# ALGORITMOS II

## 10ª LISTA DE EXERCÍCIOS
## LISTA LINEAR SIMPLESMENTE ENCADEADA

**1** Quais as vantagens e desvantagens de representar um conjunto de dados como um vetor ou numa lista encadeada?

**2** Escreva um programa para ler até 30 nomes, em qualquer ordem, e apresentá-los em ordem alfabética. Para maior eficiência, ao invés de usar um vetor de cadeia de caracteres (*strings*), use um vetor de ponteiros, alocando-as dinamicamente conforme a necessidade.

**3** Verifique a finalidade dos programas:

a)

```
typedef struct reg *no;
struct reg {
  char nome[30];
  int numero;
  struct reg *prox;
};

int main () {
  no lista, p;
  int i;
  lista = NULL;
  for (i = 1; i <=3; i++) {
     p = (no) malloc(sizeof(struct reg));
     printf ("\nNome: ");
     fflush (stdin);
     gets (p->nome);
     printf ("\nNumero: ");
     scanf ("%d",&(p->numero));
     p->prox = lista;
     lista = p;
  }
  while (p != NULL)  {
     printf ("\n%s   %d",p->nome,p->numero);
     p = p->prox;
  }
}
```

b)

```
typedef struct reg *no;
struct reg {
  char nome[30];
  int numero;
  struct reg *prox;
};
```

```
int main () {
  no lista, p;
  int i;
  lista = NULL;
  do {
     p = (no) malloc(sizeof(struct reg));
     printf ("\nNome: ");
     fflush (stdin);
     gets (p->nome);
     printf ("\nNumero: ");
     scanf ("%d",&(p->numero));
     p->prox = NULL;
     lista = p;
  } while (p->numero !=100);
  while (p != NULL) {
     printf ("\n%s   %d",p->nome,p->numero);
     p = p->prox;
  }
}
```

**4** Qual o comando não é necessário no trecho de programa abaixo?

```
typedef struct reg *no;
struct reg {
  int info;
  struct reg *prox;
};

no lista;

void mostra_lista (no *lista) {
  if (*lista == NULL) {
      printf ("\nLista vazia");
      return;
  }
  no p = (no)malloc(sizeof(struct reg));
  p = *lista;
  printf ("\nElementos da lista: ");
  do {
    printf ("%d ",p->info);
    p = p->prox;
  } while (p != NULL);
}
```

**5** Existe algum erro no trecho de programa que segue? Qual?

```
void MostraLista (no lista) {
  no p = lista;
  while (p != NULL) {
     p = (no) malloc(sizeof(struct reg));
     printf ("%d\n",p->dado);
     p = p->prox;
  }
}
```

**6** Explique o que acontece nas atribuições abaixo (dica: use desenhos):

a) `p->prox = q;` b) `p->prox = q->prox;` c) `p->info = q->info;`
d) `p = q;` e) `p->prox = NULL;` f) `p = p->prox;`
g) `p = (p->prox)->prox;` h) `p->prox = p;`

**7** *Comprimento da Lista:* Implemente uma função que retorne o comprimento (número de elementos) da lista linear simplesmente encadeada.

**8** Escreva uma função que retorne o conteúdo do primeiro nó de uma lista linear simplesmente encadeada.

**9** Escreva uma função que retorne o conteúdo do último nó de uma lista linear simplesmente encadeada.

**10** *Busca na Lista*: Implemente uma função que busque um valor específico na lista e retorne a posição do nó onde o valor foi encontrado. Se o valor não for encontrado, a função deve retornar uma indicação de erro.

**11** Escreva uma função que retorne o número de vezes que um determinado elemento ocorre numa lista linear simplesmente encadeada.

**12** Escreva uma função para inserir um elemento no início de uma lista linear simplesmente encadeada.

**13** Escreva uma função para inserir um elemento no final de uma lista linear simplesmente encadeada.

**14** Escreva uma função para inserir um elemento depois do n-ésimo elemento de uma lista linear simplesmente encadeada.

**15** Escreva uma função para inserir um elemento antes do n-ésimo elemento de uma lista linear simplesmente encadeada.

**16** Escreva uma função para remover o primeiro elemento de uma lista linear simplesmente encadeada.

**17** Escreva uma função para remover o último elemento de uma lista linear simplesmente encadeada.

**18** Escreva uma função para remover o n-ésimo elemento de uma lista linear simplesmente encadeada.

**19** Dada uma lista linear simplesmente encadeada e um elemento, escreva uma função que remova da lista todas as ocorrências do elemento.

**20** Escreva uma função para excluir todos os nós de uma lista linear simplesmente encadeada.

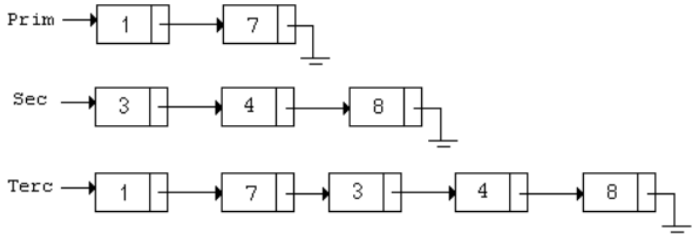
**21** Desenvolva uma função que permita a inserção ordenada (crescente) de elementos em uma lista linear simplesmente encadeada.

