

EAD **UNISANTA**

PROGRAMAÇÃO ORIENTADA A OBJETOS

Me. Eng. Claudio Ferreira de Carvalho

**GUIA DA
DISCIPLINA**

1. O VISUAL STUDIO – C# – CONCEITOS BÁSICOS E INSTALAÇÃO

1.1. Objetivo

Apresentar o C#, (Lê-se C Sharp), que faz parte do Visual Studio, e será a linguagem de programação a ser utilizado nesse curso. Nesta aula também serão apresentadas informações gerais sobre o Visual Studio na versão 2022 que será utilizada neste curso, embora, se for do interesse do aluno, outras versões mais antigas podem ser utilizadas. Será também explicada a sequência para instalação do Visual Studio na Versão 2022 que possui a Linguagem de Programação C#.

1.2. Introdução

Embora o curso tenha o nome de Programação Orientada a Objeto (POO), por ser a primeira linguagem de programação apresentada e discutida no curso, serão discutidos inicialmente os conceitos básicos de Programação Estruturada e Sistemática, para posteriormente apresentar os conceitos e aplicações de POO.

Os conceitos de Programação e mesmo de Programação Orientada a Objetos, podem ser implementados em diversas linguagens, porém esse curso utilizará exclusivamente a Linguagem de Programação C#, desenvolvida pela Microsoft e integrada ao Visual Studio, que disponibiliza uma IDE – Integrated Development Environment (Ambiente de Desenvolvimento Integrado) que permite ao desenvolvedor criar, compilar e executar programas escritos em diversas linguagens dentre elas o C#.

1.3. O que é o Visual Studio

Conforme já mencionado, o Visual Studio é um IDE – Integrated Development Environment (Ambiente de Desenvolvimento Integrado) que permite ao desenvolvedor utilizar uma ou várias das linguagens disponíveis para criar, testar, executar e implementar programas desenvolvidos em diversas linguagens de programação.

A Microsoft disponibiliza 3 versões do IDE do Visual Studio 2022:

- Visual Studio 2022 Community;
- Visual Studio 2022 Professional;
- Visual Studio 2022 Enterprise.

Estas diferentes versões possuem recursos específicos que podem ser importantes dependendo do profissional ou da empresa que a irá utilizá-las. Detalhes e componentes destas diferentes versões podem ser encontrados no site da Microsoft no endereço <https://visualstudio.microsoft.com/pt-br/vs/compare/>

Para esse curso utilizaremos o Visual Studio 2022 Community que é gratuito e satisfaz perfeitamente as necessidades do curso.

1.4. Alguns conceitos básicos

1.4.1. O que é o .NET Framework

O .NET Framework é um conjunto de programas rotinas e componentes, que são integrados ao sistema operacional com o objetivo de permitir o desenvolvimento e a execução de programas criados a partir da plataforma .NET ou outra plataforma que deseje utilizar os componentes .NET Framework.

O .NET Framework permite a criação de aplicativos para serem utilizados em “Ambiente **Windows**”, diretamente na **Web**, em **Dispositivos móveis** e em aplicativos do pacote **Office**.

1.4.2. O .NET Framework possui dois componentes principais:

- Common Language Runtime (CLR)
- Biblioteca de Classes

1.4.2.1. O Common Language Runtime CLR

É a base do .NET, fornece os principais serviços de gerenciamento de memória de arquitetura de comunicação de segurança, de compilação e outros.

Em especial é importante destacar o serviço de gerenciamento de memória utiliza o GC – Garbage Collection (coletor de lixo), um componente que de tempos em tempos varre a memória do computador identificando e limpando dados que não são mais utilizados.

A constante limpeza da memória, evita que o programador tenha que constantemente utilizar comandos para limpar a memória do sistema ou que, em caso de não existirem estes comandos, o usuário tenha que constantemente reiniciar o computador.

Reiniciar o computador além de ser uma operação bastante desagradável é muito custosa em dispositivos portáteis (que na verdade nunca são desligados e sim hibernados) e em servidores de rede.

1.4.2.2. *A Biblioteca de Classes*

É uma coleção de objetos (programas), reutilizáveis e compartilháveis que podem ser utilizados pelo programador para facilitar o desenvolvimento de seus aplicativos.

1.4.3. *O que é o C#*

O C# é uma das linguagens integrantes do Visual Studio. É uma linguagem voltada a objetos que tem suas raízes na linguagem C.

1.4.4. *Como um programa em C# é executado*

A arquitetura **.NET** apresentada no **Microsoft Visual Studio**, utiliza os mesmos conceitos da tecnologia JAVA que converte os programas por intermédio do JIT (Just In Time Compiler).

