Listas Dup. Encad. Circ. com NC

Com base no que foi visto implemente a operação criar_lista() que compõem o TAD LISTA_CIR_DUP_ENC_NC.

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

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

Listas Dup. Encad. Circ. com NC

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

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

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

Listas Dup. Encad. Circ. com NC

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

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

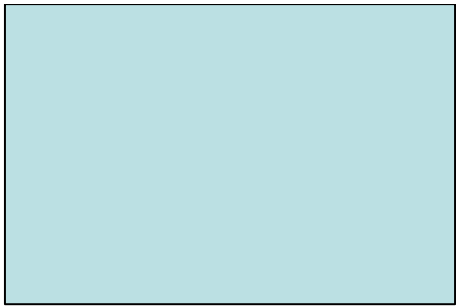
```c
int tam (LISTA_CIR_DUP_ENC_NC l)
{
    return (l->inf);
}
```

Listas Dup. Encad. Circ. com NC

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

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

Listas Dup. Encad. Circ. com NC



Dicas:

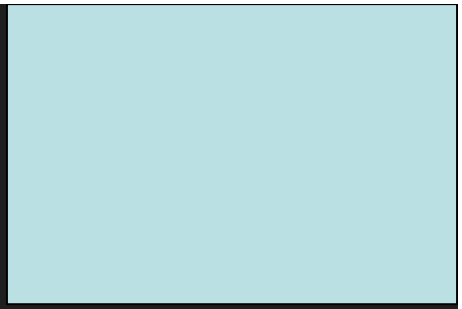
A posição é válida?

Tem espaço na memória para armazenar mais um elemento?

Todas as situações de inserção são tratadas da mesma forma?

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

```
13      novo->inf = v;
14      for (aux=l; k>1; aux=aux->prox, k--);
15      novo->prox = aux->prox;
16      novo->ant = aux;
17      aux->prox = novo;
18      novo->prox->ant=novo;
19      l->inf++;
20  }
```



Listas Dup. Encad. Circ. com NC

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

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

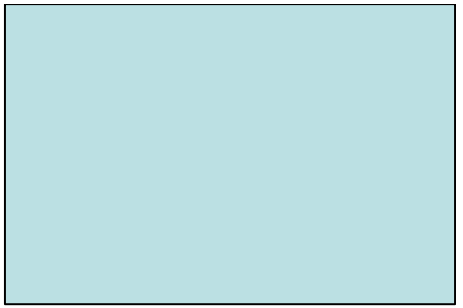
```c
int recup (LISTA_CIR_DUP_ENC_NC l, int k)
{
    if (k < 1 || k > tam(l))
    {
        printf ("\nERRO! Consulta invalida.\n");
        exit (3);
    }
    for (;k>0;k--)
        l=l->prox;
    return (l->inf);
}
```

Listas Dup. Encad. Circ. com NC

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

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

Listas Dup. Encad. Circ. com NC



Dicas:

A posição é válida?

Todas as situações de remoção são tratadas da mesma forma?

```c
void ret (LISTA_CIR_DUP_ENC_NC l, int k)
{
    if (k < 1 || k > tam(l))
    {
        printf ("\nERRO! Posição invalida para retirada.\n");
        exit (4);
    }
    l->inf--;
    for (; k>0; k--, l=l->prox);
    l->ant->prox = l->prox;
    l->prox->ant = l->ant;
    free (l);
}
```

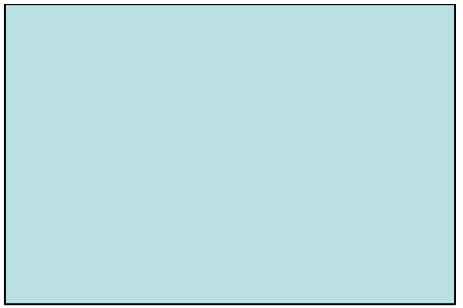
Listas Dup. Encad. Circ. com NC

Com base no que foi visto implemente a operação destruir() que compõem o TAD LISTA_CIR_DUP_ENC_NC.

