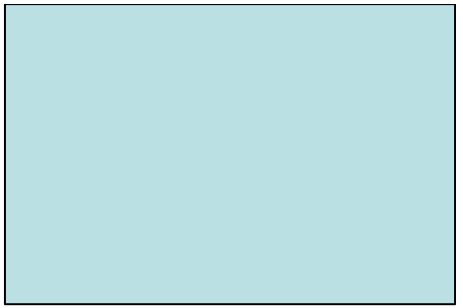
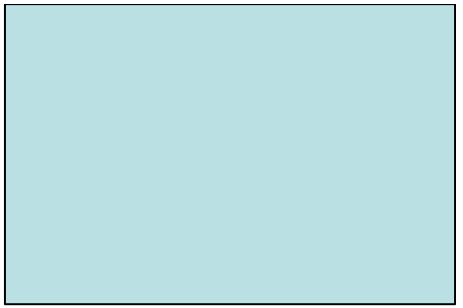
Alocação Encadeada - Exercício



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

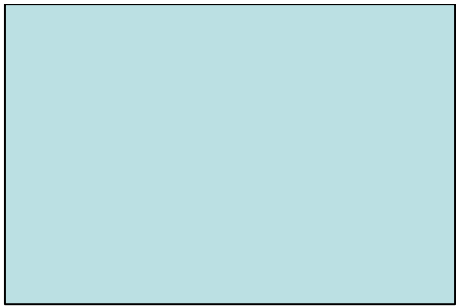
void gera_lista (LISTA_ENC *pl,int m,int n)

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

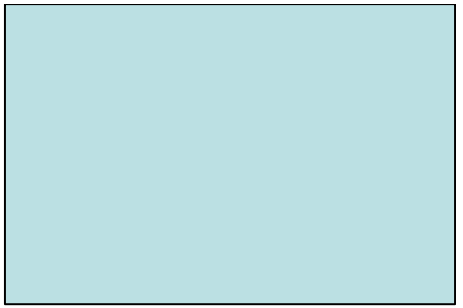


```
void gera_lista (LISTA_ENC *pl, int m,
int n)
{
   if (m>n)
   {
      printf ("\nERRO! Intervalo invalido.\n");
      exit (5);
   }
   else
```

```
    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);
```

UNIVASF

Lista Ordenada – Alocação Encadeada

Alocação Encadeada

Com base em nossos conhecimentos adquiridos, podemos agora definir um novo TAD LISTA_ENC_ORD, no qual os elementos encontram-se ordenados de forma crescente ou decrescente, ou seja, no caso da ordenação crescente, o primeiro elemento é menor ou igual ao segundo, que por sua vez é menor ou igual ao terceiro e assim sucessivamente.

Veremos agora uma definição para o TAD LISTA_ENC_ORD.

```c
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODO;
typedef NODO * LISTA_ENC_ORD;
void cria_lista (LISTA_ENC_ORD *);
int eh_vazia (LISTA_ENC_ORD);
int tam (LISTA_ENC_ORD);
void ins (LISTA_ENC_ORD *, int);
int recup (LISTA_ENC_ORD, int);
void ret (LISTA_ENC_ORD *, int);
void destruir (LISTA_ENC_ORD);
```

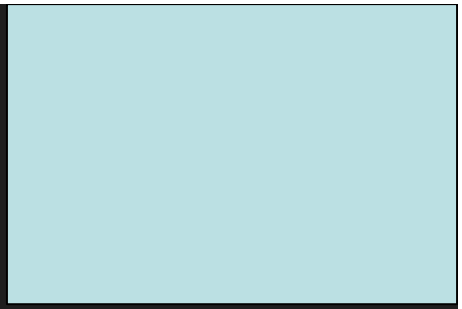
Alocação Encadeada

Utilizando como base a implementação do TAD LISTA_ENC faça as devidas adequações para implementar o TAD LISTA_ENC_ORD (considere a ordem crescente).

Com uma pequena análise, percebe-se que a única operação que requer alteração no TAD LISTA_ENC para o TAD LISTA_ENC_ORD é a operação de inserção.

```
1   void ins (LISTA_ENC_ORD *pl, int v)
2   {
3       NODO *novo; /*LISTA_ENC_ORD novo*/
4       novo = (NODO *) malloc (sizeof(NODO));
5       if (!novo)
6       {
7           printf ("\nERRO! Memoria insuficiente!\n");
8           exit (1);
9       }
10      novo->inf = v;
```

```
11      if (*pl==NULL || v<(*pl)->inf) {
12          novo->next = *pl;
13          *pl = novo;
14      }
15      else {
16          NODO *aux;
17          for (aux=*pl; aux->next!=NULL && v>(aux->next)->inf;
18          aux=aux->next);
19          novo->next = aux->next;
20          aux->next = novo;
21      }
22  }
```