Um programa desenvolvido em C# é convertido para uma linguagem Assembly, através de um compilador chamado **MSIL (Microsoft Intermediate Language)**, que é capaz de gerar arquivos de diversos tipos, como:

- **EXE** – Arquivos Executáveis, (Programas que podem ser executados diretamente nos computadores);
- **DLL** – Biblioteca de link Dinâmico que são funções que podem ser utilizadas por diversos programas.

No momento da execução do programa ele é novamente compilado, desta vez pelo **compilador JIT de acordo com o dispositivo e o ambiente em que ele será executado**.

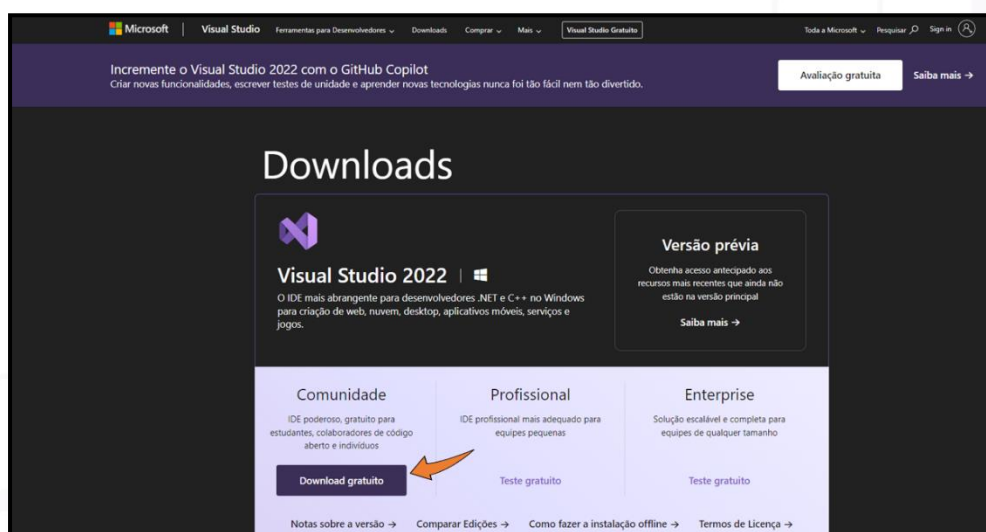
1.5. Versão C# 2020 Community

Com **objetivo de divulgar e permitir** que todos possam iniciar estudos na plataforma C#, tendo praticamente todas as facilidades da plataforma paga, a Microsoft distribui gratuitamente a versão Community (Comunidade), que pode ser baixada sem custo por estudantes e desenvolvedores, cuja instalação será apresentada a partir do próximo item.

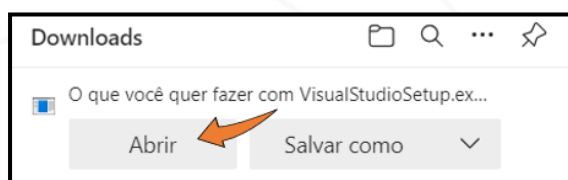
1.6. Como baixar o IDE Visual Studio 2022 Comunidade

Para baixar esta versão basta:

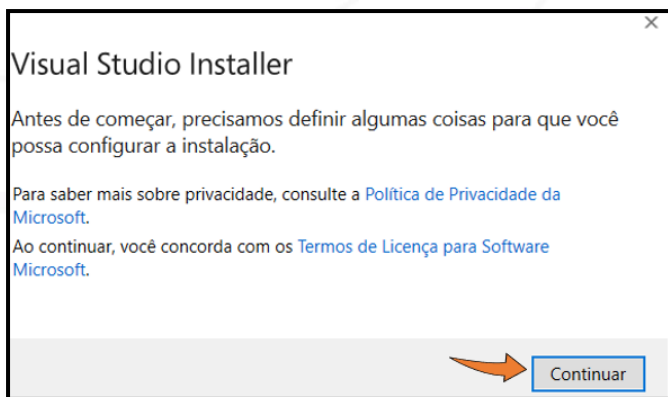
- 1) Abrir o site do Visual Studio no endereço abaixo
<https://visualstudio.microsoft.com/pt-br/downloads/>
- 2) No site do Visual Studio, 2022, clicar em “Download gratuito” no grupo Comunidade



