LISTA DUPLAMENTE ENCADEADA

```
STRUCT ELEMENTO{
   INT DADO;
   STRUCT ELEMENTO *PROX;  };
TYPEDEF STRUCT ELEMENTO
ELEMENTO;
TYPEDEF ELEMENTO* LISTA;
LISTA* CRIAR_LISTA(){
    LISTA *LI = (LISTA*)
    MALLOC(SIZEOF(LISTA));
    IF(LI != NULL) *LI = NULL;
     ELSE{ RETURN NULL;
RETURN LI;  }
VOID LIBERAR_LISTA(LISTA *LI){
 IF(LI != NULL){
    ELEMENTO *NO;
    WHILE( (*LI) != NULL ){
       NO = *LI;   *LI = (*LI)->PROX;
     FREE(NO); }
     FREE(LI);
 } }
INT TAMANHO_LISTA(LISTA *LI){
 IF(LI == NULL)    RETURN 0;
 INT CONT = 0;
 ELEMENTO *NO = *LI;
 WHILE(NO != NULL){
    CONT++;
    NO = NO->PROX;  }
 RETURN CONT;}
INT LISTA_VAZIA(LISTA *LI){
 IF(LI == NULL)   RETURN 1;
 IF(*LI == NULL)  RETURN 1;
 RETURN 0;}
INT LISTA_CHEIA(LISTA *LI)
RETURN 0;
 ELEMENTO*
CRIAR_ELEMENTO(){
 ELEMENTO *NO = (ELEMENTO*)
 MALLOC(SIZEOF(ELEMENTO));
 IF(NO == NULL)    RETURN 0;
 RETURN NO;}

INT INS_LIST_INI(LISTA *LI, INT
DADO){
 IF(LI == NULL) RETURN 0;
 ELEMENTO *NO;
 NO = CRIAR_ELEMENTO();
 NO->DADO = DADO;
 NO->PROX = NULL;
 *LI = NO;
 RETURN 1;}
```

```
INT INSER_LIST_FInal(LISTA *LI,
INT DADO){
 IF(LI == NULL)    RETURN 0;
 ELEMENTO *NO;
 NO = CRIAR_ELEMENTO();
 NO->DADO = DADO;
 NO->PROX = NULL;
 IF( (*LI) == NULL )    *LI = NO;
    ELSE{
       ELEMENTO *AUX;
       AUX = *LI;
       WHILE(AUX->PROX != NULL)
          AUX = AUX->PROX;
       AUX->PROX = NO;
    }
 RETURN 1;}

INT INSER_LIST_ORD(LISTA *LI,
INT DADO){
 IF(LI == NULL)  RETURN 0;

 ELEMENTO *NO;
 NO = CRIAR_ELEMENTO();
 NO->DADO = DADO;
 IF( (*LI) == NULL ){
     NO->PROX = NULL;
     *LI = NO;
 }
     ELSE{
        ELEMENTO *ANTERIOR,
*ATUAL;
        ATUAL = *LI;

        WHILE(ATUAL != NULL &&
ATUAL->DADO < DADO){
           ANTERIOR = ATUAL;
           ATUAL = ATUAL->PROX;
        }

     IF( ATUAL == (*LI) ){
         NO->PROX = (*LI);
         *LI = NO;
      }
         ELSE{
            NO->PROX = ATUAL;
            ANTERIOR->PROX = NO;
         }
   }
RETURN 1;}
```

```
INT REMOV_LISTA_INICIO(LISTA *LI){
 IF(LI == NULL)   RETURN 0;
 IF((*LI) == NULL)    RETURN 0;
 ELEMENTO *NO;
 NO = *LI;
 *LI = NO->PROX;
 FREE(NO);
 RETURN 1;}
INT REMOV_LISTA_FINAL(LISTA *LI){
 IF(LI == NULL)    RETURN 0;
 IF((*LI) == NULL)    RETURN 0;
 ELEMENTO *ANTERIOR, *NO;
 NO = *LI;
 WHILE(NO->PROX != NULL){
    ANTERIOR = NO;
    NO = NO->PROX;
 }
 IF(NO == (*LI))  *LI = NO->PROX;
      ELSE
         ANTERIOR->PROX = NO->PROX;
 FREE(NO);
 RETURN 1;}
INT REM_LIST_MEIO(LISTA *LI, INT
DADO){
