3. TAD Lista Sequencial

Definição: sequência linear de dados (nós) 0 ou + nós, cuja relação entre os componentes é de ordem posicional.

Tipos abstratos: representação interna + operações de manipulação:

- **incluir nó na lista**

1. *a posição do novo nó decidida pelo usuário do tipo*
2. *a posição do novo nó decidida pelo algoritmo inclusão*

- **excluir nó da lista**

1. *dada a posição*
2. *dada informação de onde o algoritmo de exclusão deve encontrar a posição*

- **acessar nó da lista**

1. *dada a posição*
2. *dada informação de onde o algoritmo de exclusão deve encontrar a posição*

```c
/* Primeiro, a definição de um tipo para o elemento */

typedef struct tinf  tINF;

/* Em seguida, um tipo para a lista  */
struct tipo_lista
{
    tINF    vnos[MAXAL];
    int     qtnos;
    int     maximo;
    int classificada;   // 1: classificada    0: não classificada
    int repeticao;      // 1: com repetição   0: sem repetição
};
typedef struct tipo_lista  tLista;
```

```c
/*cria lista vazia*/
  void cria_lista_vazia(tLista *pl,int maximo,int classif, int repet)  {
       pl->qtnos = 0;
       pl->maximo= maximo;
       pl->classificada=classif;
      pl->repeticao=repet;
  }
```

```c
//lista está vazia?
int lista_vazia( const tLista *pl){
    return ( pl->qtnos == 0);
}
```

```c
//lista está cheia?
int lista_cheia( const tLista *pl){
    return ( pl->qtnos == pl->maximo);
}
```

```c
//lista com repetição?
int lista_repet( const tLista *pl){
    return ( pl->repeticao == 1);
}
```

```c
//lista classificada?
int lista_classif( const tLista *pl){
    return ( pl->classificada == 1);
}
```

```c
//percorre a lista
void percorre(const tLista *pl)  {
    int i;
    for(i = 0; i < pl->qtnos; i++)
             exibe(pl->vnos[i]);
  }
```

Algoritmo de inclusão DADA A POSIÇÃO:

```
Testes de erros:
- se a lista já está cheia:     erro => 0;
- se é classificada  (anterior (k-1) <= novo elemento <= sucessorr (k+1))
    Se nova chave é < que o ant (k-1) ou > que o suc:erro => -2;
- Se é sem repetição   (nenhum nó da lista pode ter a mesma chave do novo)
   Se há algum nó da lista com mesma chave do novo nó: erro => -3;

Inclusão do novo nó
- se é classificada
    chega para lá do k-ésimo ao último
  senão
    coloca o k-ésimo como sucessor do último

coloca o novo nó na posição (k-1) do buffer
aumenta a quantidade de nós da lista```c
```

```c
int inclui_dada_pos(tLista *pl, int pos, T_NO no){
   int ok =1  ;
    //Testes
   if((pos > pl->qtnos) || (pos < 0))
        return -1;

   if ( lista_cheia(pl))
         return 0;

   if(lista_classif(pl)) {
        if( ! lista_vazia(pl))
```

```
            if(pos == 1) /* porque o primeiro não tem antecessor */
                 ok = pl->vnos[pos-1].id >= no.id;
            else
                if(pos == pl->qtnos +1) )  /* porque o últ não tem suc */
                     ok = pl->vnos[pos-2].id <= no.id;
                else
                     ok = ((pl->vnos[pos-1].id < no.id)  &&
                            (pl->vnos[pos-2].id => no.id));

        if (ok != 1)
            return -2;
    }

    if ( ! lista_repet(pl))
        if (verif_exist ( pl,  inf.id,  &pos)
            return -3;
      }
    chegapla(pl,pos,1);
    pl->vnos[pos] = no;
    pl->qtnos++;

    return 1;
}
```

Algoritmo de inclusão DADA A INFORMAÇÃO:

```
Testes de erros:
- se a lista já está cheia:     erro => 0;
- Determinar a pos a incluirdo novo nó preservando os atributos
- Se atributos  preservados
        Abre espaço
        Inclui novo nó

Atributos
classif   repet     o que fazer
  N         S           Pos <- 1ª livre
  N         N           Se nó não existe => pos <- 1ª livre
  S         N           Se nó não existe => pos tal que o ant. é menor
                                que o novo nó e o sucessor é maior que ele
  S         S           Pos tal que a chave identificadora do novo nó é
                        menor ou igual a do antecessor e/ou maior ou igual a do
sucessor

Obs: Ao verificar a existência do nó deve-se levar em consideração o fato da
lista estar ou não classificada, (busca binária ou sequencial) Esta rotina  já
deve determinar onde (a pos) o novo nó deve ser incluído.
```

```
int inclui_dada_inf(tLista *pl, T_NO no){
   int  posinic ;

   if ( lista_cheia(pl))
         return 0;

   if( ( lista_repet (pl) ) && ( !lista_classif(pl) ))
```

```
        posinic=pl->qtnos+1;
   else   {
        ok=  busca_prim_ocorr(pl, chave, &posinic)
        if ( ! lista_repet( pl )) && (ok)
            return -3;   // inclusão de já existente
        }
   if ( lista_classif(pl)
        chegapla(pl,posinic,1);

   pl->vnos[pos] = no;
   pl->qtnos++;
   return 1;
}
```

```
int verif_exist ( const tLista *pl,  tipo chave,  int pos ){
int i, resp;

    if (lista_classif(pl))  /* como está classificado e a pos não “quebra” a
                          classificação (pelo  teste anterior), o nó repetido
só  pode estar nas pos vizinhas a desejada */
        resp = ( (pl->vnos[pos-1].id == chave) || (pl->vnos[pos].id == chave))
    else {
         for(i = 0; (i < pl->qtnos) && (pl->vnos[i].id != chave); i++);
         resp = (i < pl-qtnos);
    }

    return resp;
}
```

Funções Auxiliares

```
//Busca DESORDENADA sem repetição

int busca_des_srep(TLISTA *pl,tipo chave,int *pos){
   int i;

   for(i=0; (i<pl->qtnos)&&(pl->vnos[i].chave!=chave); i++);
   (*pos) = i;

   return(i < pl->qtnos);
}
```

```
//Busca BINÁRIA

int busca_bin(TLISTA *pl, tipo chave, int *pos)
{
```

```
   int inicio = 0, meio, fim = pl->qtnos - 1, achou = 0;

   while((inicio <= fim) && (!achou))   {
      meio = (inicio + fim) / 2;
      if(pl->vnos[meio].chave == chave)
            achou = 1;
      else
            if(pl->vnos[meio].chave > chave)
                 fim = meio - 1;
            else
                inicio = meio + 1;
   }

   if(achou)  {
      (*pos) = meio;
      if (lista_repet(pl))  {
         do
         {
            (*pos)--;
         }while ((*pos)>=0) && (pl->vnos[(*pos)].chave ==chave);
         (*pos)++;
      }
   }
   else
         (*pos) = inicio;
   return achou;
}
```

```
//CHEGA PARA CÁ
void chegaparaca(tLista *pl, int pos, int qt)
{
   int i;

   for(i = pos; i < pl->qtnos - qt; i++)
         vet[i] = vet[i+qt];

}
```

```
// CHEGA PARA LÁ

void chegaparala(tLista *pl, int pos){
   int i;
   for(i = pl->qtnos); i > pos; i--)
         pl->vnos[i] = pl->vnos[i-1];
}
```