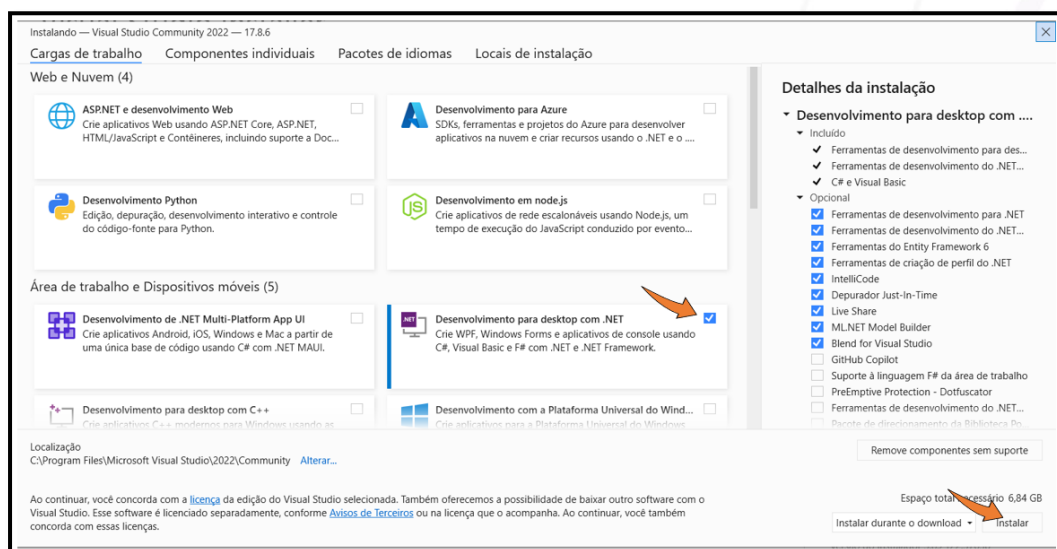
- 3) Pedir para “Abrir” o “VisualStudioSetup.exe”



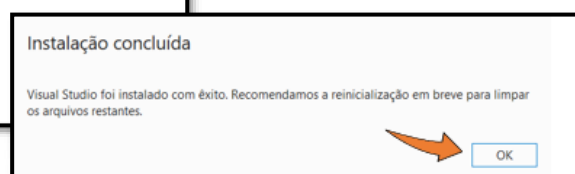
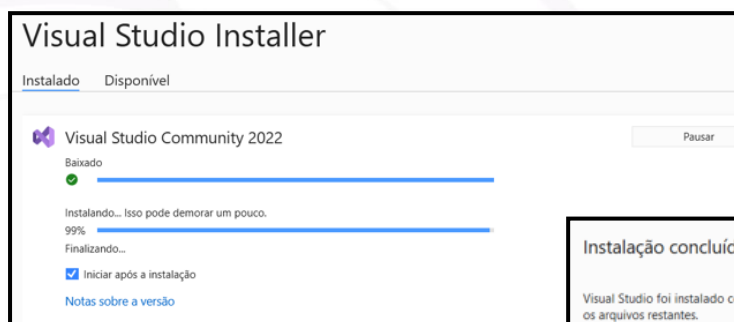
- 4) No quadro de mensagem “Visual Studio Installer” clicar em “Continuar”.



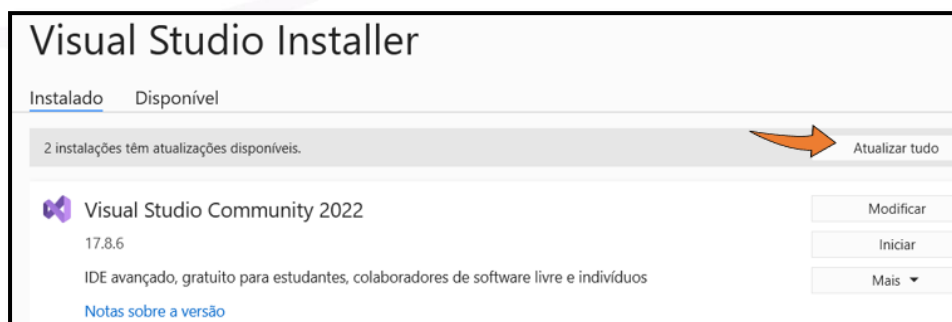
- 5) No quadro “Instalando – Visual Studio Community 2022” clicar em “Desenvolvimento para desktop com .NET”. Irá surgir o quadro “Detalhes da instalação”, não alterar, e clicar no “Botão Instalar”.