 IF(LI == NULL)    RETURN 0;
 IF((*LI) == NULL)    RETURN 0;
 ELEMENTO *ANTERIOR, *NO;
 NO = *LI;
 WHILE(NO != NULL && NO->DADO !=
DADO){
    ANTERIOR = NO;
    NO = NO->PROX;
 }
IF(NO == NULL)    RETURN 0;
IF(NO == (*LI))    *LI = NO->PROX;
  ELSE ANTERIOR->PROX =
NO->PROX;
FREE(NO);
RETURN 1;}
INT BUS_LISTA_POS(LISTA *LI, INT
POS)
{
 /*** IMPLEMENTAR ***/
}
INT BUSC_LIST_DADO(LISTA *LI, INT
DADO)
{
 /*** IMPLEMENTAR ***/
}
```

```
STRUCT ELEMENTO{
  INT DADO;
  STRUCT ELEMENTO *PROX; };
TYPEDEF STRUCT ELEMENTO
ELEMENTO;
TYPEDEF ELEMENTO* LISTA;
LISTA* CRIAR_LISTA(){
  LISTA *LI = (LISTA*)
MALLOC(SIZEOF(LISTA));
  IF(LI != NULL)  *LI = NULL;
    ELSE  RETURN NULL;
  RETURN LI;}
INT LIBERAR_LISTA(LISTA *LI){
  IF(LI == NULL)  RETURN 0;
  IF((*LI) != NULL){
     ELEMENTO *NO, *AUX;
     NO = *LI;
     WHILE( NO->PROX !=  (*LI) ){
       AUX = NO;
       NO = NO-> PROX;
       FREE(AUX);
     }
     FREE(NO);
  }
  FREE(LI);
  RETURN 1;}
INT TAMANHO_LISTA(LISTA *LI){
  IF(LI == NULL || (*LI) == NULL)
RETURN 0;
  INT CONT = 0;
  ELEMENTO *NO;
  NO = *LI;
  DO{
     CONT++;
     NO = NO->PROX;
  }WHILE(NO != (*LI));
  RETURN CONT;}
INT LISTA_VAZIA(LISTA *LI){
   IF(LI == NULL)   RETURN -1;
   IF(*LI == NULL)   RETURN 1;
   RETURN 0;}
ELEMENTO* CRIAR_ELEMENTO(){
  ELEMENTO *NO = (ELEMENTO*)
MALLOC(SIZEOF(ELEMENTO));
  IF(NO == NULL)   RETURN 0;
  RETURN NO;}
INT INSERIR_LISTA_INICIO(LISTA
*LI, INT DADO){
  IF(LI == NULL)    RETURN 0;
  ELEMENTO *NO;
  NO = CRIAR_ELEMENTO();
  NO->DADO = DADO;
  IF((*LI) == NULL){
    NO->PROX = NO;
    *LI = NO;
   ELSE{
     ELEMENTO *AUX;
     AUX = *LI;
     WHILE(AUX->PROX != (*LI))
       AUX = AUX->PROX;
     AUX->PROX = NO;
     NO->PROX = *LI;
     *LI = NO;
   }
  RETURN 1;}
INT INSERIR_LISTA_FINAL(LISTA
*LI, INT DADO){
  IF(LI == NULL)    RETURN 0;
  ELEMENTO *NO;
  NO = CRIAR_ELEMENTO();
  NO->DADO = DADO;
  IF( (*LI) == NULL ){
    NO->PROX = NO;
    *LI = NO;
  }
    ELSE{
     ELEMENTO *AUX;
     AUX = *LI;

     WHILE(AUX->PROX != (*LI))
       AUX = AUX->PROX;
     AUX->PROX = NO;
     NO->PROX = *LI;
   }
  RETURN 1;}
INT INSER_LIST_ORD(LISTA *LI,
INT DADO){
  IF(LI == NULL)    RETURN 0;
  ELEMENTO *NO;
  NO = CRIAR_ELEMENTO();
  NO->DADO = DADO;
  IF( (*LI) == NULL ){
    NO->PROX = NO;
    *LI = NO;
  }
     ELSE{
       IF((*LI)->DADO > DADO){
         ELEMENTO *AUX;
         AUX = *LI;
         WHILE(AUX->PROX != (*LI))
           AUX = AUX->PROX;
         AUX->PROX = NO;
         NO->PROX = *LI;
