**OPERAÇÕES EM LISTAS**

1) criarLista(A) [IN: não há] [OUT: A][OBJETIVO: criar lista A vazia]

```
nA ← 0
```

2) construirLista(A,n,item) [IN: n,item] [OUT: A][ OBJETIVO: construir lista com n elementos com valor item]

```
se (n>0) e (n < MaximoA)  então nA ← n  senão nA ← MaximoA-1;
para j de 1 até nA repita  A[j] ← item
```

Consideramos que a estrutura de armazenamento de dados possa armazenar MaximoA-1 elementos.Se o valor de n (quantidade de elementos) não pertencer à faixa [1..MaximoA-1], a lista será construída com o número máximo de elementos possível.

3) esvaziarLista(A) [IN: A] [OUT: A][ OBJETIVO: tornar a lista A vazia]

```
nA ← 0
```

4) obterTamanho(A,n) [IN: A] [OUT: n][ OBJETIVO: obter tamanho da lista A]

```
n ← nA
```

5) verificarListaVazia(A,ok) [IN: A] [OUT: ok][ OBJETIVO: obter true/false caso A seja vazia ou não]

```
se (nA = 0)  então  ok ← verdadeiro  senão  ok ← falso
```

6) obterElemento(A,p,item) [IN: A,p] [OUT: item][ OBJETIVO: obter elemento que está na posição p]

```
se ((p ≥1) e (p ≤nA)) então item ← A[p]  senão  item ← Fantasma
```

Se o valor de p não é um valor válido, devolvemos um sinal de erro, indicado pela constante Fantasma.

7) buscarElemento(A,item,p) [IN: A,item] [OUT: p] [OBJETIVO: obter posição do item na lista A]

```
p ← 0;
se (nA ≠ 0) então   k ← 1;
                    enquanto ((A[k] ≠ item) e (k < nA)) faça  k ← k+1;
                    se (A[k] = item) então p ← k
```

Admitimos como pré-condição: a lista A está definida. Se o item não for encontrado na lista A, devolvemos p=0, pois a posição 0 não corresponde a nenhum elemento da lista.

8) inserirNoFim(A,novo) [IN: A, novo][OUT: A][OBJETIVO: inserir novo elemento no final da lista A]

```
p ← nA + 1;  A[p] ←  novo;   nA ← p
```

Admitimos como pré-condição: a lista A está definida e o espaço reservado para a lista A ainda permite a inserção de um novo valor.

9) removerLocal(A,p) [IN: A, p] [OUT: A][OBJETIVO: remover o elemento da posição p]

```
se ((p ≥1) e (p ≤nA))
      então   se (p < nA) então  para j de (p+1) até nA repita  A[j-1] ← A[j];
              nA ← nA - 1
```

Admitimos como pré-condição: a lista A está definida.

10) construirListaRam(a,b,n) [IN: a,b,n] [OUT: A][OBJETIVO: construir lista aleatória com n inteiros na faixa de a até b.

**Exercício 1 – Fazer as implementações das funções declaradas na interface, considerando a definição do tipo ListaInt. Testar todas as funções.**

**/* TAD ListaInt exemplo 1*/**

```
/*
   Uma lista de números inteiros positivos é armazenada em um array de inteiros identificado por vetor. Os itens da
   lista são armazenados partir da posição 1. O campo tamanho guarda o comprimento da lista.
*/
      #include<Booleano.h>
      #define Maximo 6
      #define Fantasma 0
      typedef struct {
         int tamanho;
         int  vetor[Maximo];
      } ListaInt;


//interface
      ListaInt criarLista();                    // construtor
      ListaInt construirLista(int,int);         // constructor: n,valor
      ListaInt construirListaRam(int,int,int)   // construtor: [a..b],n
      ListaInt esvaziarLista(ListaInt);         // destruidor

      int obterTamanho(ListaInt);           // acesso
      bool verificarListaVazia(ListaInt);   // acesso
      int obterElemento(ListaInt, int);     // acesso
      int buscarElemento(ListaInt, int);    // acesso
      void mostrarLista(ListaInt);          // acesso

      ListaInt inserir(ListaInt,int);       // manipulação: inserir no fim da lista
      ListaInt removerLocal(ListaInt,p);    // manipulação: remover o que está em p
```

**Exercício 2 – Fazer as implementações das funções declaradas na interface, considerando a definição do tipo ListaChar. Testar todas as funções.**

**/* TAD ListaChar exemplo 2*/**

```
/* Uma lista de letras em um array identificado por elemento. Os itens da lista são armazenados partir da posição 1. O
   campo tamanho guarda o comprimento da lista. */

      #include<Booleano.h>
      #define Maximo 6
      #define Fantasma '0'
      typedef struct {
         int tamanho;
         unsigned char  vetor[Maximo];
      } ListaChar;

//interface
      ListaChar criarLista();                          // construtor
      ListaChar construirLista(int, unsigned char);    // constructor: n, valor
      ListaChar construirListaAlfa()                   // construtor: alfabeto maiúsculas
      ListaChar esvaziarLista(ListaChar);              // destruidor

      int obterTamanho(ListaChar);                     // acesso
      bool verificarListaVazia(ListaChar);             // acesso
      unsigned char obterElemento(ListaChar, int);     // acesso
      int buscarElemento(ListaChar, unsigned char);    // acesso
      void mostrarLista(ListaChar);                    // acesso

      ListaChar inserir(ListaChar, unsigned char);     // manipulação: inserir no fim da lista
      ListaChar removerLocal(ListaChar, p);            // manipulação: remover o que está em p
```

**Exercício 3 –Construir o arquivo header Elemento.h. Testar todas as funções.**

**/* TAD TipoElemento exemplo 3*/**
/* TipoElemento guarda os dados referentes a um aluno: nome e RA. */

```
#ifndef _Elemento_h
        #define _Elemento_h
typedef struct{
      char * nome;
      char *RA;
}TipoElemento;

    TipoElemento criarElemento(char *, char *);
    TipoElemento criarNovoElemento();
    TipoElemento criarFantasma();
    char* get_Nome(TipoElemento);
    char* get_RA(TipoElemento );
    void mostrarElemento(TipoElemento);
    void set_Nome(TipoElemento *, char *);
    void set_RA(TipoElemento *, char *);

//implementações
TipoElemento criarElemento(char *s, char * r){
        TipoElemento reg;
        reg.nome = s; reg.RA = r;
        return reg;
}

TipoElemento criarNovoElemento(){
        TipoElemento novo;
        novo.nome = "MARIA"; novo.RA = “00000001”;
        return novo;
}

TipoElemento criarFantasma(){
        TipoElemento f;
        f.nome = "fantasma"; f.RA = “00000000”;
        return f;
}

void set_Nome(TipoElemento *reg, char *s){
      reg->nome = s;
}

void set_RA(TipoElemento *reg, char *m){
   reg->RA = m;
}

char * get_Nome(TipoElemento reg){
   char * s;
   s = reg.nome;
   return s;
}

char get_RA(TipoElemento reg){
   char * m;
   m = reg.RA;
   return m;
}

void mostrarElemento(TipoElemento reg){
   printf(" nome = %s   ",reg.nome);      printf("   RA%s   ", reg.RA);
}

#endif
```

**Exercício 4**

**Construir o TAD Lista usando a estrutura a seguir. Testar todas as funções básicas de lista. Definir o que for necessário.**

```
typedef struct {
   int tamanho;
   TipoElemento  tab[Maximo];
} Lista;
```