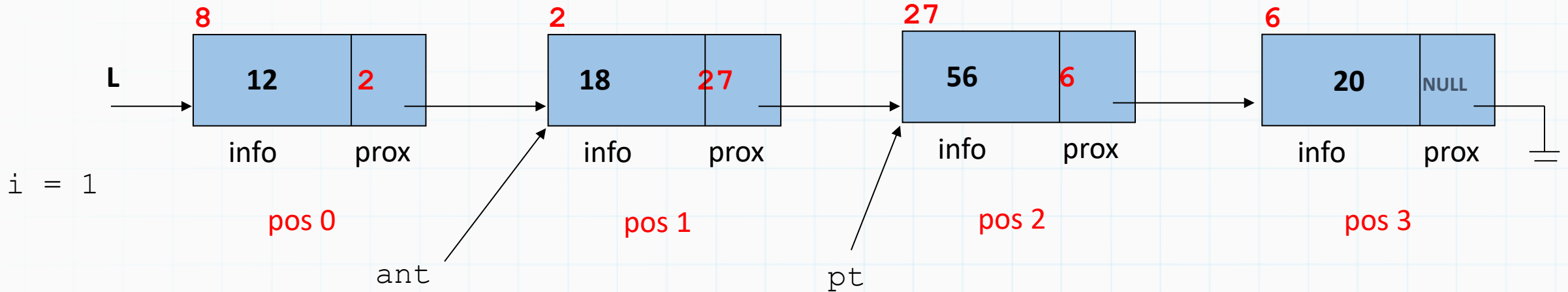
```
anterior->prox = pt->prox;
free(pt);
return L;
```

Listas encadeadas



Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos=n-1$, (e se for a ultima posição) Ex: Queremos remover na $pos = 3$



```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt      = pt->prox;
}
```

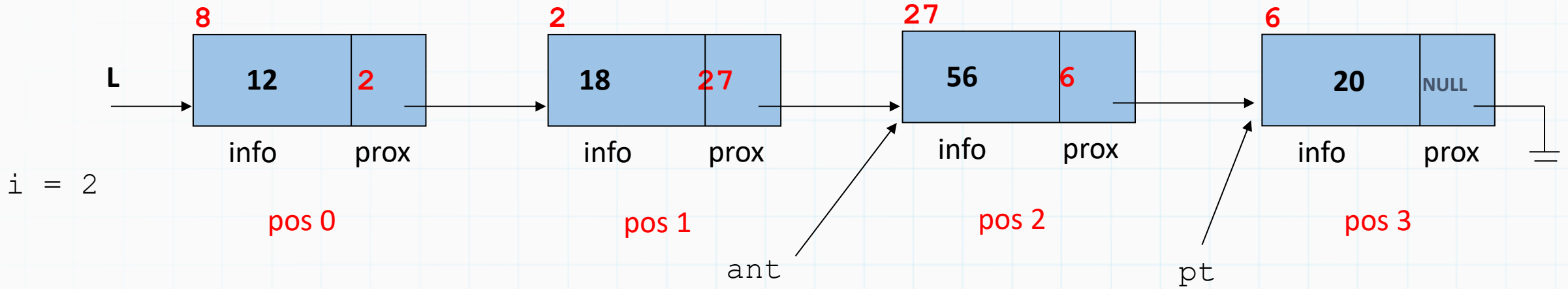
```
anterior->prox = pt->prox;
free(pt);
return L;
```

Listas encadeadas



Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos=n-1$, (e se for a ultima posição) Ex: Queremos remover na $pos = 3$



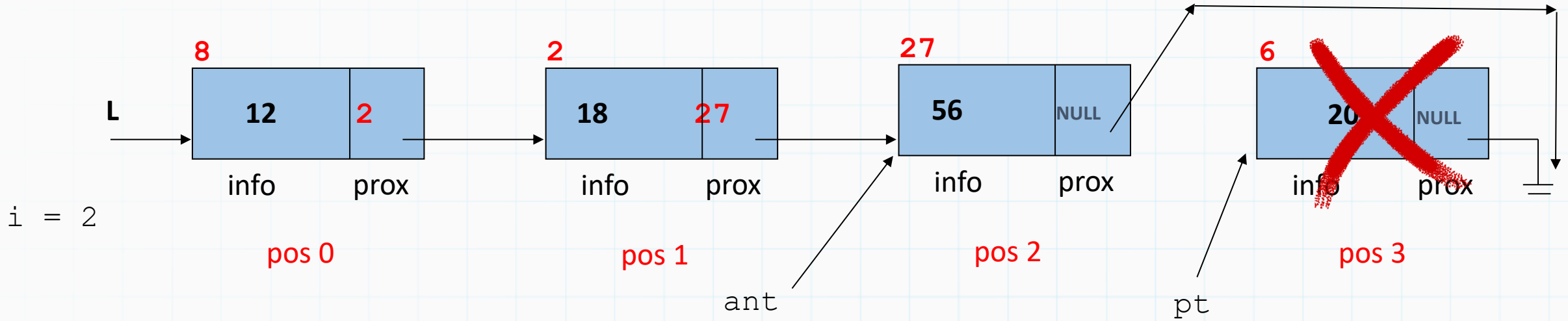
```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt      = pt->prox;
}
```

```
anterior->prox = pt->prox;
free(pt);
return L;
```

