

## Alocação Sequencial - Exercício

Implemente, no TAD LISTA, a seguinte operação:

```
int pertence (LISTA *l, int v);
```

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

```
int pertence (LISTA *l, int v)
{
    int i;
    for (i=0; i<l->N; i++)
        if (l->val[i]==v)
            return (1);
    return (0);
}
```



## Alocação Sequencial - Exercício

Implemente, no TAD LISTA, a seguinte operação:

```
int eh_ord (LISTA *l);
```

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

```
int eh_ord (LISTA *l)
{
    int i;
    for (i=0; i<l->N-1; i++)
        if (l->val[i] > l->val[i+1])
            return (0);
    return (1);
}
```



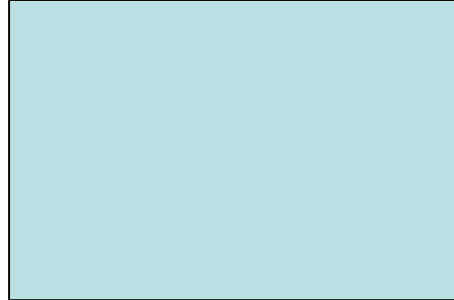
## Alocação Sequencial - Exercício

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

```
void gera_lista (LISTA *l, 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 *l, int m, int n) {  
    if (m>n) {  
        printf ("\nERRO! Intervalo invalido.\n");  
        exit (5);  
    }  
    else  
        if (m==n) {  
            cria_lista (l);  
            ins (l, m, 1);  
        }  
}
```



**else**

**{**

**gera\_lista (l, m+1, n);**

**ins (l, m, 1);**

**}**

**}**





# Listas – Alocação Encadeada

# Alocação Encadeada

Desconsiderando as questões associadas

ao armazenamento estático em memória, você consegue identificar alguma desvantagem em usar o armazenamento sequencial para representar listas?

# Alocação Encadeada

Na alocação encadeada:

- a posição relativa dos elementos na memória não tem que corresponder estritamente à sua posição lógica na estrutura;
- os nós de uma lista encontram-se aleatoriamente dispostos na memória;
- sendo interligados por referências, que indicam a posição do próximo elemento da lista.

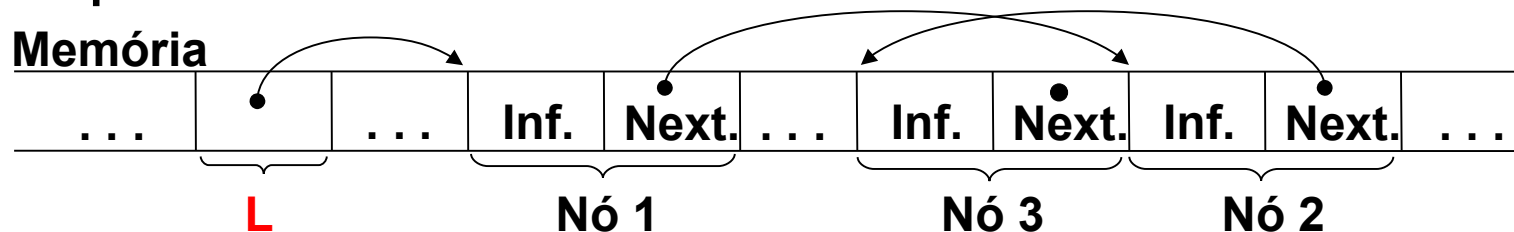
# Alocação Encadeada

Logo:

- aumento no custo de armazenamento  
x flexibilidade;

- acréscimo de um campo em cada nó para armazenar uma referência da posição de seu sucessor.

Esquema:



# Alocação Encadeada

Uma lista encadeada pode ser armazenada em um vetor alocado estaticamente.

A abordagem de armazenar listas encadeadas em vetores estáticos ainda irá impor limitações/desvantagens.

Você identifica alguma limitação/desvantagem?

- uma quantidade fixa de armazenamento permanecer alocada para a lista;
- não mais do que essa quantidade fixa de armazenamento poderá ser utilizada.

# Alocação Encadeada

Como sanar estas limitações/  
desvantagens?

