

# Alocação Encadeada

Com base no que foi visto implemente a operação recup() que compõem o TAD LISTA\_ENC.

```
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODO;
typedef NODO * LISTA_ENC;
```

```
void cria_lista (LISTA_ENC *);
int eh_vazia (LISTA_ENC);
int tam (LISTA_ENC);
void ins (LISTA_ENC *, int, int);
int recup (LISTA_ENC, int);
void ret (LISTA_ENC *, int);
void destruir (LISTA_ENC);
```

```
int recup (LISTA_ENC l, int k)
```

```
{
```

```
    if (k < 1 || k > tam(l))
```

```
    {
```

```
        printf ("\nERRO! Consulta invalida.\n");
```

```
        exit (3);
```

```
    }
```

```
    for (;k>1;k--)
```

```
        l=l->next;
```

```
    return (l->inf);
```

```
}
```



# Alocação Encadeada

Com base no que foi visto implemente a operação ret() que compõem o TAD LISTA\_ENC.

```
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODO;
typedef NODO * LISTA_ENC;
```

```
void cria_lista (LISTA_ENC *);
int eh_vazia (LISTA_ENC);
int tam (LISTA_ENC);
void ins (LISTA_ENC *, int, int);
int recup (LISTA_ENC, int);
void ret (LISTA_ENC *, int);
void destruir (LISTA_ENC);
```

# Alocação Encadeada

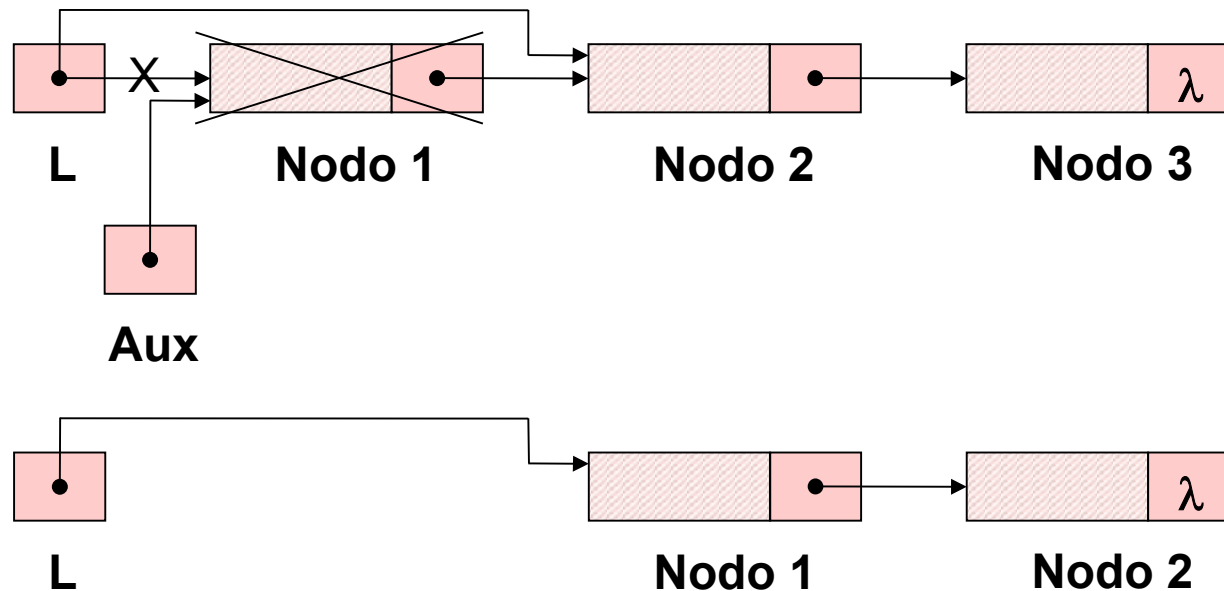
**Questionamentos pré-codificação:**

A posição é válida?

Todas as situações de remoção são tratadas da mesma forma?