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

```c
void destruir (LISTA_CIR_DUP_ENC_NC l)
{
    LISTA_CIR_DUP_ENC_NC aux;
    int tam = l->inf;
    do
    {
        aux = l;
        l = l->prox;
        free (aux);
    }while (tam--);
}
```

Listas Dup. Encad. Circ. com NC



Implemente, no **TAD LISTA_CIR_DUP_ENC_NC**, a seguinte operação:

void inverter_lista (LISTA_CIR_DUP_ENC_NC l);

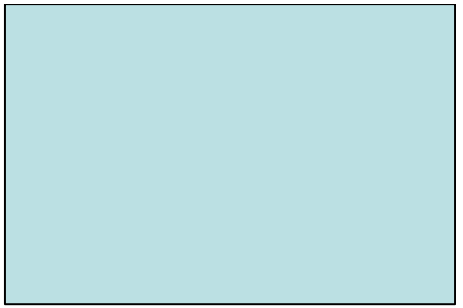
a qual recebe uma referência para uma lista circular duplamente encadeada com nó cabeçalho e inverte a ordem de seus elementos.

UNIVASF

```c
void inverter_lista (LISTA_CIR_DUP_ENC_NC l)
{
    int i=tam(l);
    if (i>1)
    {
        NODO *aux;
        for (i++; i; l=l->ant,i--)
        {
            aux=l->ant;
            l->ant=l->prox;
            l->prox=aux;
        }
    }
}
```

Listas Circulares Duplamente Encadeadas com Nó Cabeçalho Exercícios

Listas Dup. Encad. Circ. com NC



Os espectadores mais atentos de nossas videoaulas já devem ter se perguntado sobre a exploração das vantagens oriundas de cada abordagem.

Por exemplo:

As operações de inserção, recuperação e retirada, apresentadas na videoaula que versou sobre lista circulares duplamente encadeada com nó cabeçalho, poderiam ser mais eficientes?

UNIVASF

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

Listas Dup. Encad. Circ. com NC

Com base no que foi visto reimplemente a operação ins() que compõem o TAD LISTA_CIR_DUP_ENC_NC.

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

```c
void ins (LISTA_CIR_DUP_ENC_NC l, int v, int k)
{
    LISTA_CIR_DUP_ENC_NC aux, novo;
    if (k < 1 || k > l->inf+1) {
        printf ("\nERRO! Posição invalida para insercao.\n");
        exit (1);
    }
    novo = (LISTA_CIR_DUP_ENC_NC) malloc (sizeof(NODO));
    if (!novo) {
        printf ("\nERRO! Memoria insuficiente!\n");
        exit (2);
    }
    novo->inf = v;
```

```
14      if (k<=(l->inf)/2) {
15          for (aux=l; k>1; aux=aux->prox, k--);
16          novo->prox = aux->prox;
17          novo->ant = aux;
18          aux->prox = novo;
19          novo->prox->ant=novo;
20      } else {
21          for (aux=l; k<=l->inf; aux=aux->ant, k++);
22          novo->ant = aux->ant;
23          novo->prox = aux;
24          aux->ant = novo;
25          novo->ant->prox=novo;
26      }
27      l->inf++;
28  }
```

Listas Dup. Encad. Circ. com NC

Com base no que foi visto reimplemente a operação recup() que compõem o TAD LISTA_CIR_DUP_ENC_NC.

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

```c
int recup (LISTA_CIR_DUP_ENC_NC l, int k)
{
    int tamanho = l->inf;
    if (k < 1 || k > tamanho){
       printf ("\nERRO! Consulta invalida.\n");
       exit (3);
    }
    if (k<=tamanho/2)
       for (;k>0;k--)
          l=l->prox;
    else
       for (;k<=tamanho;k++)
          l=l->ant;
    return (l->inf);
}
```

Listas Dup. Encad. Circ. com NC

Com base no que foi visto reimplemente a operação ret() que compõem o TAD LISTA_CIR_DUP_ENC_NC.

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

```c
void ret (LISTA_CIR_DUP_ENC_NC l, int k)
{
   int tamanho = l->inf;
   if (k < 1 || k > tamanho)
   {
      printf ("\nERRO! Posição invalida para retirada.\n");
      exit (4);
   }
   l->inf--;
   if (k<=(l->inf)/2)
      for (; k>0; k--, l=l->prox);
   else
      for (; k<=tamanho; k++, l=l->ant);
   l->ant->prox = l->prox;
   l->prox->ant = l->ant;
   free (l);
}
```