Listas encadeadas

Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos=n-1$, (e se for a ultima posição) Ex: Queremos remover na $pos = 3$



```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt = pt->prox;
}
```

```
anterior->prox = pt->prox;
free(pt);
return L;
```

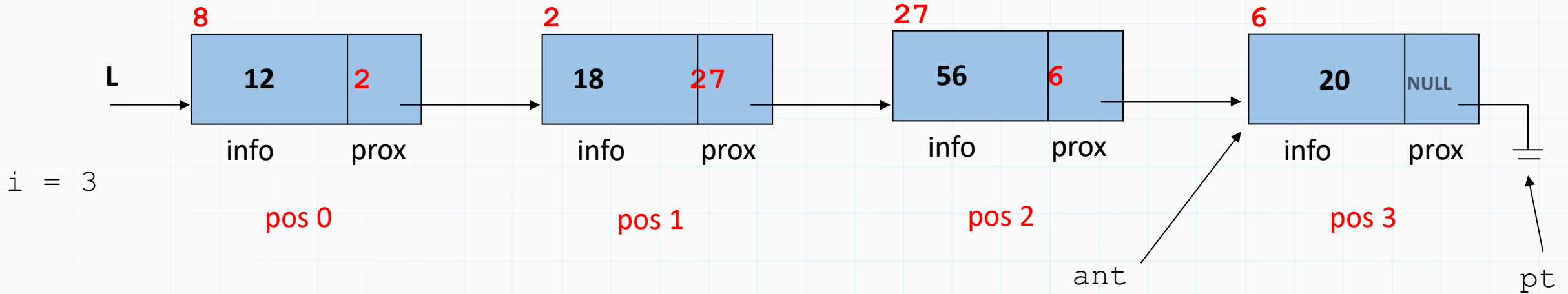


Listas encadeadas



Remover: Remover numa posição específica $pos = \{0, 1, 2, \dots, n-1\}$

$pos > n-1$, (e se for depois da ultima posição) Ex: Queremos remover na $pos = 4$



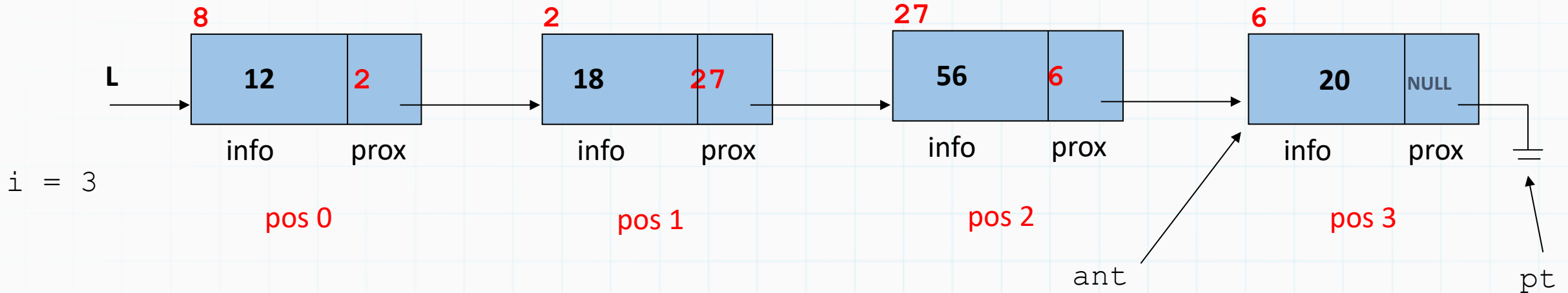
```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt = pt->prox;
}
```

Listas encadeadas



Remover: Remover numa posição específica $pos = \{0, 1, 2, \dots, n-1\}$

$pos > n-1$, (e se for depois da ultima posição) Ex: Queremos remover na $pos = 4$



```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt = pt->prox;
}
```

```
anterior->prox = pt->prox;
free(pt);
return L;
```

ERRO !

