Documentação - Lista de Exercícios

Integrantes do Grupo: Rafael dos Santos, João Alexandre, Caio Voitena

---

Exercício 1: [ValidadorDeSenha.cs]

Descrição do Problema

O problema em questão trabalha com a validação de senhas, em primeiro deve se pedir que o usuário envie uma senha após isso o programa verifica se a senha atende determinados requisitos, sendo eles:

- Pelo menos 8 caracteres
- Pelo menos uma letra maiuscula
- Pelo menos um número
- Pelo menos um caractere especial (como `!`, `@`, `#`, etc.)

Passo a Passo Lógico da Resolução

1) **Leitura da senha:** A primeira parte do código é a leitura da senha digitada pelo usuário. Para isso, usamos o comando `Console.ReadLine()` e salvamos o valor digitado na variável do tipo `string` chamada *senha*.

2) **Verificação do tamanho da senha:** Logo após a leitura, a primeira validação feita é sobre o tamanho da senha. Criamos uma variável do tipo *bool* chamada `temMinimo`, que recebe o valor da expressão `senha.Length >= 8`. Essa verificação retorna `true` ou `false`, dependendo do comprimento da senha.

3) Após isso começamos a desenvolver o resto do nosso código como visto a primeira verificação é o tamanho da senha e para isso será criado uma variável booleana ou bool apenas e o valor que ela receberá será dado por “senha.Length >= 8” O resultado dessa verificação podera ser “True ou False” e isso é armazenado na variável booleana `temMinimo`.

4) **Criação das outras variáveis:** Agora iremos declarar outras três variáveis booleanas: `temMaiuscula` que verifica se existe **pelo menos uma letra maiuscula,** `temNumero` que verifica se existe **pelo menos um número,** `temEspecial` que verifica se existe **pelo menos um caractere especial,** Todas essas variáveis começam com o valor `false`, pois ainda não fizemos nenhuma verificação.

5) **Verificação senhas**: utilizamos uma estrutura de repetição `foreach (char c in senha)` para analisar **cada caractere da senha individualmente**. Dentro desse laço, fazemos três verificações:
    a) A primeira `char.IsUpper(c)` se o caractere for uma letra maiuscula,`temMaiuscula` vira `true.`
    b) A segunda `char.IsDigit(c)` se o caractere for um número, `temNumero` vira `true.`
    c) A terceira `!char.IsLetterOrDigit(c)` se o caractere não for letra nem número, consideramos especial, então `temEspecial` vira `true.`
    d) Esse laço continua até o último caractere da senha. Mesmo que todas as condições já tenham sido satisfeitas antes do fim, o laço percorre toda a senha.

6) **Verificação final:** Fora do laço `foreach`, fazemos uma verificação final com a estrutura `if (temMinimo && temMaiuscula && temNumero && temEspecial)`. Essa condição só será verdadeira se todas as quatro variáveis booleanas forem `true`, ou seja, se a senha atender todos os requisitos definidos.

7) **Saída no console:** No final do código, usamos uma estrutura `if` para verificar se todas as condições foram satisfeitas. Se forem verdadeiras, é exibida a mensagem "Senha forte!" se caso alguma das condições não seja satisfeita, o programa imprime "Senha fraca!" e exibe exatamente quais critérios não foram cumpridos, um por um.

Estruturas e Comandos Utilizados

- **Leitura de dados:**
  `Console.ReadLine()` foi utilizado para capturar a senha digitada pelo usuário. É o comando padrão para entrada de texto no console.

- **Variáveis booleanas (bool):**
  Foram usadas para armazenar o resultado de cada verificação (tamanho mínimo, presença de maiúscula, número e caractere especial). Isso facilitou o controle e a leitura do código.

- **Estrutura de repetição – `foreach`:**
  Utilizada para percorrer cada caractere da senha. Essa estrutura foi escolhida por ser simples e direta ao iterar sobre uma string.

- **Estrutura condicional – `if` / `else`:**
  Responsável por fazer as verificações e decidir qual mensagem será mostrada ao usuário. Também foi usada dentro do `foreach` para verificar o tipo de cada caractere.

