INSTITUTO FEDERAL DA PARAÍBA
CAMPUS CAMPINA GRANDE
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO
DISCIPLINA DE ESTRUTURAS DE DADOS
PROF. VICTOR ANDRÉ PINHO DE OLIVEIRA

Estruturas de Dados

Atividade Prática 4 - Listas Lineares Sequenciais

Instruções

Responda às questões abaixo. Pode usar este próprio documento. Questões práticas devem ser anexadas separadamente.

Legenda:

Quirrell: Questões mais simples e diretas

Olho-Tonto: Questões de nível intermediário

Dolores: Questões de nível mais difícil, exigentes

Voldemort: Questões com nível de exigência altíssimo, desafiadoras

Questões

1. **Quirrell** Considerando os conceitos que trabalhamos em sala, responda:
    a. O que é uma Lista?
    b. O que é uma Lista Linear?
    c. O que é uma Lista Linear Sequencial?
    d. Qual a diferença entre uma Lista Linear não-Ordenada para um Ordenada?
2. **Olho-Tonto** Considerando o código da Lista Linear Sequencial não Ordenada presente no material, crie uma função inserir_ini que permite inserir um elemento no início da lista.
3. **Olho-Tonto** Considerando o código da Lista Linear Sequencial não Ordenada presente no material, faça:
    a. modifique a função inserir de modo que ela não permita a inserção de um novo elemento caso ele já exista na Lista
    b. modifique a função remover de modo que ela remova todas as ocorrências do elemento a ser removido
4. **Dolores** Considerando o código da Lista Linear Sequencial Ordenada presente no material, modifique-o de forma a termos uma Lista Linear Sequencial Ordenada de string (ordem lexicográfica ou alfabética). Considere o tamanho máximo para string de 20 caracteres (não esqueça do nulo). Obs.: Você pode usar as funções prontas da biblioteca <string.h>. De nada 🙂!
5. **ASSISTIDA** A forma como implementamos nossas Listas no material traz uma grande limitação: o nosso programa só pode manipular uma Lista por vez. E isso não é bom!!! Podemos resolver isso criando um registro LISTA, que contém as variáveis particulares necessárias para o controle de cada LISTA. Desse modo, basta adicionarmos um novo parâmetro às funções para que elas operem em cima da Lista passada como argumento. A partir do código abaixo, implemente uma versão melhorada de uma Lista Linear Sequencial Ordenada de inteiros.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
 unsigned MAX;
 int *arr, pos;
} LISTA;
```

```
void criar(LISTA *lst, int tam_MAX);
void apagar(LISTA *lst);

void inserir_ord(LISTA *lst, int elemento);
void remover(LISTA *lst, int elemento);
int buscar(LISTA *lst, int elemento);

int obter(LISTA *lst, int indice);
int tamanho(LISTA *lst);
void imprimir(LISTA *lst);
```

Note que, da forma como criamos o registro, cada LISTA terá seu próprio tamanho máximo. Ao invés do array, temos um ponteiro para inteiro. O array precisará ser alocado, conforme comentaremos abaixo.

Considere:

- a função **criar** vai alocar dinamicamente espaço para o campo arr do registro LISTA considerando o valor passado ao segundo parâmetro: tam_MAX.
- a função **apagar** deverá desalocar o espaço alocado pela função criar.
- as demais funções farão a mesma coisa conforme visto no material. Desta vez, no entanto, considerando o parâmetro LISTA *lst, que é passada como ponteiro para cada função.

Em essência, o código está praticamente pronto no material, à exceção das funções criar e remover. Você fará apenas as adequações necessárias para atender às novas especificações.

Para as questões a seguir, use como base a solução da questão ASSISTIDA (anterior), mas admita, para os propósitos das questões que seguem, que a lista possa inserir elementos de forma não Ordenada.

6. **Quirrell** Adicione uma função chamada *contar* que recebe um valor inteiro e devolve a quantidade de ocorrências deste valor na Lista.

```
int contar(LISTA *lst, int valor);
```

7. **Olho-Tonto** Adicione uma função chamada *inserirPos* que recebe uma posição *pos* e um *valor* inteiro a ser inserido na Lista. A função só deve inserir *valor* se *pos* for uma posição válida na Lista e se houver espaço.

```
void inserirPos(LISTA *lst, int pos, int valor);
```

8. **Dolores** Adicione uma função chamada *copiar* que recebe duas Listas e faz uma cópia da segunda na primeira. Cuidado com os ponteiros e a alocação de memória!

```
void copiar(LISTA *lst1, LISTA *lst2);
```

9. **Voldemort** Adicione uma função chamada *estender* que recebe duas Listas e estende, isto é, anexa uma uma cópia dos elementos da segunda a partir do fim da primeira. Avalie se a primeira Lista suporta anexar os elementos da segunda; caso contrário, a primeira deverá ser ajustada para caber todos os elementos.

```
void estender(LISTA *lst1, LISTA *lst2);
```