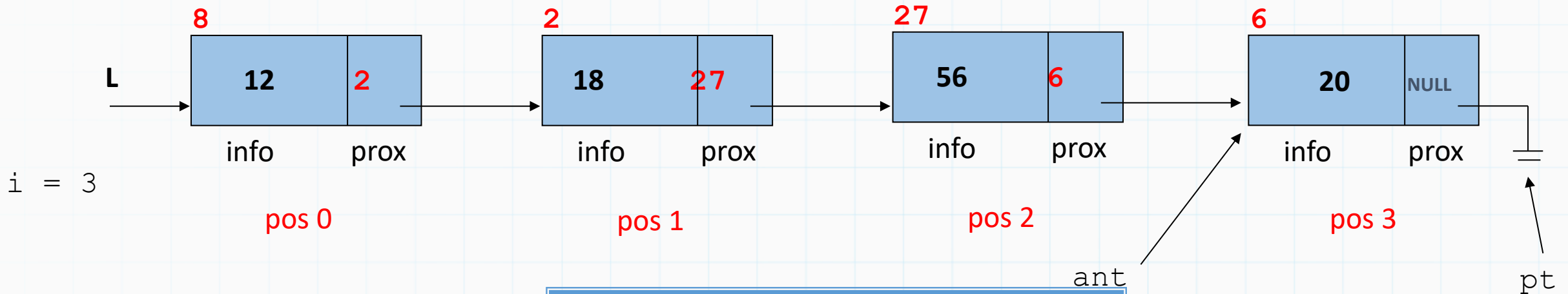


Listas encadeadas



Remover: Remover numa posição específica $pos=\{0, 1, 2, \dots, n-1\}$

$pos > n-1$, (e se for depois da ultima posição) Ex: Queremos remover na $pos = 4$



```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt      = pt->prox;
}
}
```



```
pt = L; anterior = NULL;
for(int i=0; i<pos; i++)
{
    anterior = pt;
    pt      = pt->prox;

    // passou do fim
    if (pt == NULL)
    {
        printf("Posicao invalida\n");
        return L;
    }
}
}
```

Listas encadeadas

```
lista * remove_lista_pos(lista* L, int pos)
{
    lista * pt, * anterior;

    if ((pos < 0) || (L == NULL))
    {
        printf("Posicao invalida/Lista vazia\n");
        return L;
    }

    // remocao no inicio
    if(pos == 0)
    {
        pt = L;
        L = L->prox;
        free(pt);
        return L;
    }
    // -----

```

```

    else
    {
        pt = L;
        // encontra posicao e anterior do elemento
        for(int i=0; i<pos; i++)
        {
            anterior = pt;
            pt = pt->prox;

            // passou do fim
            if (pt == NULL)
            {
                printf("Posicao invalida\n");
                return L;
            }
        }

        anterior->prox = pt->prox;
        free(pt);
        return L;
    }
}

```

Listas encadeadas

```
lista * remove_lista_pos(lista* L, int pos)
{
    lista * pt, * anterior;

    if ((pos < 0) || (L == NULL))
    {
        printf("Posicao invalida/Lista vazia\n");
        return L;
    }

    // remocao no inicio
    if(pos == 0)
    {
        pt = L;
        L = L->prox;
        free(pt);
        return L;
    }
}
```

Use só o que aprendemos até hoje

Exercício 1) Como seria uma função para excluir a lista toda (sem usar a função remove):

```
lista * exclui_lista(lista* L)
```

```
else
{
    pt = L;
    // encontra posicao e anterior do elemento
    for(int i=0; i<pos; i++)
    {
        anterior = pt;
        pt = pt->prox;

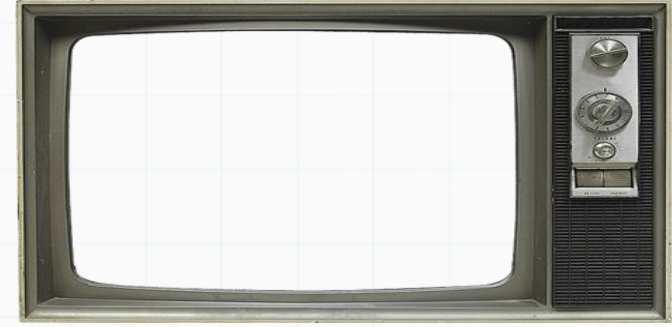
        // passou do fim
        if (pt == NULL)
        {
            printf("Posicao invalida\n");
            return L;
        }
    }

    anterior->prox = pt->prox;
    free(pt);
    return L;
}
```