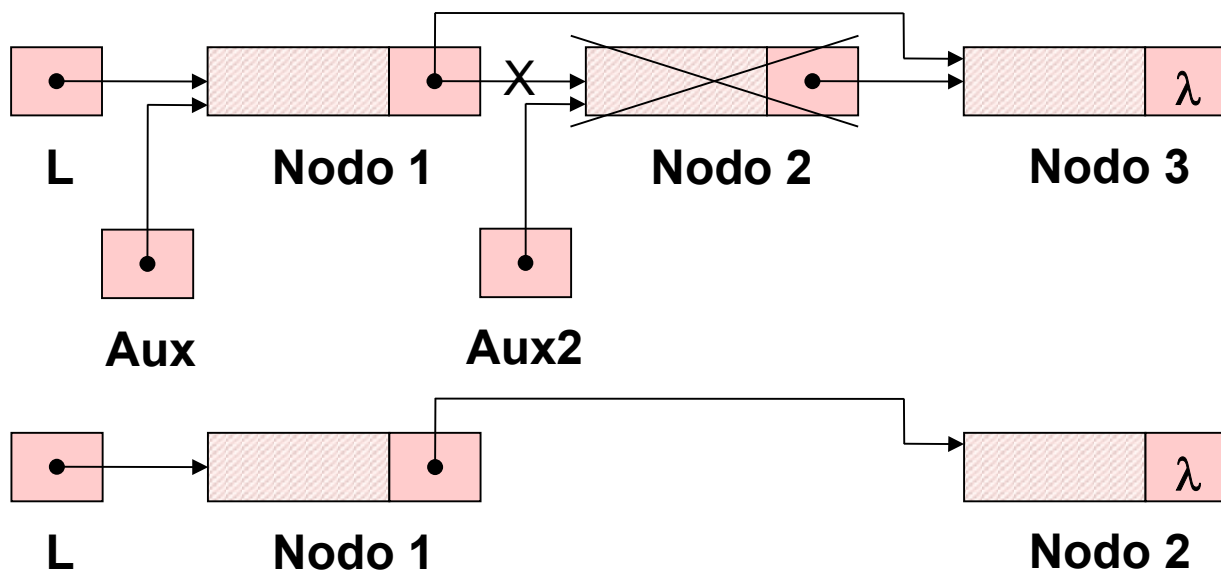
# Alocação Encadeada

Esquema do processo da retirada de um nó da lista. (situação um, remoção do primeiro elemento)



# Alocação Encadeada

Esquema do processo da retirada de um nó da lista. (situação dois, considerando a remoção do elemento da 2ª posição)



```
void ret (LISTA_ENC *pl, int k) {
    NODO *aux;
    if (k < 1 || k > tam(*pl)) {
        printf ("\nERRO! Posição invalida para retirada.\n");
        exit (4);
    }
    if (k==1) {
        aux = *pl;
        *pl = aux->next;
        free (aux);
    }
}
```



**else**

```
{  
  NODO *aux2;  
  for (aux=*pl; k>2; k--, aux=aux->next);  
  aux2 = aux->next;  
  aux->next = aux2->next;  
  free (aux2);  
}  
}
```



# Alocação Encadeada


Com base no que foi visto implemente a operação destruir() que compõem o TAD LISTA\_ENC.

```
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODO;
typedef NODO * LISTA_ENC;
```

```
void cria_lista (LISTA_ENC *);
int eh_vazia (LISTA_ENC);
int tam (LISTA_ENC);
void ins (LISTA_ENC *, int, int);
int recup (LISTA_ENC, int);
void ret (LISTA_ENC *, int);
void destruir (LISTA_ENC);
```

```
void destruir (LISTA_ENC l)
{
    LISTA_ENC aux;
    while (l)
    {
        aux = l;
        l = l->next;
        free(aux);
    }
}
```





# **Listas – Alocação Encadeada – Exercícios**

# Alocação Encadeada

Com base no que foi visto implemente a operação tam() que compõem o TAD LISTA\_ENC utilizando recursividade.

```
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODO;
typedef NODO * LISTA_ENC;
```

```
void cria_lista (LISTA_ENC *);
int eh_vazia (LISTA_ENC);
int tam (LISTA_ENC);
void ins (LISTA_ENC *, int, int);
int recup (LISTA_ENC, int);
void ret (LISTA_ENC *, int);
void destruir (LISTA_ENC);
```

```
int tam (LISTA_ENC I)
{
    if (!I)
        return (0);
    else
        return (1 + tam(I->next));
}
```



# Alocação Encadeada

Com base no que foi visto implemente a operação `recup()` que compõem o TAD `LISTA_ENC` utilizando recursividade.

```
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODO;
typedef NODO * LISTA_ENC;
```

```
void cria_lista (LISTA_ENC *);
int eh_vazia (LISTA_ENC);
int tam (LISTA_ENC);
void ins (LISTA_ENC *, int, int);
int recup (LISTA_ENC, int);
void ret (LISTA_ENC *, int);
void destruir (LISTA_ENC);
```

```
int recup (LISTA_ENC l, int k)  
{  
    if (k < 1 || k > tam(l))  
    {  
        printf ("\nERRO! Consulta invalida.\n");  
        exit (3);  
    }  
    if (k==1)  
        return (l->inf);  
    else  
        return (recup(l->next, k-1));  
}
```



