Alocação Encadeada

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

Você pontuaria 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?

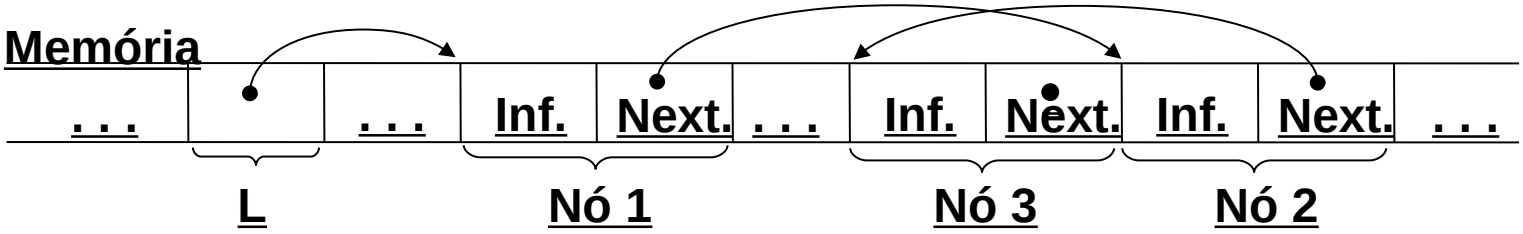
Utilização da alocação dinâmica.

Será mais fácil!

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);
}
```

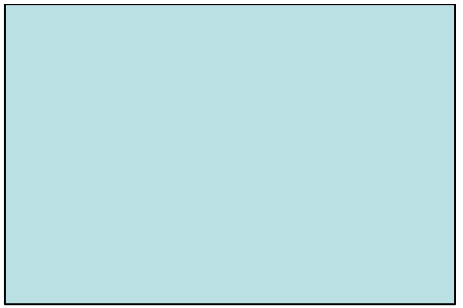
Alocação Encadeada

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



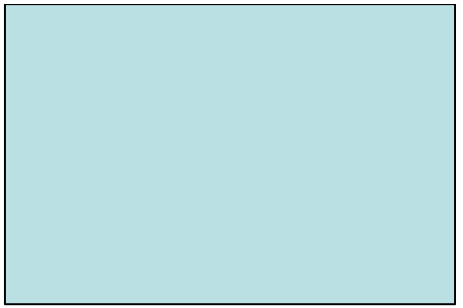
Dicas:

A posição é válida?

Tem espaço na memória para armazenar mais um elemento?

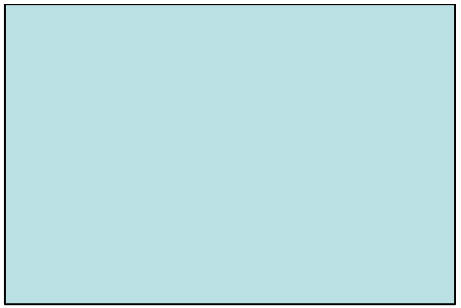
Todas as situações de inserção são tratadas da mesma forma?

UNIVASF



```
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);
   }
```

```
    novo->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);
```

UNIVASF