Instituto de Computação
Curso de Sistemas de Informação
Estruturas de Dados (TCC00171)

Lista de Exercícios 2

Nas questões 1 a 14, considere listas encadeadas que armazenam valores inteiros, nas quais cada elemento é do tipo *Elemento*, descrito abaixo:

```
struct elemento {
   int info;                /*  Valor inteiro armazenado              */
   struct elemento *prox;   /*  Ponteiro para o próximo elemento      */
};
typedef struct elemento Elemento;
```

1) Escreva uma função que remove o *n*-ésimo elemento de uma lista. A função recebe como parâmetros o ponteiro *lst* para o início da lista e o inteiro *n*, e deve retirar da lista, liberando o espaço alocado na memória, o elemento localizado na *n*-ésima posição, contando a partir do início da lista. A função retorna o ponteiro para o início da lista atualizado. Se a lista tiver menos de *n* elementos, nada deve ser feito. O protótipo da função é:

```
Elemento* remove_elemento_n(Elemento* lst, int n);
```

2) Escreva uma função RECURSIVA que remove os *n* primeiros elementos de uma lista. A função recebe como parâmetros o ponteiro *lst* para o início da lista e o inteiro *n*, indicando a quantidade de elementos que devem ser retirados do início da lista, e retorna o ponteiro para o início da lista atualizado. O espaço correspondente na memória deve ser liberado. Se a lista tiver *n* elementos ou menos, toda ela deve ser liberada e a função deve retornar NULL. O protótipo da função é:

```
Elemento* remove_n_elementos(Elemento* lst, int n);
```

3) Escreva uma função que verifica se uma lista armazena algum valor repetido. A função recebe como parâmetro o ponteiro *lst* para o início da lista e deve retornar 1, se há valores repetidos na lista, ou 0, caso contrário. O protótipo da função é:

```
int valores_repetidos(Elemento* lst);
```

4) Escreva uma função que recebe como parâmetros o ponteiro *lst*, para o primeiro elemento de uma lista, e os valores *min* e *max*, e remove e libera todos os elementos desta lista que armazenam valores menores que *min* ou maiores que *max*. A função deve retornar o ponteiro para o início da lista, que pode ter sido modificado, e tem o seguinte protótipo:

```
Elemento* filtra(Elemento *lst, int min, int max);
```

5) Escreva uma função RECURSIVA que remove de uma lista todos os elementos que são maiores que um dado valor. A função recebe como parâmetros o ponteiro *lst* para o início da lista e o valor real *x*, e deve retirar da lista, liberando o espaço alocado na memória, todos os elementos que contêm valores maiores que *x*. A função retorna o ponteiro para o início da lista atualizado. O protótipo da função é:

```
Elemento* remove_maiores(Elemento* lst, int x);
```

6) Escreva uma função que cria uma cópia de uma lista encadeada, ou seja, uma nova lista encadeada cujo conteúdo armazenado é idêntico – contém os mesmos valores e na mesma ordem – ao da lista original. A função recebe como parâmetro o ponteiro para o primeiro elemento da lista original (àquela que será copiada) e retorna um ponteiro para a nova lista (a cópia). A lista original não deve ser alterada. O protótipo da função é:

```
Elemento* copia(Elemento* lst);
```

7) Refaça o exercício anterior criando uma função RECURSIVA. O protótipo da função é:

```
Elemento* copia_rec(Elemento *lst);
```

8) Escreva uma função que inverte a ordem dos elementos de uma lista. A função recebe como parâmetro o ponteiro *lst* para o início da lista e deve invertê-la, de tal forma que o último elemento passe a ser o primeiro, e assim por diante, retornando o novo ponteiro para o início da lista. O protótipo da função é:

```
Elemento* inverte_lista(Elemento *lst);
```

9) Escreva uma função que concatena duas listas, anexando a segunda lista ao final da primeira. A função recebe como parâmetros os ponteiros *lst1* e *lst2* para o início das listas e deve fazer o último elemento da primeira lista apontar para o primeiro elemento da segunda lista. Se a primeira lista for vazia, o ponteiro para a primeira lista deve ser atualizado para apontar para a segunda lista. Se a segunda lista for vazia, nada deve ser feito. O protótipo da função é:

```
Elemento* concatena_listas(Elemento *lst1, Elemento *lst2);
```

10) Escreva uma função que intercala duas listas cujos elementos estão dispostos em ordem crescente, formando uma nova lista também em ordem crescente. A função recebe como parâmetros os ponteiros *lst1* e *lst2* para o início das listas e deve retornar o ponteiro para o início da nova lista, contendo todos os elementos das listas originais, em ordem crescente. O protótipo da função é:

```
Elemento* intercala_listas(Elemento *lst1, Elemento *lst2);
```

11) Refaça o exercício anterior criando uma função RECURSIVA. O protótipo da função é:

```
Elemento* intercala_listas_rec(Elemento *lst1, Elemento *lst2);
```

12) Escreva uma função RECURSIVA que busca um valor em uma lista que não possui valores repetidos. A função recebe como parâmetros o ponteiro *lst* para o início da lista e o inteiro *x*, que é o valor procurado, e retorna o ponteiro para o elemento da lista que contém este valor. Se nenhum elemento com este valor for encontrado, a função retorna NULL. O protótipo da função é:

```
Elemento* busca_rec(Elemento* lst, float x);
```

13) Escreva uma função RECURSIVA que percorre duas listas e retorna o maior valor armazenado em qualquer uma delas. A função recebe como parâmetros os ponteiro *lst1* e *lst2* para o início das listas e retorna o maior valor encontrado em qualquer uma das listas. Considere que as listas contêm apenas valores positivos e o maior valor contido em uma lista vazia é 0. O protótipo da função é:

```
int maior_valor_rec(Elemento* lst1, Elemento* lst2);
```

14) Escreva uma função RECURSIVA que compara duas listas L1 e L2 e verifica se a segunda é subconjunto da primeira, isto é, todos os elementos de L2 estão também em L1. As listas estão em ordem crescente e não têm valores repetidos. A função recebe como parâmetros os ponteiro *lst1* e *lst2* para o início das listas e retorna 1 se L2 é subconjunto de L1, e 0, caso contrário. Considere que uma lista vazia é subconjunto de qualquer lista. O protótipo da função é:

```
int compara_ordenado_rec(Elemento* L1, Elemento* L2);
```

15) Os alunos de uma disciplina estão organizados em uma lista encadeada, onde cada elemento da lista é definido pela estrutura a seguir:

```
struct elemento {
   Aluno* info;
   struct elemento* prox;
};
typedef struct elemento Elemento;
```

O conteúdo de cada elemento da lista é um ponteiro para o tipo estruturado *Aluno*, que descreve alunos de um determinado curso, definido a seguir:

```
struct aluno {
   long int mat;
   char turma;
   float p1, p2, p3;
};
typedef struct aluno Aluno;
```

Implemente uma função que retorna uma nova lista contendo somente os alunos aprovados que pertencem a uma determinada turma. Para ser aprovado, o aluno deve ter a média aritmética das notas da P1, P2 e P3 maior ou igual a 5. Na nova lista, os alunos devem aparecer na mesma ordem relativa em que aparecem na lista original. A lista original não pode ser alterada. A função recebe como parâmetros o ponteiro *lst*, para o início da lista, e o caractere *turma*, que indica a turma desejada, e deve retornar o ponteiro para o início da nova lista, obedecendo ao seguinte protótipo:

```
Elemento* lst_aprovados(Elemento* lst, char turma);
```