Listas Contíguas

1. Considerando o TAD `ListaCont` que representa uma lista contígua, faça uma função para trocar de posição dois elementos na lista. A função deve retornar se a operação foi possível ou não.

```
bool ListaCont::troca(int posicao1, int posicao2);
```

2. Considerando o TAD `ListaCont` que representa uma lista contígua, faça uma função para aumentar a capacidade máxima (definida por `max`) do vetor que representa a lista sem que nenhum dado atualmente representado na lista seja perdido. Se a nova capacidade definida pelo parâmetro `novoMax` for menor que a capacidade atual, a operação não deve fazer nada. Ao final retornar se a operação foi ou não realizada.

```
bool ListaCont::aumentaCapacidade(int novoMax);
```

Listas Encadeadas

3. Considerando o TAD `ListaEncad` (lista simplesmente encadeada), implementar uma operação do TAD que tenha como valor de retorno o comprimento da lista, isto é, que calcule o número de nós de uma lista.

```
int ListaEncad::comprimento();
```

4. Considerando o TAD `ListaEncad` que representa uma lista simplesmente encadeada de valores inteiros, implementar uma função para retornar o números de nós da lista que possuem o campo `info` maior que o valor inteiro `x`.

```
int ListaEncad::maiores(int x);
```

5. Criar um TAD `ListaEncad` como sendo uma lista simplesmente encadeada de números inteiros de forma que todos os nós da lista estejam em ordem crescente pelo valor do nó. Basta aproveitar o TAD para lista encadeada estudado em aula e alterar a função de inserção de forma que esta sempre percorra os elementos da lista a fim de encontrar a posição correta do novo nó.

```
void ListaEncad::insereOrdenado(int valor);
```

6. Uma lista circular é uma lista cujo último elemento tem como próximo o primeiro elemento da lista, formando assim um ciclo. Não existe conceitualmente o primeiro e o último nó. Para percorrer todos os elementos da lista circular, começamos de um nó e visitamos todos os elementos até alcançar novamente esse novo elemento. Desenvolver todo o TAD `ListaCirc` com suas operações usuais.

7. O TAD para lista simplesmente encadeada possui algumas limitações, como por exemplo, a impossibilidade de percorrer eficientemente os nós do final para o início, assim como a dificuldade de remover um nó, dado um ponteiro para este, já que para isso é necessário percorrer a lista até encontrar o nó desejado para descobrir o seu anterior a fim de acertar o encadeamento

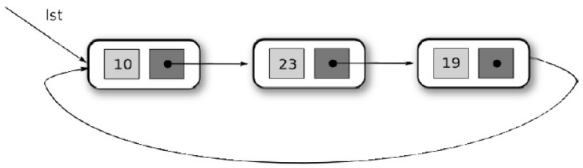

Figura 1: Exemplo de lista circular.

na remoção. Outra implementação de lista é aquela na qual o nó guarda a informação, um ponteiro para o próximo e um ponteiro para o anterior. Um exemplo pode ser visto na figura abaixo. A seguir apresentamos a estrutura que deve ser utilizada e os protótipos das operações

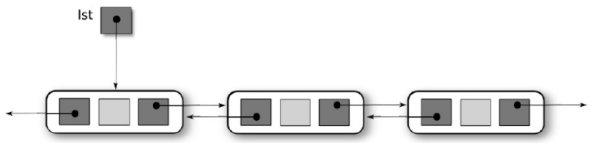

Figura 2: Exemplo de lista duplamente encadeada.

a serem implementadas.

```
class NoDuplo
{
 public:
  NoDuplo();
  ~NoDuplo();
  void setAnt(NoDuplo* p);
  void setProx(NoDuplo* p);
  void setInfo(int val);
  NoDuplo* getAnt();
  NoDuplo* getProx();
  int getInfo();

 private:
  int info;      // informacao
  NoDuplo* ant;  // ponteiro para anterior
  NoDuplo* prox; // ponteiro para proximo
};

class ListaDupla
{
 public:
  ListaDupla();
  ~ListaDupla();
  bool busca(int val);
  void insereInicio(int val);
  void eliminaInicio();
  void insereFinal(int val);
  void eliminaFinal();
```

```
 private:
  int n;
  NoDuplo* primeiro;
  NoDuplo* ultimo;
};
```

