

# 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);
```

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



## 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.

```
int pertence (LISTA_ENC l, int v)
{
    int t;
    for (t=tam(l); t>0; t--, l=l->next)
        if (l->inf==v)
            return (1);
    return (0);
}
```



```
int pertence (LISTA_ENC l, int v)
{
    while (l)
    {
        if (l->inf==v)
            return (1);
        l=l->next;
    }
    return (0);
}
```



```
int pertence (LISTA_ENC l, int v)  
{  
  if (l)  
  {  
    if (l->inf==v)  
      return (1);  
    return (pertence (l->next, v));  
  }  
  return (0);  
}
```



## 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.

```
int eh_ord (LISTA_ENC l)
{
    int t;
    for (t=tam(l); t>1; t--, l=l->next)
        if (l->inf > (l->next)->inf)
            return (0);
    return (1);
}
```



```
int eh_ord (LISTA_ENC l) {  
    if (eh_vazia(l) || tam(l)==1)  
        return (1);  
    do  
    {  
        if (l->inf > (l->next)->inf)  
            return (0);  
        l=l->next;  
    }while (l->next);  
    return (1);  
}
```



```
int eh_ord (LISTA_ENC l) {  
    if (!l || (l && !(l->next)))  
        return (1);  
    do  
    {  
        if (l->inf > (l->next)->inf)  
            return (0);  
        l=l->next;  
    }while (l->next);  
    return (1);  
}
```



```
int eh_ord (LISTA_ENC l)
{
    if (!l || (l && !(l->next)))
        return (1);
    if (l->inf > (l->next)->inf)
        return (0);
    return (eh_ord (l->next));
}
```



## 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].

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

```
{
```

```
    if (m>n)
```

```
    {
```

```
        printf ("\nERRO! Intervalo invalido.\n");
```

```
        exit (5);
```

```
    }
```

```
else
```





```
if (m==n) {  
    cria_lista (pl);  
    ins (pl, m, 1);  
}  
else {  
    gera_lista (pl, m+1, n);  
    ins (pl, m, 1);  
}  
}
```

# Alocação Encadeada

Com base no TAD LISTA\_ENC, que acabamos de implementar, construa um programa que ofereça ao usuário a possibilidade de inserir, remover e consultar o valor de um elemento em uma lista. Bem como, imprimir a sequência de elementos contidos na mesma.

```
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);
```

# **Listas – Alocação Encadeada com Nó Cabeçalho**

## Alocação Encadeada – Nó de cabeçalho

Ocasionalmente, é desejável manter um nó adicional no início de uma lista.

Esse nó **não** representa um item (elemento) na lista e é chamado *nó de cabeçalho* ou *cabeçalho de lista*.

A parte *inf* deste nó é usada para manter informações globais sobre a lista.

## Alocação Encadeada – Nó de cabeçalho

Por exemplo, a parte *inf* do nó de cabeçalho pode ser usada para armazenar o número de elementos na lista. No caso particular em que temos uma lista com o campo *inf* numérico, um nó de lista pode ser utilizado como nó cabeçalho.

Definiremos agora, um TAD LISTA\_ENC\_NC, o qual representa uma lista linear encadeada dinamicamente com a presença de um nó de cabeçalho contendo no campo *inf* o número de elementos contidos na lista.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_ENC_NC;
7  void cria_lista (LISTA_ENC_NC *);
8  int eh_vazia (LISTA_ENC_NC);
9  int tam (LISTA_ENC_NC);
10 void ins (LISTA_ENC_NC, int, int);
11 int recup (LISTA_ENC_NC, int);
12 void ret (LISTA_ENC_NC, int);
13 void destruir (LISTA_ENC_NC);
```

## Alocação Encadeada – Nó de cabeçalho

Com base no que foi visto implemente a operação `cria_lista()` que compõem o TAD `LISTA_ENC_NC`.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_ENC_NC;
7  void cria_lista (LISTA_ENC_NC *);
8  int eh_vazia (LISTA_ENC_NC);
9  int tam (LISTA_ENC_NC);
10 void ins (LISTA_ENC_NC, int, int);
11 int recup (LISTA_ENC_NC, int);
12 void ret (LISTA_ENC_NC, int);
13 void destruir (LISTA_ENC_NC);
```