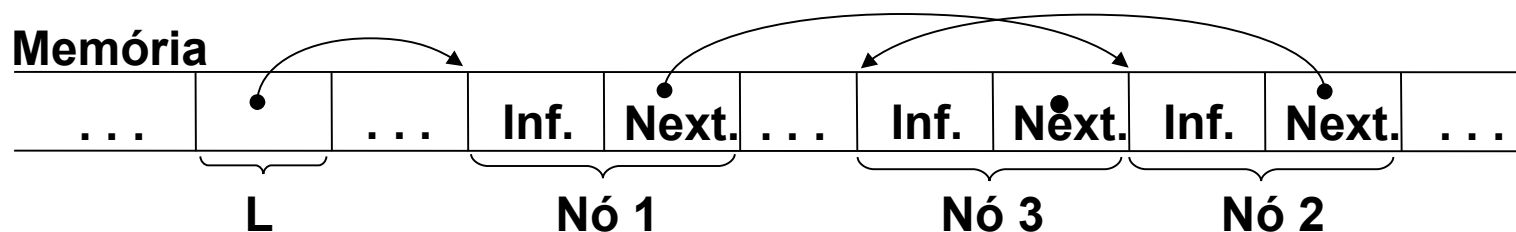
Utilização da alocação dinâmica.



# Alocação Encadeada

Na alocação encadeada dinâmica os nós de uma lista encontram-se aleatoriamente dispostos na memória e são interligados por ponteiros, que indicam a posição do próximo elemento da lista.

Contudo, esta abordagem não elimina a necessidade de um custo de armazenamento adicional por nó (referência ao sucessor). Nem tão pouco a existência da referência externa à lista.



# Alocação Encadeada

Com estas informações, podemos definir o TAD LISTA\_ENC como (considerando que o campo informação armazena um valor inteiro):

```
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

Com base no que foi visto implemente a operação `cria_lista()` 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

```
void cria_lista (LISTA_ENC *pl)
{
    *pl=NULL;
}
```

## Alocação Encadeada

Com base no que foi visto implemente a operação `eh_vazia()` 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

```
int eh_vazia (LISTA_ENC l)  
{  
    return (l == NULL);  
}
```

# Alocação Encadeada

Com base no que foi visto implemente a operação tam() 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