# Alocação Encadeada

Com base no que foi visto implemente a operação destruir() que compõem o TAD LISTA\_ENC utilizando recursividade.

```
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODO;
typedef NODO * LISTA_ENC;
```

```
void cria_lista (LISTA_ENC *);
int eh_vazia (LISTA_ENC);
int tam (LISTA_ENC);
void ins (LISTA_ENC *, int, int);
int recup (LISTA_ENC, int);
void ret (LISTA_ENC *, int);
void destruir (LISTA_ENC);
```

```
void destruir (LISTA_ENC l)
```

```
{
```

```
    if (l)
```

```
    {
```

```
        destruir (l->next);
```

```
        free (l);
```

```
    }
```

```
}
```



# Alocação Encadeada

Com base no que foi visto implemente a operação ins() que compõem o TAD LISTA\_ENC utilizando recursividade.

```
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODO;
typedef NODO * LISTA_ENC;
```

```
void cria_lista (LISTA_ENC *);
int eh_vazia (LISTA_ENC);
int tam (LISTA_ENC);
void ins (LISTA_ENC *, int, int);
int recup (LISTA_ENC, int);
void ret (LISTA_ENC *, int);
void destruir (LISTA_ENC);
```

Recursão...

Qual o critério de parada?

Qual o passo recursivo?



Vamos com calma...

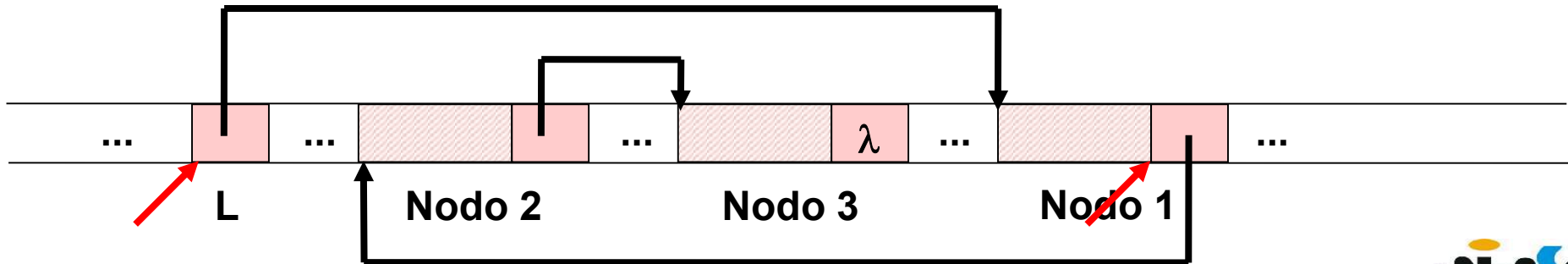
Todas as situações de inserção são tratadas da mesma forma?

Não.

Existem quantas situações? Quais são estas?

Duas, ser o primeiro ou não ser o primeiro.

Pronto, critério de parada e passo recursivo.



```
void ins (LISTA_ENC *pl, int v, int k) {
    if (k < 1 || k > tam(*pl)+1) {
        printf ("\nERRO! Posição invalida para insercao.\n");
        exit (1);
    }
    if (k==1) {
        NODO *novo;
        novo = (NODO *) malloc (sizeof(NODO));
        if (!novo) {printf ("\nERRO! Memoria insuficiente!\n"); exit (2);}
        novo->inf = v;
        novo->next = *pl;
        *pl = novo;
    } else
        ins(&((*pl)->next),v,k-1);
}
```

# Alocação Encadeada

Com base no que foi visto implemente a operação `ret()` que compõem o TAD `LISTA_ENC` utilizando recursividade.

```
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODO;
typedef NODO * LISTA_ENC;
```

```
void cria_lista (LISTA_ENC *);
int eh_vazia (LISTA_ENC);
int tam (LISTA_ENC);
void ins (LISTA_ENC *, int, int);
int recup (LISTA_ENC, int);
void ret (LISTA_ENC *, int);
void destruir (LISTA_ENC);
```

## Alocação Encadeada - Exercício

Implemente, no TAD LISTA\_ENC, a seguinte operação:

```
int pertence (LISTA_ENC l, int v)
```

a qual retorna 1 (um) se v pertence a lista l e 0 (zero) caso contrário.

Faça duas implementações, uma sem fazer uso de recursividade e outra utilizando recursividade.

## Alocação Encadeada - Exercício

Implemente, no TAD LISTA\_ENC, a seguinte operação:

```
int eh_ord (LISTA_ENC l)
```

a qual retorna 1 (um) se a lista l está em ordem crescente e 0 (zero) caso contrário.

Faça duas implementações, uma sem fazer uso de recursividade e outra utilizando recursividade.

## Alocação Encadeada - Exercício

Implemente, no TAD LISTA\_ENC, utilizando recursividade, a seguinte operação:

```
void gera_lista (LISTA_ENC *pl,int m,int n)
```

a qual utilizando-se das operações do TAD LISTA produz uma lista de inteiros correspondente a [m..n].