```
1 void cria_lista (LISTA_ENC_NC *p1)
2 {
3     *p1 = (NODO *) malloc (sizeof(NODO));
4     if (!*p1)
5     {
6         printf ("\nERRO! Memoria insuficiente!\n");
7         exit (2);
8     }
9     (*p1)->inf = 0;
10    (*p1)->next = NULL;
11 }
```

## Alocação Encadeada – Nó de cabeçalho

Com base no que foi visto implemente a operação `eh_vazia()` que compõem o TAD `LISTA_ENC_NC`.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_ENC_NC;
7  void cria_lista (LISTA_ENC_NC *);
8  int eh_vazia (LISTA_ENC_NC);
9  int tam (LISTA_ENC_NC);
10 void ins (LISTA_ENC_NC, int, int);
11 int recup (LISTA_ENC_NC, int);
12 void ret (LISTA_ENC_NC, int);
13 void destruir (LISTA_ENC_NC);
```

```
1 int eh_vazia (LISTA_ENC_NC l)
2 {
3     return (l->inf==0);
4 }
```

```
1 int eh_vazia (LISTA_ENC_NC l)
2 {
3     return (!(l->next));
4 }
```

## Alocação Encadeada – Nó de cabeçalho

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

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_ENC_NC;
7  void cria_lista (LISTA_ENC_NC *);
8  int eh_vazia (LISTA_ENC_NC);
9  int tam (LISTA_ENC_NC);
10 void ins (LISTA_ENC_NC, int, int);
11 int recup (LISTA_ENC_NC, int);
12 void ret (LISTA_ENC_NC, int);
13 void destruir (LISTA_ENC_NC);
```

```
1 int tam (LISTA_ENC_NC l)
2 {
3     return (l->inf);
4 }
```

## Alocação Encadeada – Nó de cabeçalho

Com base no que foi visto implemente a operação `ins()` que compõem o TAD `LISTA_ENC_NC`.

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_ENC_NC;
7  void cria_lista (LISTA_ENC_NC *);
8  int eh_vazia (LISTA_ENC_NC);
9  int tam (LISTA_ENC_NC);
10 void ins (LISTA_ENC_NC, int, int);
11 int recup (LISTA_ENC_NC, int);
12 void ret (LISTA_ENC_NC, int);
13 void destruir (LISTA_ENC_NC);
```