E como seria uma função de buscar elemento, se achar retorna ponteiro, se não achar retorna NULL:

```
lista* buscar_lista(lista* L, int el)
```

Listas encadeadas



Exercício 1) Como seria uma função para excluir a lista toda (sem usar a função remove):

```
lista * exclui_lista (lista* L)
```

```
lista * exclui_lista (lista* L)
{
    lista* no = L;
    while (no != NULL)
    {
        lista* temp = no->prox;
        free(no);
        no = temp;
    }

    return NULL;
}
```

E como seria uma função de buscar elemento, se achar retorna ponteiro, se não achar retorna NULL:

```
lista* buscar_lista(lista* L, int el)
```

```
lista* buscar_lista(lista* L, int el)
{
    lista* no = L;

    while (no != NULL)
    {
        if (no->info == el)
            return no;

        no = no->prox;
    }

    return NULL;
}
```

Listas encadeadas

Exercício 2) Faça um programa que dado o programa abaixo, use uma função para checar se uma lista esta ordenada (crescente). Considere que uma lista vazia ou com apenas um elemento está ordenada:

```
#include <stdio.h>
#include <stdlib.h>
#include "tad.c"

int main()
{
    lista *L = NULL;

    L = insere_lista(L, 12);
    L = insere_lista(L, 67);
    L = insere_lista(L, 7);
    L = insere_lista(L, 78);
    L = insere_lista(L, 31);
    imprime_lista(L);

    if (esta_ordenada(L) == 1)
        printf("lista esta ordenada\n");
    else
        printf("lista nao esta ordenada\n");


    L = exclui_lista(L);

    return 0;
}
```

insere no
inicio da
lista



Exclui lista



Use só o que
aprendemos
até hoje

Listas encadenadas

```
int esta_ordenada(lista* L)
{
    if (L == NULL)
        return 1;
    else
    {
        int val = L->info;
        lista* no = L->prox;

        while (no != NULL)
        {
            if (no->info < val)
                return 0;

            val = no->info;
            no = no->prox;
        }

        return 1;
    }
}
```

Listas encadeadas

Exercício 3) Faça um programa que dado o programa abaixo, use uma função para ordenar a lista pelo método da bolha.


```
#include <stdio.h>
#include <stdlib.h>
#include "tad.c"

int main()
{
    lista *L = NULL;

    L = insere_lista(L, 12);
    L = insere_lista(L, 67);
    L = insere_lista(L, 7);
    L = insere_lista(L, 78);
    L = insere_lista(L, 31);
    implime_lista(L);

    ordena_bolha(L);
    implime_lista(L);

    L = exclui_lista(L);
    return 0;
}
```



Use só o que aprendemos até hoje

Lembrando:

ordenação da bolha em um vetor v de tamanho n

```
for (i=0; i<n; i++)
    for (j=0; j<n-1; j++)
        if (v[j] > v[j+1])
        {
            temp = v[j];
            v[j] = v[j+1];
            v[j+1] = temp;
        }
```

void ordena_bolha(lista * L)

Utilize a mesma ideia da ordenação em vetor, mas use ponteiros para percorrer a lista

Listas encadeadas

```
void ordena_bolha(lista * L)
{
    lista * i, * j;


    i = L;
    while (i != NULL)
    {
        j = L;
        while (j->prox != NULL)
        {
            if (j->info > j->prox->info)
            {
                int temp      = j->info;
                j->info        = j->prox->info;
                j->prox->info  = temp;
            }

            j = j->prox;
        }

        i = i->prox;
    }
}
```

Listas encadeadas

Exercício 4) Dado uma lista L que esteja em ordem crescente. Implemente uma função que insira um novo elemento **el** na posição correta, de modo que a lista continue ordenada.



