```
int recup (LISTA_CIRCULAR l, int k)
{
   if (k < 1 || k > tam(l))
   {
      printf ("\nERRO! Consulta invalida.\n");
      exit (3);
   }
   for (;k>0;k--)
      l=l->next;
   return (l->inf);
}
```

Listas Circulares

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

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

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

```
else
{
   NODO *aux, *aux2;
   int i;
   for (aux=*pl, i=k; i>1; i--, aux=aux->next);
   aux2 = aux->next;
   aux->next = aux2->next;
   if (k==tamanho)
      *pl=aux;
   free (aux2);
}
}
```

Listas Circulares – Nó de Cabeçalho

O conceito de ***nó de cabeçalho*** também pode ser empregado nas listas circulares.

A implementação de um TAD LISTA_CIRCULAR_COM_NC é sugerida como uma exercício de fixação.

UNIVASF

Listas Duplamente Encadeadas

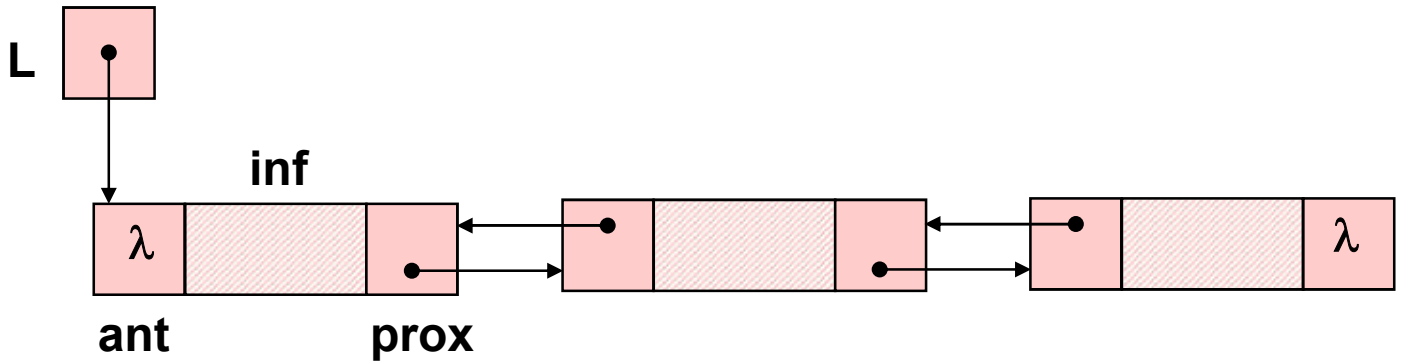
Como vimos, uma lista circular possui vantagens sobre uma lista linear, contudo esta ainda possui limitações.

Por exemplo, não podemos percorrê-la no sentido contrário o que impõe, por exemplo, a necessidade de se ter um ponteiro para o antecessor para inserirmos ou retirarmos um k-ésimo elemento.

Com o objetivo de sanar estas limitações surgiram as ***listas duplamente encadeadas***.

Listas Duplamente Encadeadas

Em uma ***lista duplamente encadeada*** os elementos possuem três campos: o campo ***inf*** o qual contém a informação, o campo ***ant*** que possui um ponteiro para o elemento antecessor e o campo ***prox*** que é uma referência para o elemento que o sucede.



Listas Duplamente Encadeadas

```
/*Exemplo de definição para TAD LISTA_DUP_ENC*/
  typedef struct nodo
  {
       int inf;
       struct nodo * ant;
       struct nodo * prox;
  }NODO;
  typedef NODO * LISTA_DUP_ENC;
  void cria_lista (LISTA_DUP_ENC *);
  int eh_vazia (LISTA_DUP_ENC);
  int tam (LISTA_DUP_ENC);
  void ins (LISTA_DUP_ENC *, int, int);
  int recup (LISTA_DUP_ENC, int);
  void ret (LISTA_DUP_ENC *, int);
```

Listas Duplamente Encadeadas

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

```
typedef struct nodo
{
   int inf;
   struct nodo * ant;
   struct nodo * prox;
}NODO;
typedef NODO * LISTA_DUP_ENC;
void cria_lista (LISTA_DUP_ENC *);
int eh_vazia (LISTA_DUP_ENC);
int tam (LISTA_DUP_ENC);
void ins (LISTA_DUP_ENC *, int, int);
int recup (LISTA_DUP_ENC, int);
void ret (LISTA_DUP_ENC *, int);
```

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

Listas Duplamente Encadeadas

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

```
typedef struct nodo
{
   int inf;
   struct nodo * ant;
   struct nodo * prox;
}NODO;
typedef NODO * LISTA_DUP_ENC;
void cria_lista (LISTA_DUP_ENC *);
int eh_vazia (LISTA_DUP_ENC);
int tam (LISTA_DUP_ENC);
void ins (LISTA_DUP_ENC *, int, int);
int recup (LISTA_DUP_ENC, int);
void ret (LISTA_DUP_ENC *, int);
```

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

Listas Duplamente Encadeadas

Com base no que foi visto implemente a operação tam() que compõem o TAD LISTA_DUP_ENC.

```
typedef struct nodo
{
   int inf;
   struct nodo * ant;
   struct nodo * prox;
}NODO;
typedef NODO * LISTA_DUP_ENC;
void cria_lista (LISTA_DUP_ENC *);
int eh_vazia (LISTA_DUP_ENC);
int tam (LISTA_DUP_ENC);
void ins (LISTA_DUP_ENC *, int, int);
int recup (LISTA_DUP_ENC, int);
void ret (LISTA_DUP_ENC *, int);
```

```
int tam (LISTA_DUP_ENC l)
{
   int cont;
   for (cont=0; l!= NULL /*ou apenas l*/; cont++)
      l = l->prox;
   return (cont);
}
```

Listas Duplamente Encadeadas

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

```
typedef struct nodo
{
   int inf;
   struct nodo * ant;
   struct nodo * prox;
}NODO;
typedef NODO * LISTA_DUP_ENC;
void cria_lista (LISTA_DUP_ENC *);
int eh_vazia (LISTA_DUP_ENC);
int tam (LISTA_DUP_ENC);
void ins (LISTA_DUP_ENC *, int, int);
int recup (LISTA_DUP_ENC, int);
void ret (LISTA_DUP_ENC *, int);
```

```
void ins (LISTA_DUP_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) /*situações um e dois*/
{
   novo->ant = NULL;
   novo->prox = *pl;
   *pl = novo;
   if ((*pl)->prox) /*situação dois*/
      (*pl)->prox->ant=novo;
}
else /*situações três e quatro*/
{
   LISTA_DUP_ENC aux;
   for (aux=*pl; k>2; aux=aux->prox, k--);
```

```
        novo->prox = aux->prox;
        aux->prox = novo;
        novo->ant=aux;
        if (novo->prox) /*situação quatro*/
           novo->prox->ant=novo;
    }
}
```

Listas Duplamente Encadeadas

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

```
typedef struct nodo
{
   int inf;
   struct nodo * ant;
   struct nodo * prox;
}NODO;
typedef NODO * LISTA_DUP_ENC;
void cria_lista (LISTA_DUP_ENC *);
int eh_vazia (LISTA_DUP_ENC);
int tam (LISTA_DUP_ENC);
void ins (LISTA_DUP_ENC *, int, int);
int recup (LISTA_DUP_ENC, int);
void ret (LISTA_DUP_ENC *, int);
```