INF1301 – Programação Modular

Prof. Ivan Mathias Filho

Exercício 2.3

O objetivo deste exercício é implementar um TAD Lista por meio de uma lista duplamente encadeada. Para tal, você deverá criar um arquivo referente à interface do módulo (.h) e outro referente à sua implementação (.c).

Structs e typedefs

As seguintes **structs** terão de ser usadas na implementação:

```
struct no {
   void *info;
   struct no *prox;
   struct no *ant;
};
struct lista {
      int tam;
      No *ini;
      No *fin;
      No *corr;
};
```

Para cada **struct** terá de ser criado um **typedef** correspondente. São eles:

```
typedef struct no No;
typedef struct lista Lista;
```

Funções do TAD Lista

Lista *lst_cria(void) – cria o header de uma lista (lista vazia) e retorna o seu endereço (endereço do header).

int lst_vazia(Lista *) – retorna **1** se a lista estiver vazia ou **0**, caso contrário.

void lst_insIni(Lista *, void *) – insere um elemento (void *) no início da lista (Lista *).

void lst_insFin(Lista *,void *) – insere um elemento (void *) no final da lista (Lista *).

void *lst_retIni(Lista *) – retira o primeiro elemento da lista (Lista *) e retorna o seu endereço. Caso a lista esteja vazia a função deve retornar **NULL**. Obviamente, esta função deve manter o encadeamento da lista recebida como parâmetro.

void *lst_retFin(Lista *) – retira o último elemento da lista (Lista *) e retorna o seu endereço. Caso a lista esteja vazia a função deve retornar **NULL**. Obviamente, esta função deve manter o encadeamento da lista recebida como parâmetro.

void lst_posIni(Lista *) – esta função será usada para percorrer sequencialmente uma lista, do primeiro até o último elemento. Quando ela for executada o primeiro elemento da lista passará a ser o elemento corrente (campo **corr**). Caso a lista esteja vazia, o campo **corr** irá conter o valor **NULL** após a execução desta função.

void lst_posFin(Lista *) – esta função será usada para percorrer sequencialmente uma lista, do último até o primeiro elemento. Quando ela for executada o último elemento da lista passará a ser o elemento corrente (campo **corr**). Caso a lista esteja vazia, o campo **corr** irá conter o valor **NULL** após a execução desta função.

void *lst_prox(Lista *) – esta função será usada para percorrer sequencialmente uma lista, do primeiro até o último elemento. Ela retorna o endereço do elemento armazenado no nó corrente (campo **corr**) e faz com que o campo **corr** referencie o próximo nó da lista. Caso a campo **corr** contenha o valor **NULL** esta operação deverá retornar o valor **NULL**.

void *lst_ant(Lista *) – esta função será usada para percorrer sequencialmente uma lista, do último até o primeiro elemento. Ela retorna o endereço do elemento armazenado no nó corrente (campo **corr**) e faz com que o campo **corr** referencie o nó anterior ao nó corrente. Caso a campo **corr** contenha o valor **NULL** esta operação deverá retornar o valor **NULL**.

void lst_libera(Lista *) – esta função deverá percorrer sequencialmente uma lista recebida como parâmetro (Lista *) e para cada nó encontrado ela deverá: a) liberar a área de memória referente ao elemento apontado pelo nó (**info**) e b) liberar a área de memória do próprio nó. Após todos os nós terem sido liberados, o header da lista deverá ser liberado.

Campos da struct no

void *info – endereço de um elemento (genérico) pertencente à lista.

struct no *prox – endereço do próximo nó da lista. O valor deste campo referente ao último nó da lista será **NULL**.

struct no *ant – endereço do nó anterior da lista. O valor deste campo referente ao primeiro nó da lista será **NULL**.

Campos da struct lista (header da lista)

int tam – valor correspondente à quantidade de elementos da lista.

No *ini – endereço do primeiro nó da lista. Se a lista estiver vazia **ini** será igual a **NULL**.

No *fin – endereço do último nó da lista. Se a lista estiver vazia **fin** será igual a **NULL**.

No *corr – endereço do nó corrente da lista. Se a lista estiver vazia **corr** será igual a **NULL**. Antes de a função **prox()** (ou **ant()**) ser chamada pela primeira vez é necessário que o nó corrente seja *inicializado* por meio da função **posIni()** (ou **posFin()**).

Função main()

Use a função main() abaixo (copy and paste) para testar o seu TAD Lista.

```
#include <stdio.h>
#include <stdlib.h>
#include "lista.h"

void exibeLista(Lista *f) {
      int *aux;

      puts("=== Elementos da Lista ***");
      lst_posIni(f);
      aux=(int*)lst_prox(f);
      while(aux) {
            printf("%d\n",*aux);
            aux=(int*)lst_prox(f);
```

```
      }

}

int main(void) {
      Lista *f=lst_cria();
      int *a=(int*) malloc(sizeof(int));
      int *b=(int*) malloc(sizeof(int));
      int *c=(int*) malloc(sizeof(int));
      int *d=(int*) malloc(sizeof(int));
      int *aux;
      *a=10;
      *b=20;
      *c=30;
      *d=40;

      lst_insFin(f,a);
      lst_insFin(f,b);
      lst_insFin(f,c);
      lst_insFin(f,d);

      exibeLista(f);
      aux=lst_retIni(f);
      printf("*** Elemento Retirado %d ***\n",*aux);

      exibeLista(f);
      aux=lst_retIni(f);
      printf("*** Elemento Retirado %d ***\n",*aux);

      exibeLista(f);
      aux=lst_retFin(f);
      printf("*** Elemento Retirado %d ***\n",*aux);

      exibeLista(f);
      aux=lst_retFin(f);
      printf("*** Elemento Retirado %d ***\n",*aux);

      exibeLista(f);
      exibeLista(f);

      lst_libera(f);

      return 0;
}
```