         *LI = NO;
       }
      ELEMENTO *ANTERIOR,
*ATUAL;
       ANTERIOR = *LI;
       ATUAL = ANTERIOR->PROX;
       WHILE(ATUAL != (*LI) &&
ATUAL->DADO < DADO){
         ANTERIOR = ATUAL;
         ATUAL = ATUAL->PROX;
       }
       ANTERIOR->PROX = NO;
       NO->PROX = ATUAL;
    }
  RETURN 1;}
INT
REMOVER_LISTA_INICIO(LISTA
*LI){
  IF(LI == NULL)    RETURN 0;
  IF((*LI) == NULL)    RETURN 0;
  IF ((*LI)->PROX == (*LI)){
    FREE(*LI);
    *LI = NULL;
    RETURN 1;
  }
  ELEMENTO *NO, *AUX;
  NO = *LI;
  AUX = *LI;
  WHILE(AUX->PROX != (*LI)){
    AUX = AUX->PROX;
  }
  AUX->PROX = NO->PROX;
  *LI = NO->PROX;
  FREE(NO);
  RETURN 1;}
INT
REMOVER_LISTA_FINAL(LISTA
*LI){
  IF(LI == NULL)    RETURN 0;
  IF((*LI) == NULL)   RETURN 0;
  IF ((*LI)->PROX == (*LI)){
    FREE(*LI);
    *LI = NULL;
    RETURN 1;
  }
  ELEMENTO *ANTERIOR, *NO;
  NO = *LI;
  WHILE(NO->PROX != (*LI)){
    ANTERIOR = NO;
    NO = NO->PROX;
  }
  ANTERIOR->PROX = NO->PROX;
  FREE(NO);
  RETURN 1;}
INT REM_LIST_MEIO(LISTA *LI,
INT DADO){
  IF(LI == NULL)   RETURN 0;
  IF((*LI) == NULL)    RETURN 0;
  ELEMENTO *NO;
  NO = *LI;
  IF(NO->DADO == DADO){
    IF (NO->PROX == NO){
      *LI = NULL;
      FREE(NO);
      RETURN 1;
   }
     ELSE{
       ELEMENTO *ULTIMO;
       ULTIMO = *LI;
       WHILE(ULTIMO->PROX !=
(*LI))
         ULTIMO =
ULTIMO->PROX;
       ULTIMO->PROX =
NO->PROX;
       *LI = NO->PROX;
       FREE(NO);
       RETURN 1;
     }
  }
  ELEMENTO *ANTERIOR;
  ANTERIOR = NO;
  NO = NO->PROX;
  WHILE(NO != (*LI) &&
NO->DADO != DADO){
    ANTERIOR = NO;
    NO = NO->PROX;
  }
  IF(NO == (*LI))    RETURN 0;
  ANTERIOR->PROX =
NO->PROX;
  FREE(NO);
  RETURN 1;}
INT
BUSCAR_LISTA_POSICAO(LIST
A *LI, INT POS){
  IF(LI == NULL || (*LI) == NULL ||
POS <= 0)
     RETURN -1;
  INT I = 1;
  ELEMENTO *NO;
  NO = *LI;
  WHILE( NO->PROX != (*LI) && I
< POS){
     NO = NO->PROX;
     I++;
  }
  IF(I != POS)    RETURN -1;
  RETURN NO->DADO;}
INT
BUSCAR_LISTA_DADO(LISTA
*LI, INT DADO){
  IF(LI == NULL || (*LI) == NULL)
     RETURN -1;
  INT I = 1;
  ELEMENTO *NO;
  NO = *LI;
  WHILE( NO->PROX != (*LI) &&
NO->DADO != DADO){
     NO = NO->PROX;
     I++;
  }
  IF(NO->DADO != DADO)
RETURN -1;
  RETURN I;}
INT IMPRIMIR_LISTA(LISTA *LI){
  IF(LI == NULL)    RETURN 0;
  IF((*LI) == NULL)  PRINTF("
LISTA VAZIA!");
  ELEMENTO *NO;
  NO = (*LI);
  WHILE(NO->PROX != (*LI)){
    PRINTF(" %D", NO->DADO);
    NO = NO->PROX;
  }
  PRINTF(" %D", NO->DADO);
  RETURN 1;}
```