8. Implemente uma operação do TAD `ListaEncad` que verifique se duas listas encadeadas são iguais e retorne **true** caso sejam e **false** caso contrário. Duas listas são consideradas iguais se têm a mesma sequência de elementos.

```
bool ListaEncad::igual(ListaEncad* l2);
```

9. Um polígono geométrico pode ser considerado como uma lista de vértices. Para desenhar tal polígono, basta traçar uma reta ligando vértices adjacentes. Projete um TAD para armazenar esse polígono (defina as operações necessárias). Considere a necessidade de desenhar o polígono, eliminar arestas e incluir novas arestas.

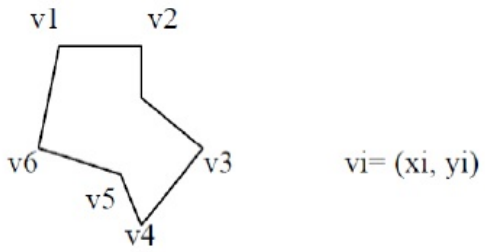

Figura 3: Exemplo de polígono.

10. Considere o TAD `ListaEncad` que representa uma lista simplesmente encadeada com elementos repetidos. Implemente uma operação para eliminar todas as ocorrências do valor `v` da lista.

```
void ListaEncad::eliminaValor(int v);
```

11. Implementar uma lista encadeada para manipulação de polinômios do tipo: $a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x^1 + a_0$, onde cada elemento i da lista deve armazenar o i-ésimo termo do polinômio (diferente de 0). Cada item da lista deve conter o valor da potência (inteiro) e o coeficiente correspondente (real).

```
class Polinomio
{
 public:
  Polinomio(int tam);
  ~Polinomio();
  void inserirTermo(int c, int p);
  void imprimir();
  void eliminaTermo(int i);

 private:
  No * primeiro;
}
```

Exercícios de Pilhas e Filas

12. Pilha com Vetor(contígua). Implementar um TAD Pilha utilizando a representação vetorial, criando as operações especificadas nos protótipos a seguir:

```
class PilhaCont
{
 private:
  int max;  // capacidade maxima.
  int topo; // posicao do topo da pilha.
  int *x;   // vetor que armazena os dados da pilha.

 public:
  PilhaCont(int tam);
  ~PilhaCont();
  void empilha(int val);  // insere No no topo.
  int desempilha();       // elimina No do topo.
  int consulta();         // consulta o topo.
  bool pilhaVazia();
};
```

13. Fila com Vetor (contígua). Implementar um TAD Fila utilizando a representação vetorial, criando as operações especificadas nos protótipos a seguir:

```
class FilaCont
{
 private:
  int m;        // maximos de elemento do vetor
  int c;        // indice do No que esta no incio
  int f;        // indice do No que esta no fim
  int *x;       // vetor que armazena a fila

 public:
  FilaCont(int tam);
  ~FilaCont();
  int consultaInicio(); // consulta No do inicio
  void entra(int val);  // insere No no fim
  void sai();           // elimina No do inicio
  bool vazia();         // verifica se esta vazia
};
```

14. Na representação de Fila e Pilha contígua (com vetor) pode acontecer de a pilha ou a fila estar cheia no momento de inserir novos valores (nós). Fazer um procedimento, no bloco private, para realocar o vetor que suporta a pilha ou a fila com o dobro da capacidade que havia anteriormente. Refazer as operações para levar em conta essa nova capacidade de realocamento. Atenção: será preciso alocar o novo vetor e em seguida copiar todos os dados do antigo vetor para o novo e, somente, após essa cópia desalocar o antigo.

