# Estruturas de Dados

## Exercícios Resolvidos – Listas Encadeadas

**1)** Crie uma lista encadeada para armazenar valores reais (cada chave será um número real). A lista deve possuir um método para adicionar um elemento no início da lista e outro para adicionar no final. A lista também deve possuir um método para remover um elemento no início da lista e outro para remover um elemento no final da lista.

**2)** Na lista encadeada criada no exercício anterior (exercício 1), crie o método (AdicionarVariosNoFinal) que recebe uma lista encadeada e concatena-a à lista atual, adicionando os elementos da lista passada por parâmetro no final da lista atual.

**3)** Na lista encadeada criada no exercício 1, crie o método (AdicionarVariosNoInicio) que recebe uma lista encadeada e concatena-a à lista atual, adicionando os elementos da lista passada por parâmetro no início da lista atual.

**4)** Na lista encadeada criada no exercício 1, crie o método Ordenar que ordena os elementos da lista em ordem crescente de seus valores.

Respostas na próxima página

## Questão 1)

```
internal class No
{
    public double Valor { get; set; }
    public No? Proximo { get; set; }
}

internal class ListaSimplesmenteEncadeada
{
    public void AdicionarNoInicio(double valor)
    {
        var novoNo = new No { Valor = valor };

        if (primeiro == null)
            primeiro = ultimo = novoNo;
        else
        {
            novoNo.Proximo = primeiro;
            primeiro = novoNo;
        }
    }

    public void AdicionarNoFinal(double valor)
    {
        var novoNo = new No { Valor = valor };

        if (ultimo == null)
            primeiro = ultimo = novoNo;
        else
        {
            ultimo.Proximo = novoNo;
            ultimo = novoNo;
        }
    }

    public void RemoverNoInicio()
    {
        if (primeiro == null)
            throw new Exception("Não é possível remover elementos de uma lista
vazia.");

        if (primeiro == ultimo)
            primeiro = ultimo = null;
        else
            primeiro = primeiro.Proximo;
    }
```

```
    public void RemoverNoFinal()
    {
        if (primeiro == null)
            throw new Exception("Não é possível remover elementos de uma lista
vazia.");

        if (primeiro == ultimo)
            primeiro = ultimo = null;
        else
        {
            var penultimo = primeiro;

            while (penultimo?.Proximo?.Proximo != null)
                penultimo = penultimo.Proximo;

            if (penultimo != null)
            {
                penultimo.Proximo = null;
                ultimo = penultimo;
            }
        }
    }

    public void ParaCada(Action<double> acao)
    {
        var atual = primeiro;

        while (atual != null)
        {
            acao(atual.Valor);
            atual = atual.Proximo;
        }
    }

    private No? primeiro = null;
    private No? ultimo = null;
}
```

## Questão 2)

```
//Adicione o código abaixo à classe ListaSimplesmenteEncadeada:

    public void AdicionarVariosNoFinal(ListaSimplesmenteEncadeada lista)
    {
        lista.ParaCada(x => AdicionarNoFinal(x));
    }
```

## Questão 3)

```
//Adicione o código abaixo à classe ListaSimplesmenteEncadeada:

    public void AdicionarVariosNoInicio(ListaSimplesmenteEncadeada lista)
    {
        var listaAux = new ListaSimplesmenteEncadeada();

        //Guarda os elementos em ordem inversa em uma lista auxiliar.
        lista.ParaCada(x => listaAux.AdicionarNoInicio(x));

        //Passa os elementos da lista auxiliar para o início desta lista
        listaAux.ParaCada(x => AdicionarNoInicio(x));

        //Ao término deste método, os elementos da lista passada por parâmetro
        //serão adicionados no início da lista guardada em this, sendo que a
        //ordem em que estes elementos estavam na lista original será mantida.
    }
```

## Questão 4)

```
//Adicione o código da próxima página à classe ListaSimplesmenteEncadeada:
```

```
/// <summary>
/// Ordena a lista pelo método bolha
/// </summary>
public void Ordenar()
{
    if (primeiro == null)
        return;

    No atual;
    bool houveTroca;

    do
    {
        houveTroca = false;
        atual = primeiro;

        while (atual.Proximo != null)
        {
            if (atual.Valor > atual.Proximo.Valor)
            {
                TrocarValor(atual, atual.Proximo);
                houveTroca = true;
            }

            atual = atual.Proximo;
        }
    } while (houveTroca);
}

private void TrocarValor(No no1, No no2)
{
    var aux = no1.Valor;
    no1.Valor = no2.Valor;
    no2.Valor = aux;
}
```