1. TAD Lista

Para trabalhar com uma lista seqüencial necessitamos de 2 variáveis:

- vetor de elementos (espaço reservado para armazenar os elementos da lista)
- contador de elementos (armazena o número de elementos da lista. Usualmente, como o primeiro elemento da lista é guardado na primeira posição do vetor, esse contador é suficiente para recuperar a lista propriamente dita, no caso, os elementos da posição 0 até a posição num –1 do vetor)

Para cada lista que desejarmos utilizar, devemos ter um vetor de elementos e um contador (“o kit lista”).

Resolvemos então agrupar essas duas variáveis sob um nome (usando a velha palavra conhecida struct) e definimos assim um novo tipo: O tipo_lista e operações básicas sobre a lista.

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

typedef struct tinf  tINF;

/* Em seguida, um tipo para a lista  */
struct tipo_lista
{
    tINF    vet[MAXAL];
    int     num;
};
typedef struct tipo_lista  tLista;
```

Assim sendo, se desejássemos trabalhar com uma lista de alunos faríamos:

```
/* Primeiro, a definição de um tipo para o elemento */
struct tipo_do_aluno
{
    int     mat;
    char    nome[20];
    int     idade;
    char    sexo;
};
typedef struct tipo_do_aluno  tINF;

/* Em seguida, um tipo para a lista de alunos */
struct tipo_lista_de_alunos
{
    t_aluno     alunos[MAXAL];
    int     num;
};
typedef struct tipo_lista_de_alunos  t_lista_al;
```

Lista em Alocacao Sequencial (Qualquer, sem repetição de elementos):

- **incluir nó na lista**
    1. a posição do novo nó decidida pelo cliente do tipo
    2. a posição do novo nó decidida pelo algoritmo inclusão
- **excluir nó da lista**
    1. dada a posição
    2. dada informação (o algoritmo de exclusão deve encontrar a posição)
- **acessar nó da lista**
    1. dada a posição
    2. dada informação (o algoritmo de acesso deve encontrar a posição)

```c
#include <stdio.h>
#include<stdlib.h>
#define MAXELEM 30

/* Define tipo de 1 elemento da lista. */

typedef struct tno {
    int     id;
    int     campo1;
    int    campo2;

}T_NO;



/* Cria um tipo para lista, agregando as duas componentes necessárias na
implementação de uma lista em alocação sequencial */



typedef struct {
    T_NO     vnos[MAXELEM];  /* buffer p/armazenar os elementos */
    int  qt_nós;  /* Numero de elementos da lista */
} T_LISTA;

 ```C



```C
/* Funcao cria-lista_vazia( ):
recebe (por referencia) uma variavel (ja criada) do tipo t_lista e inicializa
essa variavel. */

void cria_lista_vazia(tT_LISTA *p){
    p->qt_nos = 0;
}```c
```

```c
/*Funcao lista_vazia() testa se a lista está vazia.
Recebe uma lista e retorna 1, se a lista estiver vazia, ou 0, cc */

int lista_vazia(T_LISTA){
    return ( p->qt_nos == 0);
}
```

```c
/*Funcao lista_cheia( ) testa se o espaço para os elementos está todo ocupado.
Recebe uma lista e retorna 1, se o espaco para os elementos da lista esta cheio,
ou 0, caso contrario. */


int lista_cheia(T_LISTA){
    return ( p->qt_nos == MAXELEM);
}
```

```c
/*Funcao acessa( ) verifica se um elemento pertence à lista.
    Recebe uma lista e um valor (chave), busca o elemento com campo id igual a
chave retornando 1, se obteve sucesso, ou 0, caso o elemento nao tenha sido
encontrado.
    No caso do elemento ter sido localizado,  retorna o endereço deste elemento
da lista para que possa ser utilizado */

int lista_acessa() (T_LISTA , int chv, T_NO * pProcurado){

    int ind;   /* Usada  para receber a posição do elemento se ele estiver   */
    if( lista_vazia(plt) )  /* Nao tem ninguém. */
                return 0;
    if (busca (plt, chv, &ind) == 0)
              return (-1);          /* Não existe. */
    *pprocurado = plt->vnos[ind]
    return 1;
}
```

```c
/** Funcao inclui_dada_inf( ): recebe uma lista e um novo elemento a ser inserido
na lista. (Esse elemento pode ou nao ser passado por referencia.)
-->Caso o elemento já esteja na lista, a insercao nao é feita e a funcao retorna
-1.
-->Caso o elemento nao esteja na lista, mas nao exista mais espaco para ele (no
caso do numero de elementos na lista ser igual ao maximo de posições reservadas
para os elementos da lista (MAXELEM), a funcao retorna 0.
-->Caso a insercao seja feita, a funcao retorna 1. */

int lista_inclui(T_LISTA *plt, T_NO *pnovo){
   int ind;    // Usada na chamada da busca
    if (busca (plt, pnovo->id, &ind) == 1)
        return (-1);          /* Ja esta, nao insere. */
     if( lista_cheia(plt) )
         return 0;                  /* Nao tem espaco. */
```

```
     plt->vnos[plt->qt_nos] = *pnovo;            /* Inseriu */
     plt->qt_nos++; /* O numero de elem da lista é incr.*/
      return 1;

}
```

```
/* Funcao exclui_dada_inf( ):
recebe uma lista e valor de id, retirando o elemento o correspondente da lista e
retornando o conteudo desse elemento em uma variavel passada por referencia. Caso
o elemento seja encontrado, a funcao retorna 1, caso contrário, retorna -1 (lista
vazia) ou  0 (se não existe) .*/
int lista_exclui (T_LISTA*plt, int idret, T_NO *psai){
    int ind; /* Usada na chamada da busca */
    if (lista-vazia(plt))
        return -1;
    if (busca (plt, idret, &ind) == 0)
        return 0;      /* Elemento nao encontrado */
    *psai = plt->vnos[ind];  /* Guarda o conteudo do elem. a ser retirado*/
     plt->vnos[ind] = plt->vnos[plt->qt_nos- 1]; /* O ultimo e' copiado para o
lugar do retirado, a fim de nao deixar "buracos"*/
     plt->qt_nos--;/* O numero de elem. da lista e decr */
     return 1;
}
```

```
/*--------------------------------------------------------------------*/

int busca (T_LISTA *p, int proc, int *pos){
    int i;
    for (i=0; i<p->qt_nos; i++)
    if (p->vnps[i].id == proc){}
        *pos = i;      /* Achou, informa a posicao */
        return 1;
    }
    *pos = -1;
    return 0;
}
```