- **Funções prontas do C# (`char.IsUpper`, `char.IsDigit`, `char.IsLetterOrDigit`)**
  Utilizadas para identificar o tipo de cada caractere. Isso evitou a necessidade de criar checagens manuais com tabelas ASCII, deixando o código mais limpo e eficiente.

Justificativa da Abordagem

Optamos por essa abordagem por ser **simples, funcional e de fácil leitura**. Cada etapa do código está separada de forma clara, o que facilita tanto o entendimento como possíveis futuras manutenções.

Além disso, usamos recursos prontos da linguagem C#, como métodos para verificação de caracteres, o que aumenta a **eficiência** e reduz a complexidade do código. A forma como tratamos as validações também permite dar **feedbacks específicos ao usuário**, apontando exatamente o que falta na senha, o que melhora muito a usabilidade.

Exercício 2: [Tabuada.cs]

Descrição do Problema

O problema proposto busca gerar e exibir a **tabuada de multiplicação** de um número escolhido pelo usuário. O programa deve primeiramente solicitar que o usuário insira um número. Em seguida, utilizando uma estrutura de repetição, ele deve apresentar a multiplicação desse número por valores que vão de 1 até 10, exibindo todos os resultados de forma organizada no console

Passo a Passo Lógico da Resolução

1) **Solicitação do número ao usuário:** A primeira etapa do código é exibir uma mensagem pedindo que o usuário insira um número inteiro. Para isso, usamos o comando.

2) **Leitura e conversão do número:** Após o usuário digitar o número, utilizamos o `Console.ReadLine()` para capturar a entrada em forma de texto. Em seguida, convertemos o valor para inteiro usando `int.Parse()`.
    a) O valor convertido é armazenado na variável `numero`, que será usada para gerar a tabuada.

3) **Organização do código em métodos:** Ao invés de colocar toda a lógica diretamente no `Main()`, foi criado um método separado chamado `MostrarTabuada(int numero)`. Essa separação deixa o código mais limpo, organizado e facilita a manutenção.

4) **Exibição do cabeçalho da tabuada:** Dentro do método `MostrarTabuada`, é exibida uma mensagem indicando qual tabuada será mostrada, utilizando `Console.WriteLine("\nTabuada do " + numero + ":");`.

5) **Criação de laço para gerar a tabuada:** Utilizamos uma estrutura de repetição `for` que começa em 1 e vai até 10 (`for (int i = 1; i <= 10; i++)`). Em cada repetição, o programa calcula `numero * i` e exibe a operação formatada no console. Cada linha segue o formato: `numero x i = resultado.`

6) **Finalização do programa:** Após imprimir todos os 10 resultados da tabuada, o método `MostrarTabuada` finaliza, e como não há mais instruções após isso, o programa encerra sua execução normalmente.

Estruturas e Comandos Utilizados

- **Leitura de dados:**
  `Console.ReadLine()` foi utilizado para capturar a entrada do usuário como texto. `int.Parse()` foi usado para converter o texto recebido em um número inteiro, permitindo operações matemáticas.

- **Métodos separados:**
  O método `MostrarTabuada` foi criado para isolar a responsabilidade de gerar e mostrar a tabuada, melhorando a organização do código e facilitando futuras alterações.

- **Estrutura de repetição – for:**
  A estrutura `for` foi escolhida por ser a mais adequada para quando sabemos exatamente o número de repetições necessárias (neste caso, 10 vezes).

- **Exibição de dados – Console.WriteLine:**
  Utilizamos `Console.WriteLine` para mostrar a cada linha o cálculo da multiplicação. A concatenação de strings foi feita utilizando o operador `+` para montar a mensagem final.

Justificativa da Abordagem

Optamos por essa abordagem pois ela é simples, direta e de fácil entendimento.. Ao separar o código em métodos, deixamos a função `Main()` mais limpa e objetiva, o que melhora a legibilidade e facilita futuras manutenções ou expansões (como adicionar novas funcionalidades no futuro).

O uso da estrutura `for` foi escolhido pois o número de iterações é conhecido e fixo, e o uso de `Console.WriteLine` com concatenação tornou a apresentação dos resultados clara e organizada.