UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
http://www.icmc.usp.br

SCC-0202 Algoritmos e Estruturas de Dados I

2ª. Lista de Exercícios (Listas Seqüenciais)

Profa. Graça Nunes

1. Crie o Tipo Abstrato de Dados (TAD) lista linear ordenada (pelo campo chave), em C, de forma modular, tal que sua implementação seja sequencial, e inclua procedimentos para:
   (a) Verificar se uma lista está ordenada ou não (a ordem pode ser crescente ou decrescente);
   (b) Fazer uma cópia da lista L_1 em uma outra lista L_2;
   (c) Fazer uma cópia da lista L_1 em outra L_2, eliminando os elementos repetidos;
   (d) Inverter uma lista L_1 colocando o resultado em L_2;
   (e) Intercalar duas listas, L_1 e L_2, gerando uma lista L_3. Considere que L_1, L_2 e L_3 estão ordenadas;
   (f) Dada uma lista L_1, gerar uma lista L_2 onde cada registro contém dois campos de informação: *elem*, que contém um elemento de L_1 e *count*, que contém quantas vezes este elemento apareceu em L_1;
   (g) Assumindo que os elementos de uma lista L_1 são inteiros positivos, fornecer os elementos que aparecem o maior e o menor número de vezes (forneça os elementos e o número de vezes correspondente).

2. Seja uma TAD **Dicionário,** com as seguintes operações associadas:
   (a) Criação de um dicionário vazio.
   (b) Inserção de uma nova entrada.
   (c) Remoção de uma entrada no dicionário.
   (d) Pesquisa de traduções de palavras em inglês.
   (e) Alteração de uma entrada.
   (f) Gravação do dicionário em um arquivo.
   (g) Reinicialização do dicionário.
   (h) Inicialização do dicionário a partir de um arquivo.
   (i) Testes de dic_cheio? dic_vazio? (rotinas auxiliares)

Implementar um Dicionário Inglês/Português com um dos 2 tipos de dados abaixo, em C de forma modular:

(a)

```
#define MAX_ENT 100

typedef struct{
    char pal_ing[15];
    char pal_port[15];
}Entrada;

struct{
    Entrada dic[MAX_ENT];
    int tam;
}Dicionario;
```

(b)

```
#define MAX_ENT 100
#define MAX_TRAD 10

typedef struct{
    char pal_ing[15];
    char trad [MAX_TRAD][15];
    int num_trad;
}Entrada;

struct {
    Entrada dic[MAX_ENT];
    int tam;
}Dicionario;
```

3. Considere as seguintes declarações:

```
#define n 100

int a[n];
int ult;
```

Considere também que no array acima está seqüencialmente armazenada uma lista ordenada, cujo último elemento é apontado por `ult`. Declare, em C, as seguintes operações:

(a) *insere* (v, a) – dado o inteiro v, ele deve inseri-lo na lista a, caso ele já não esteja lá. Se já estiver, nada faz.

(b) *elimina* (v, a) – elimina o registro com valor v da lista a, caso ele esteja lá. Se não estiver, imprime uma mensagem.

(c) *busca* (v, a) – verifica se o valor v pertence a lista, retornando o valor do índice do array onde ele está, neste caso, ou 0, caso contrário. Faça duas versões: uma com sentinela e outra sem sentinela.

4. Seja L = (a_1, a_2, a_3, ..., a_n) uma lista linear representada num array V[n]. Usando o mapeamento o i-ésimo elemento de L é armazenado em V[i-1].

Escreva um algoritmo para reverter a ordem dos elementos em V, isto é, o algoritmo deve transformar V tal que V[i] contenha o elemento *n* – i de L. O único espaço adicional disponível para seu algoritmo é suficiente para apenas uma variável simples. A entrada para seu algoritmo é V e *n*.