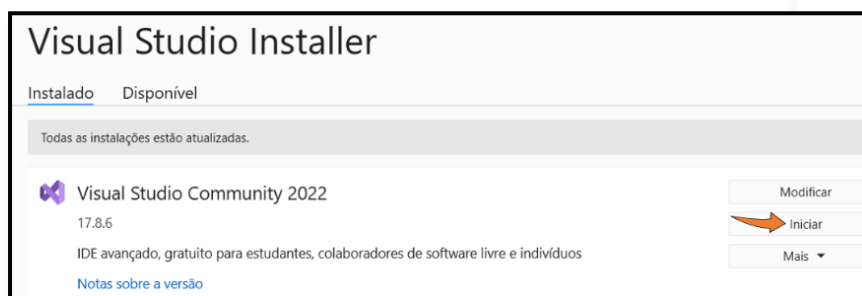
- 6) Aguardar baixar e encerrar a instalação. Dependendo do computador pode levar em até uns 45 minutos. Ao encerrar surgirá o quadro “Instalação Concluída”, basta clicar no “Botão OK”



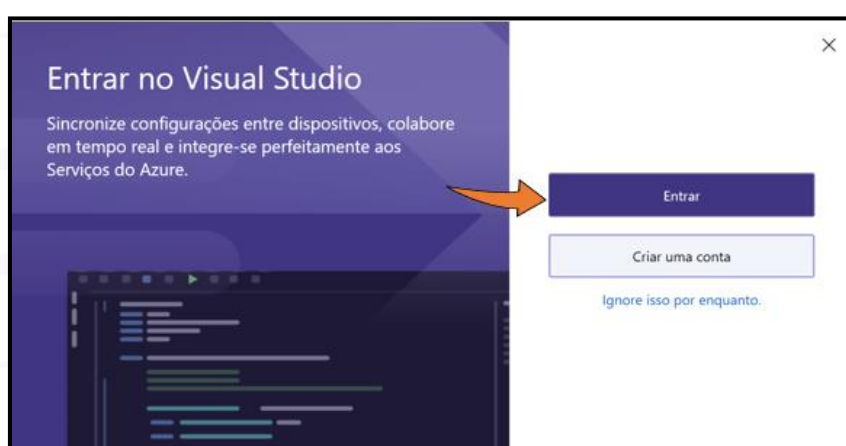
- 7) Após a conclusão, provavelmente surgirá um quadro informando que existem atualizações a serem feitas, basta clicar no item “Atualizar tudo” e aguardar as novas atualizações que podem durar mais algum tempo (talvez o mesmo tempo da primeira parte da instalação).



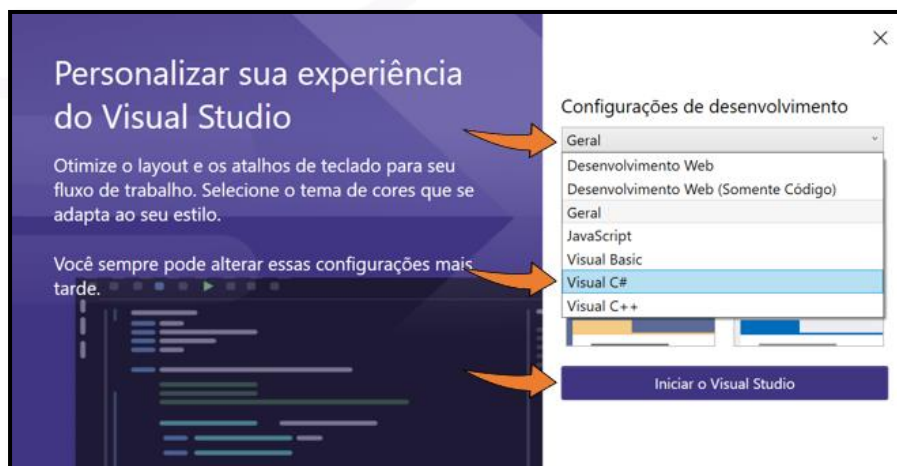
- 8) Após encerrar as atualizações a instalação voltará à tela “Visual Studio Installer”. Basta clicar em “Iniciar”