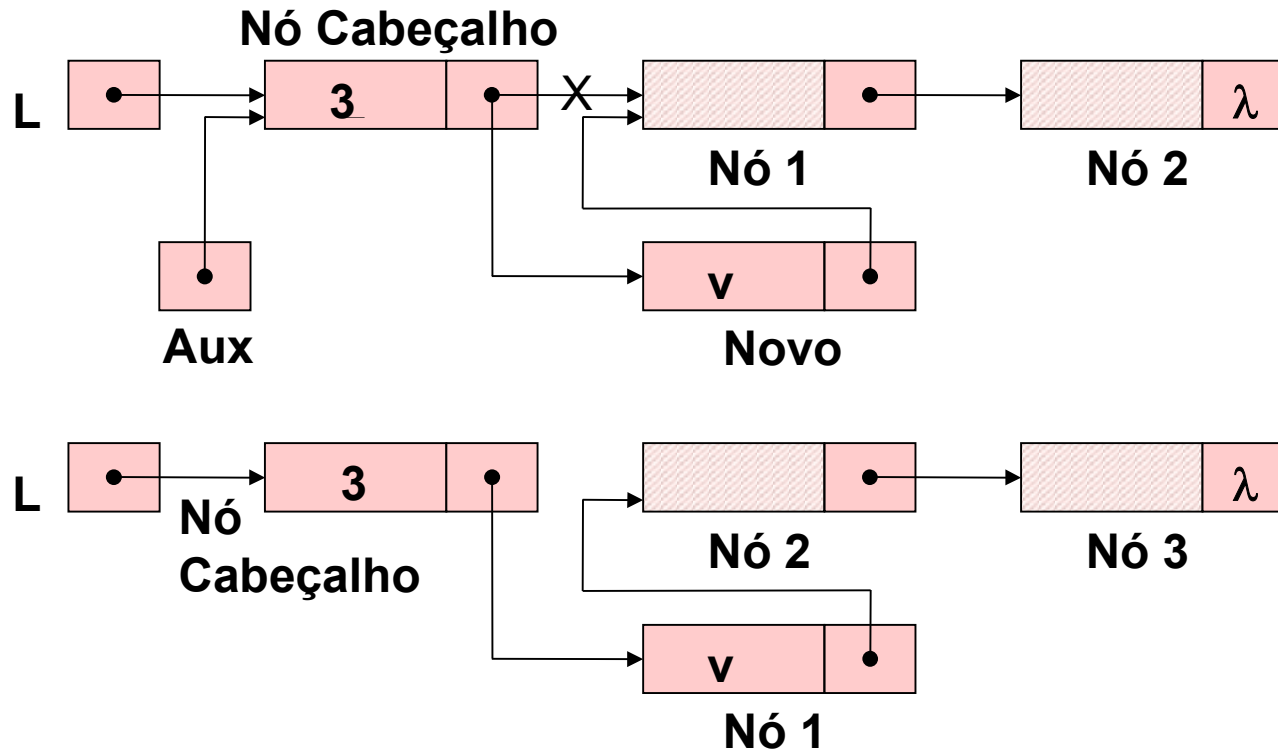
## Alocação Encadeada – Nó de cabeçalho



Este caso particular, da existência do nó cabeçalho com a mesma estrutura de um elemento da lista, elimina a ocorrência de duas situações na operação de inserção.

# Alocação Encadeada – Nó de cabeçalho

Esquema do processo da inserção de um novo nó na lista com nó cabeçalho.



```
1 void ins (LISTA_ENC_NC l, int v, int k)
2 {
3     NODO *novo, *aux;
4     if (k < 1 || k > l->inf/*ou tam()*/+1)
5     {
6         printf ("\nERRO! Posição invalida para insercao.\n");
7         exit (1);
8     }
9     novo = (NODO *) malloc (sizeof(NODO));
10    if (!novo)
11    {
12        printf ("\nERRO! Memoria insuficiente!\n");
13        exit (2);
14    }
```

```
1     novo->inf = v;
2     for (aux=l; k>1; aux=aux->next, k--);
3     novo->next = aux->next;
4     aux->next = novo;
5     l->inf++;
6 }
```

## Alocação Encadeada – Nó de cabeçalho

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

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_ENC_NC;
7  void cria_lista (LISTA_ENC_NC *);
8  int eh_vazia (LISTA_ENC_NC);
9  int tam (LISTA_ENC_NC);
10 void ins (LISTA_ENC_NC, int, int);
11 int recup (LISTA_ENC_NC, int);
12 void ret (LISTA_ENC_NC, int);
13 void destruir (LISTA_ENC_NC);
```

## Alocação Encadeada – Nó de cabeçalho

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

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_ENC_NC;
7  void cria_lista (LISTA_ENC_NC *);
8  int eh_vazia (LISTA_ENC_NC);
9  int tam (LISTA_ENC_NC);
10 void ins (LISTA_ENC_NC, int, int);
11 int recup (LISTA_ENC_NC, int);
12 void ret (LISTA_ENC_NC, int);
13 void destruir (LISTA_ENC_NC);
```

## Alocação Encadeada – Nó de cabeçalho

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

```
1  typedef struct nodo
2  {
3      int inf;
4      struct nodo * next;
5  }NODO;
6  typedef NODO * LISTA_ENC_NC;
7  void cria_lista (LISTA_ENC_NC *);
8  int eh_vazia (LISTA_ENC_NC);
9  int tam (LISTA_ENC_NC);
10 void ins (LISTA_ENC_NC, int, int);
11 int recup (LISTA_ENC_NC, int);
12 void ret (LISTA_ENC_NC, int);
13 void destruir (LISTA_ENC_NC);
```