**22** Desenvolva uma função que permita a remoção de um elemento em uma lista ordenada linear simplesmente encadeada.

**23** Desenvolva uma função que remova de uma lista linear ordenada todas as ocorrências de um determinado elemento.

**24** *Inversão da Lista*: Implemente uma função que inverta a ordem dos elementos da lista. A função deve reorganizar os ponteiros de cada nó para que a lista seja invertida sem criar uma nova lista.

**25** *Concatenar duas Listas*: Implemente uma função que receba duas listas lineares simplesmente encadeadas e retorne uma nova lista que seja a concatenação das duas.




**26** Escreva uma função para retornar o endereço e o conteúdo do último nó de uma lista simplesmente encadeada.

**27** *Soma dos elementos*: Escreva uma função que retorne a soma dos elementos de uma lista linear simplesmente encadeada.

**28** *Listas idênticas:* Escreva uma função que informe se as duas listas lineares simplesmente encadeada são idênticas.

**29** *Clonagem de lista*: Implemente uma função que receba uma lista encadeada e retorne uma nova lista encadeada que seja uma cópia exata da lista original, sem compartilhamento de nós entre as duas listas.

**30** Escreva uma função que gere uma lista linear simplesmente encadeada L2 onde cada registro contém dois campos de informação: `dado` contém um elemento de L1, e `qte` contém quantas vezes este elemento apareceu em L1.

**31** Desenvolva uma função, que dado uma lista linear simplesmente encadeada de números inteiros positivos, forneça os elementos que aparecem o maior e o menor número de vezes (a função deve informar ambos: os elementos e o número de vezes).

**32** *Remoção de Elementos Duplicados*: Implemente uma função que percorra a lista e remova todos os elementos duplicados, mantendo apenas a primeira ocorrência de cada valor.

**33** Faça uma função que verifique se uma lista linear simplesmente encadeada está ordenada ou não (a ordem pode ser crescente ou decrescente).

**34** Dada duas listas ordenadas L1 e L2, escreva uma função que combine L1 e L2 em uma única lista ordenada L3. Após a criação de L3, L1 e L2 devem estar vazias (*nulas*).

**35** *Intercalação*: Escreva uma função que mescle duas listas lineares simplesmente encadeadas, alternando os nós de cada lista.

**36** *Dividir uma Lista ao Meio*: Implemente uma função que divida uma lista ao meio, criando duas listas a partir de uma lista original. Se o número de elementos for ímpar, a primeira lista deve ter um elemento a mais. Utilize o conceito de lista linear simplesmente encadeada.

**37** *Média dos Elementos*: Supondo que a lista linear simplesmente encadeada contenha apenas números inteiros, implemente uma função que calcule a média dos valores dos elementos da lista.

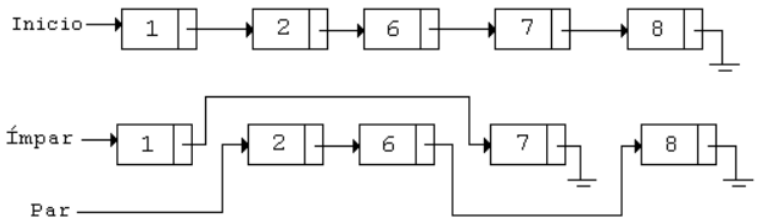
**38** *Verificação de Palíndromo:* Implemente uma função para verificar se os elementos de uma lista encadeada formam um palíndromo. A lista deve ser lida de trás para frente e comparada com sua leitura normal.

**39** *Interseção de duas Listas:* Implemente uma função que receba duas listas encadeadas como entrada e retorne uma nova lista contendo a *interseção* das duas listas (elementos comuns entre elas).

**40** *Busca por Valor Mínimo e Máximo*: Implemente funções para buscar o menor e o maior valor presentes na lista linear simplesmente encadeada**.**

**41** Dada uma lista encadeada que armazena números inteiros escreva uma função que transforma a lista dada em duas listas encadeadas: a primeira contendo os elementos cujo conteúdo é par e a segunda contendo os elementos com conteúdos impares. Sua função deve manipular somente os apontadores e **não** o conteúdo das células.

Exemplo:




**42** *Manipulação de polinômios:* Deseja-se manipular polinômios do tipo $p(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$. Tais polinômios podem ser representados por listas lineares simplesmente encadeadas onde cada nó da lista possui três campos: um para o coeficiente que é um número real, um para o expoente que é um número inteiro e um campo que armazena um ponteiro para o próximo nó. Escreva programas para: ler um polinômio e armazená-lo na lista, somar dois polinômios, multiplicar dois polinômios e derivar um polinômio.