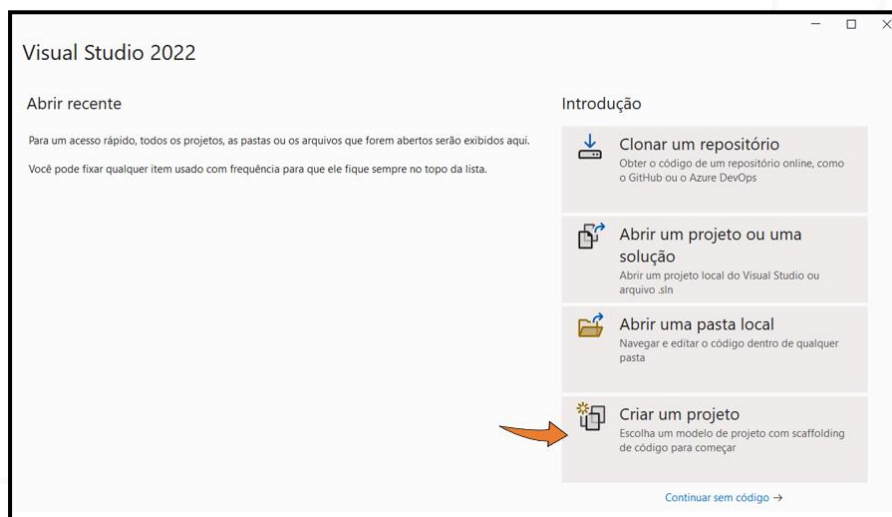
- 9) O Visual Studio 2022 será aberto e surgirá a tela de entrada, basta clicar o “Botão Entrar”.



- 10) O Visual Studio 2022 apresentará a tela de configuração. Clicar em “Geral” e no menu suspenso que surgir escolher “C#”. Caso deseje alterar as cores do layout¹ e clicar em “Iniciar o Visual Studio”.

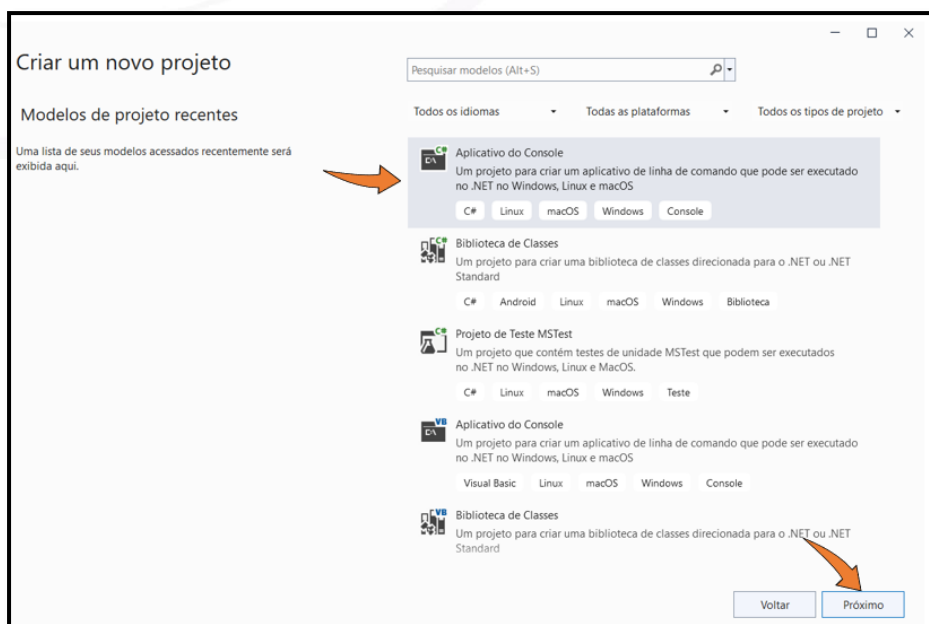


- 11) Na tela de entrada do Visual Studio 2022 escolher “Criar um projeto”