```
int tam (LISTA_ENC l)
{
    int cont;
    for (cont=0; l!= NULL; cont++)
        l = l->next;
    return (cont);
}
```

# Alocação Encadeada

Com base no que foi visto implemente a operação ins() 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

## Dicas:

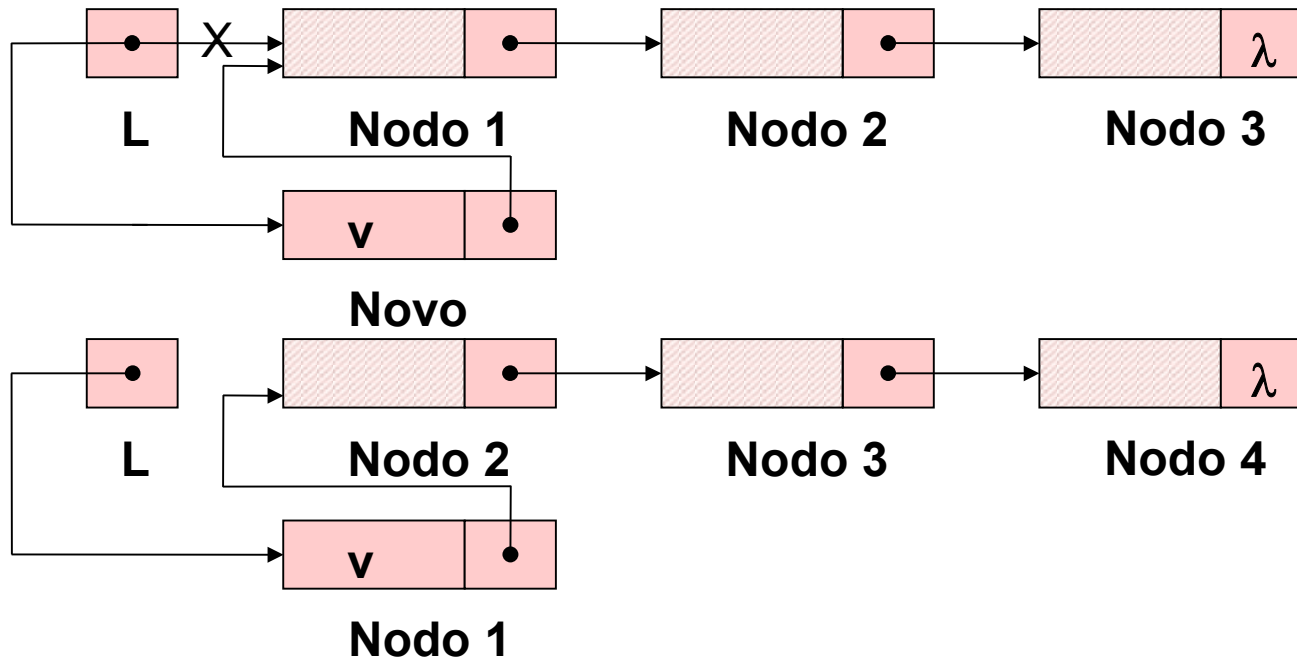
A posição é válida?

Tem espaço na memória para armazenar mais um elemento?

Todas as situações de inserção são tratadas da mesma forma?

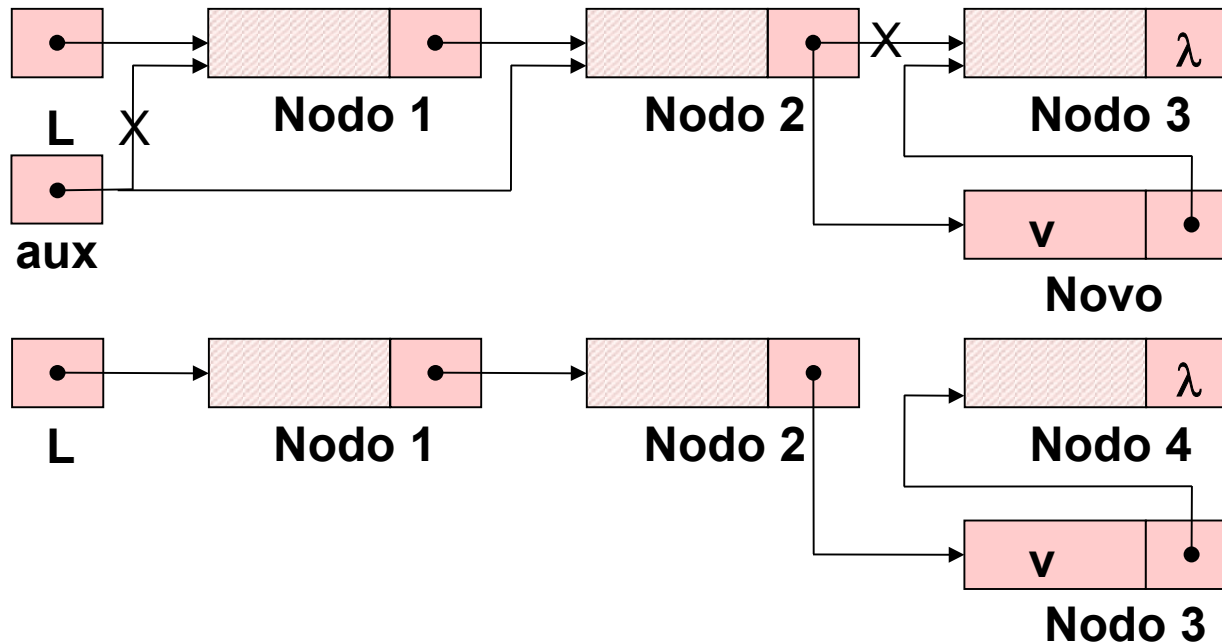
# Alocação Encadeada

Esquema do processo da inserção de um novo nó na lista. (situação um, inserção de um novo primeiro)



# Alocação Encadeada

Esquema do processo da inserção de um novo nó na lista. (situação dois, inserção de um elemento na 3<sup>o</sup> posição)



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

```
ново->inf = v;
```

```
if (k==1){
```

```
    novo->next = *pl;
```

```
    *pl = novo;
```

```
}
```

```
else {
```

```
    LISTA_ENC aux;
```

```
    for (aux=*pl; k>2; aux=aux->next, k--);
```

```
    novo->next = aux->next;
```

```
    aux->next = novo;
```

```
}
```

```
}
```



# 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);
```

# 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);
```

# 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);
```