Use só o que aprendemos até hoje

```
int main()
{
    lista *L = NULL;

    L = insere_lista(L, 201);
    L = insere_lista(L, 60);
    L = insere_lista(L, 7);
    L = insere_lista(L, 5);
    L = insere_lista(L, 1);
    implime_lista(L);

    // inserir loop
    L = insere_ordenado(L, 10);
    implime_lista(L);

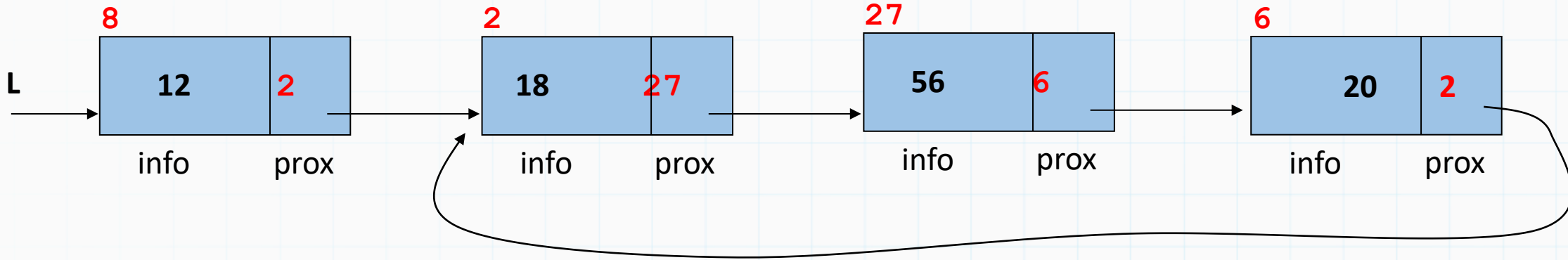
    return 0;
}
```

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

Listas encadeadas



Exercício 5) Escreva uma função que dado uma lista, descobrir se tem um loop nessa lista.



```
int main()
{
    lista *L = NULL;

    L = insere_lista(L, 2);
    L = insere_lista(L, 6);
    L = insere_lista(L, 7);
    L = insere_lista(L, 8);
    L = insere_lista(L, 20);
    implime_lista(L);

    // inserir loop
    L->prox->prox->prox->prox = L;

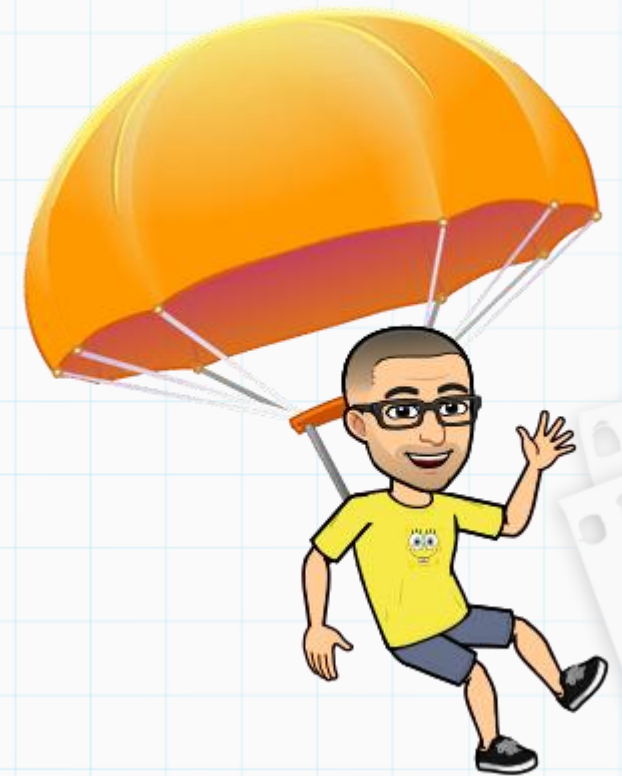
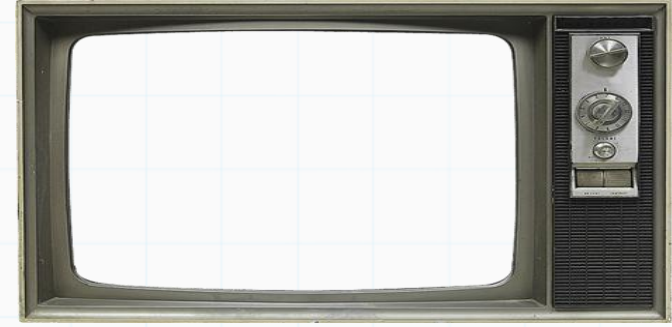
    if (tem_loop(L) == 1)
        printf("achou loop\n");
    else
        printf("nao achou loop\n");
    return 0;
}
```

ideia: caminhar pela lista usando 2 ponteiros (p1 e p2)

- só que a cada iteração p1 anda uma casa
- e p2 anda duas casas

Se houver loop, fatalmente p2 vai se alcançar p1 (dar uma volta) e nesse momento identificamos o loop

Até a próxima



Slides baseados no curso de Aline Nascimento