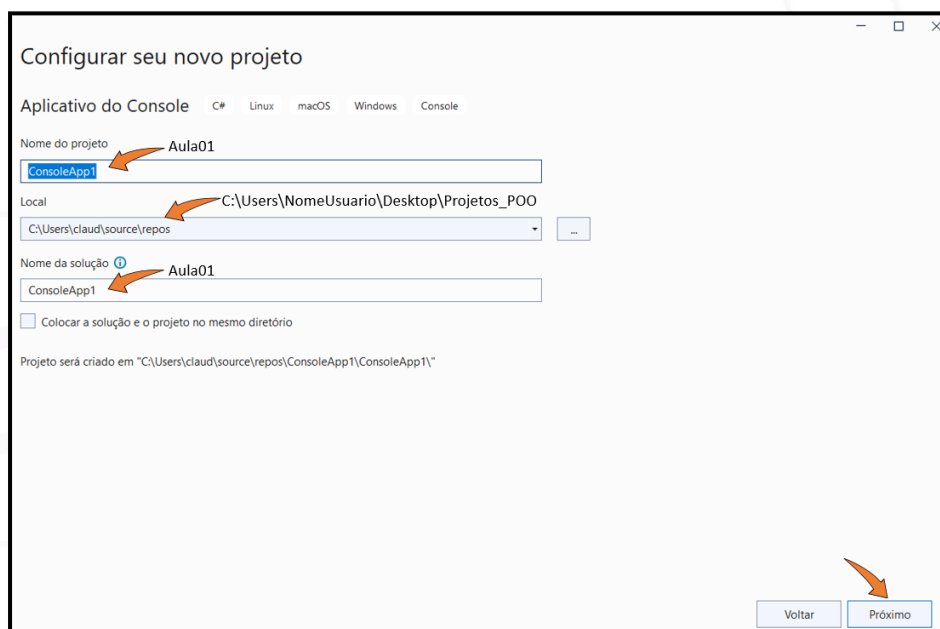


¹ Para melhorar a qualidade das telas capturadas, neste material foi escolhido “Azul com alto contraste” para a tela. Entretanto o usuário pode escolher as cores desejadas.

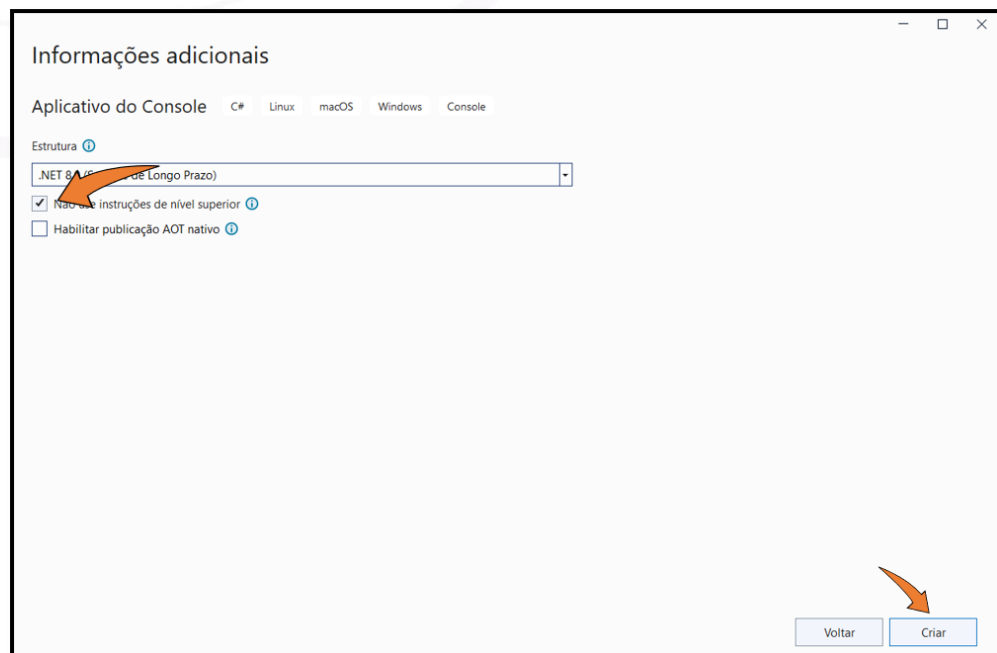
- 12) Na tela de “Criar um novo projeto” escolher “Aplicativo de console” e clicar o “Botão Próximo”



- 13) Na tela de “Configurar seu novo projeto” colocar o nome do projeto, da solução e a pasta que ele será gravado. A sugestão é que tanto o nome do projeto como da solução seja Aula01 e que seja criada uma pasta de nome Projetos_POO na área de trabalho. Em seguida clicar o “Botão Próximo”