15. Desenvolva uma função para comparar se uma Fila `f1` tem mais elementos do que uma fila `f2`. Deve retornar -1 se `f1` $<$ `f2`, 0 se `f1` $=$ `f2` e 1 se `f1` $>$ `f2`. Considere que a `FilaEncad` possui um descritor com a informação de quantos nós a fila possui e uma função de acesso a esse descritor. Caso não esteja disponível, implemente o descritor e a função de acesso.

```
class FilaEncad
{
 public:
  // ...
  int getNumNos();

 private:
  // ...
  int numNos;
};
```

Em seguida implemente a função:

```
int compara(FilaEncad* f1, FilaEncad* f2);
```

16. Considere a existência de um TAD PilhaEncad de números reais, cujas operações são:

```
class PilhaEncad
{
  PilhaEncad();
  ~PilhaEncad();
  float consulta();
  void empilha(float v);
  float desempilha();
  bool vazia();
}
```

Sem conhecer a representação interna desse tipo abstrato:

a) Implemente uma função que receba uma pilha como parâmetro e retorne o valor armazenado em seu topo, restaurando o conteúdo da pilha.

b) Implemente uma função que receba duas pilhas, p1 e p2, e passe todos os elementos da pilha p2 para o topo da pilha p1. A figura abaixo ilustra essa concatenação de pilha.

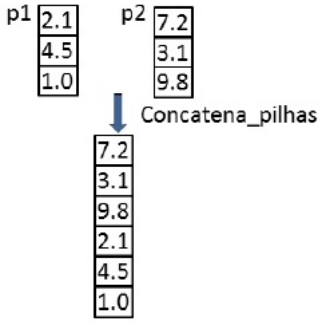

Figura 4: Concatenação de pilhas.

c) Implemente uma função que receba uma pilha p como parâmetro e retorne como resultado uma cópia dessa pilha. Ao final, a pilha p recebida como parâmetro deve ter seu conteúdo original inalterado.

17. Escreva operações do TAD `ListaEncad` para (i) copiar um vetor com `n` itens para uma lista encadeada e para (ii) copia a lista encadeada para um vetor.

```
ListaEncad* ListaEncad::vetor2Lista(int n, int *vet);
int* ListaEncad::lista2Vetor(ListaEncad* l);
```

A função que cria uma lista a partir de um vetor deve alocar dinamicamente uma lista, preenchê-la com os dados do vetor e retornar o ponteiro para a lista crida. Por outro lado, a função que cria um vetor a partir de uma lista deve alocar dinamicamente um vetor (com capacidade igual ao número de nós da lista), copiar a lista para o vetor e retornar o endereço deste.

18. Desenvolver um procedimento para ler 300 valores inteiros e montar um pilha representada por encadeamento, atendendo às seguintes características:

    - Primeiro valor lido será incluído como primeiro nó da pilha;
    - A partir do segundo valor lido, ele só será incluído se for maior do que o nó que está no topo da pilha;

    Dica: use o TAD pilha com lista encadeada usado em sala de aula e disponível na página do curso.

19. Dada uma fila cujos valores de seus nós são inteiros quaisquer, desenvolver um procedimento para esvaziar esta fila, gerando uma nova fila que contenha somente os nós positivos da anterior.

20. Usando obrigatoriamente o TAD `PilhaCont`, implementar uma função para verificar se a pilha de inteiros passada como parâmetro é palíndromo, ou seja, é a mesma quando lida da esquerda para a direita ou da direita para a esquerda. Caso isso aconteça retornar **true**, caso contrário retornar **false**. Exemplo: 1,2,3,4,5,4,3,2,1 é palíndromo, 1,2,3,6,7,8 não é palíndromo.

```
bool ePalindromo(int n, PilhaCont* p);
```

21. Escreva um algoritmo, usando uma pilha, que inverte as letras de cada palavra de um texto terminado por "."preservando a ordem das palavras. Por exemplo, dado o texto:

    `ESTE EXERCÍCIO É MUITO FÁCIL.`

    A saída deve ser:

    `ETSE OICÍCREXE É OTIUM LICÁF`

22. A conversão de números inteiros, na base 10, para outras bases numéricas se dá através de sucessivas divisões do número n pelo valor da base na qual se queira converter. Faça um programa para obter a conversão numérica, de acordo com a opção do usuário. Utilize uma estrutura de dados do tipo pilha.

    (a) Decimal para Binário (base 2);
    (b) Decimal para Octal (base 8);
    (c) Decimal para Hexadecimal (base 16).