Alocação Encadeada - Exercício

Implemente, no TAD LISTA_ENC_ORD, a seguinte operação:

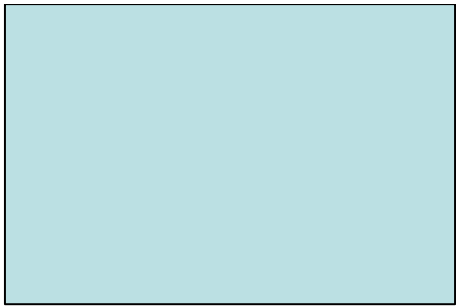
int ret_com_base_no_valor (LISTA_ENC_ORD *, int);

a qual recebe uma referência para uma lista e um valor que deve ser retirado desta. Caso o valor pertença à lista e consequentemente seja retirado a operação retorna 1; caso contrário retorna 0.

```c
int ret_com_base_no_valor (LISTA_ENC_ORD *pl, int v)
{
    NODO *aux=*pl;
    int k=1;
    for (; aux && aux->inf<=v; aux=aux->next, k++)
       if (aux->inf==v)
       {
          ret (pl, k);
          return 1;
       }
    return 0;
}
```

Observação: A solução apenas remove a primeira ocorrência do valor.

Alocação Encadeada - Exercício



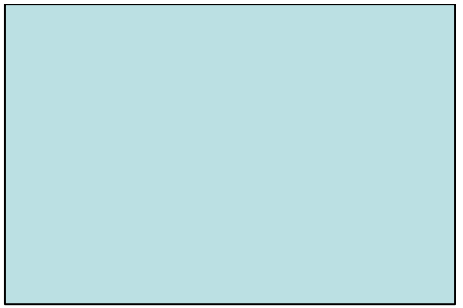
Efetue uma nova implementação da função **ret_com_base_no_valor ()** onde, caso exista mais de uma ocorrência do valor a ser removido, todas sejam retiradas da lista. Neste caso, se ao menos uma remoção foi efetuada a operação retorna 1; caso contrário retorna 0.

```c
int ret_com_base_no_valor (LISTA_ENC_ORD *pl,
int v) {
    NODO *aux=*pl;
    int k=1, retorno=0;
    for (; aux && aux->inf<=v; aux=aux->next, k++)
        if (aux->inf==v) {
            ret (pl, k--);
            retorno=1;
        }
    return retorno;
}
```

```c
int ret_com_base_no_valor (LISTA_ENC_ORD *pl,
int v) {
    NODO *aux=*pl;
    int k=1, retorno=0;
    while (aux && aux->inf<=v)
        if (aux->inf==v) {
            aux=aux->next;
            ret (pl, k);
            retorno=1;
        } else {
            aux=aux->next;
            k++;
        }
    return retorno;
}
```

Listas – Alocação Encadeada com Nó Cabeçalho

Alocação Encadeada – Nó de cabeçalho

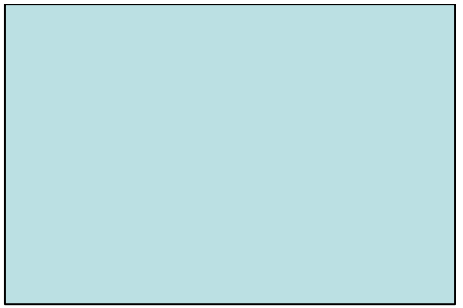


Ocasionalmente, é desejável manter um nó adicional no início de uma lista.

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.

UNIVASF

```c
typedef struct nodo
{
    int inf;
    struct nodo * next;
}NODO;
typedef NODO * LISTA_ENC_NC;
void cria_lista (LISTA_ENC_NC *);
int eh_vazia (LISTA_ENC_NC);
int tam (LISTA_ENC_NC);
void ins (LISTA_ENC_NC, int, int);
int recup (LISTA_ENC_NC, int);
void ret (LISTA_ENC_NC, int);
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);
```

```c
void cria_lista (LISTA_ENC_NC *pl)
{
    *pl = (NODO *) malloc (sizeof(NODO));
    if (!*pl)
    {
        printf ("\nERRO! Memoria insuficiente!\n");
        exit (2);
    }
    (*pl)->inf = 0;
    (*pl)->next = NULL;
}
```

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);
```

```c
int eh_vazia (LISTA_ENC_NC l)
{
   return (l->inf==0);
}
```

```c
int eh_vazia (LISTA_ENC_NC l)
{
   return (!(l->next));
}
```

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);
```

```c
int tam (LISTA_ENC_NC l)
{
   return (l->inf);
}
```

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

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);
```