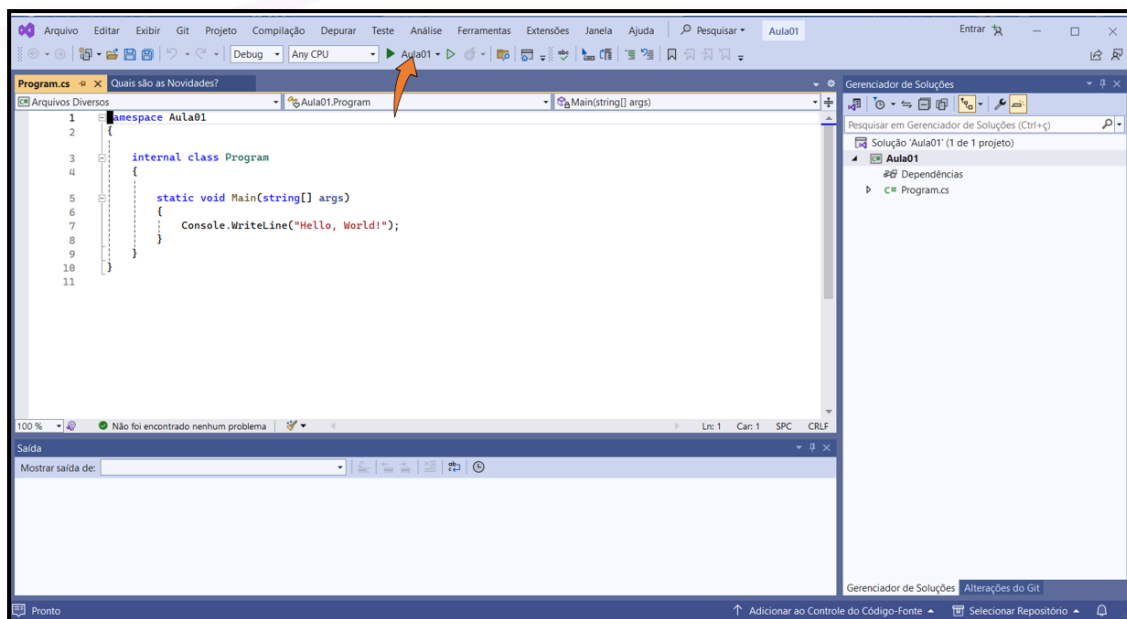
- 14) Na tela de “Informações adicionais” marcar “**Não use instruções de nível superior**” e clicar o “Botão Criar”



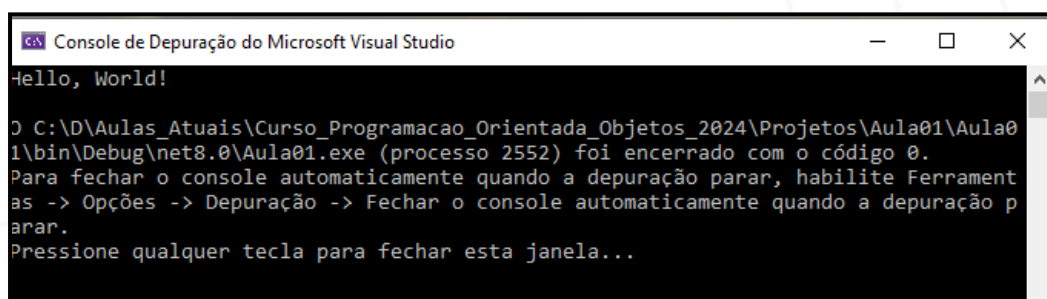
Importante

Atenção, é importantíssimo marcar a opção “Não use instruções de nível superior”

- 15) A tela que surgiu, é ao “Ambiente de Programação” que será utilizado durante todo o curso. Nesta tela está escrito um programa que tem a função de escrever na tela “Hello, Word”; basta clicar no ícone de “Pay” com o nome Aula01 que o programa será executado.



- 16) A janela de execução do programa é a indicada abaixo. Após observar as informações basta clicar qualquer tecla para fechar a janela. ”



Na próxima aula serão apresentados os elementos da tela de programação. Como não foram feitas alterações no programa, basta fechar as janelas abertas.

2. INICIANDO O C#, IDE, EXECUÇÃO DE PROGRAMAS E DEBUG

2.1. Objetivo

Apresentar a IDE do C#, a execução de programas e o conceito e utilização do Debug.

2.2. Introdução

Neste curso serão criados programas em **C#**, utilizando a opção “**Console**”, a partir do Visual Studio 2022. Estes programas devem ficar armazenados no computador do usuário em uma pasta escolhida. É importante que esta pasta seja de fácil acesso para que o material criado possa facilmente ser visualizado e, se necessário, manipulado.

Para que isso deve ser criada uma pasta com o nome Projetos_POO em um local de fácil acesso. Na aula anterior foi sugerida uma pasta na “Área de trabalho”, entretanto fica a cargo do aluno escolher o nome da pasta e a localização desta pasta².

Para melhorar a qualidade na captura de telas, foi escolhido o layout “Azul de alto contraste”, entretanto fica a cargo do usuário escolher o layout que mais agradar.

2.3. Abrindo o Visual Studio

Para iniciar o **Visual Studio**, basta:

² Na construção deste material a pasta foi colocada em outro local com um link para ela na área de trabalho. Esta é uma solução bem interessante para permanecer durante o curso.

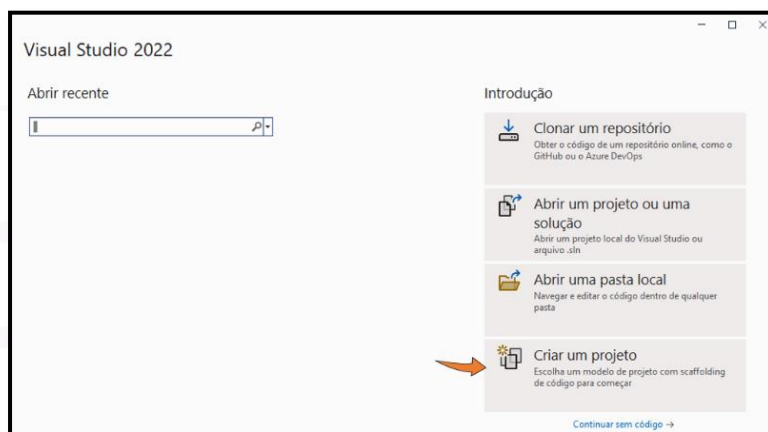
- 1) Utilizar o link “**Visual Studio 2022**” que apresentará a Tela de Splash:



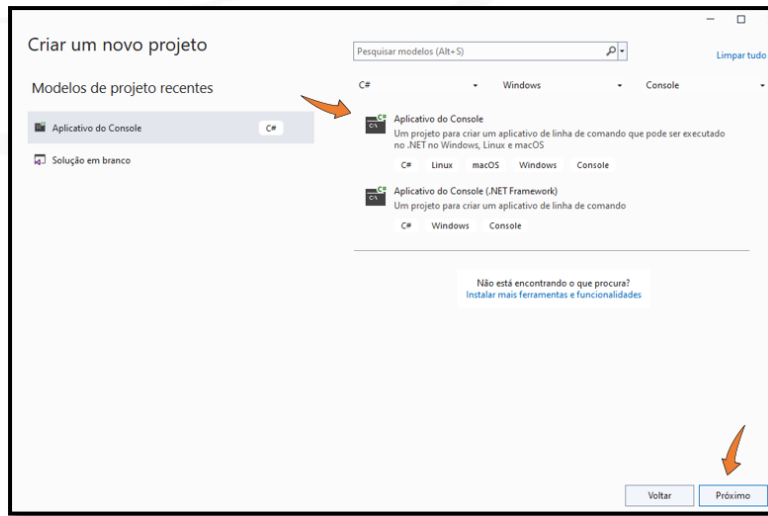
Importante

Atenção: Não utilizar o Blend for Visual Studio 2022 ele é para aplicativos Web e XAML

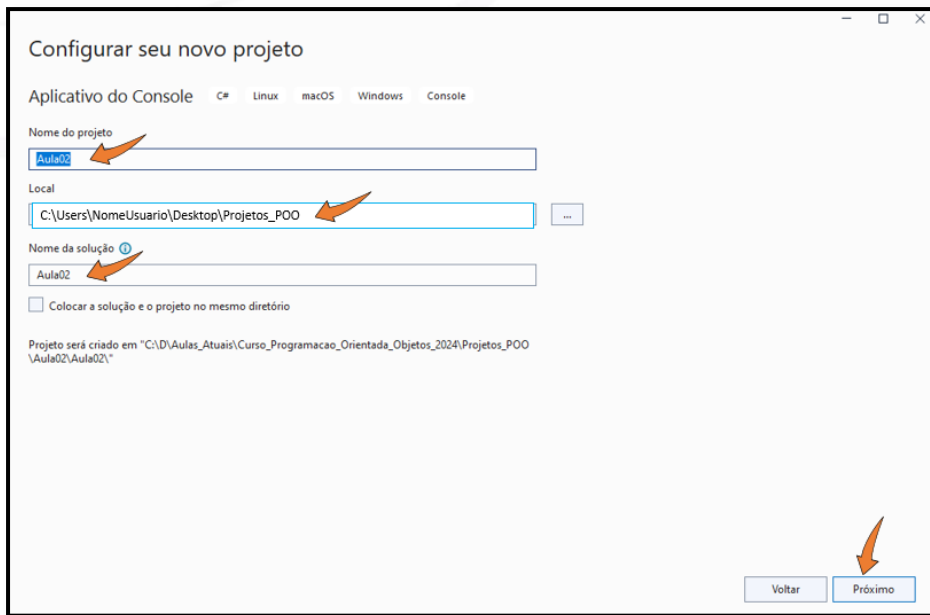
- 2) Após a tela de Splash, aparecerá a tela para escolha da opção onde deve ser escolhido “Criar um novo projeto”.



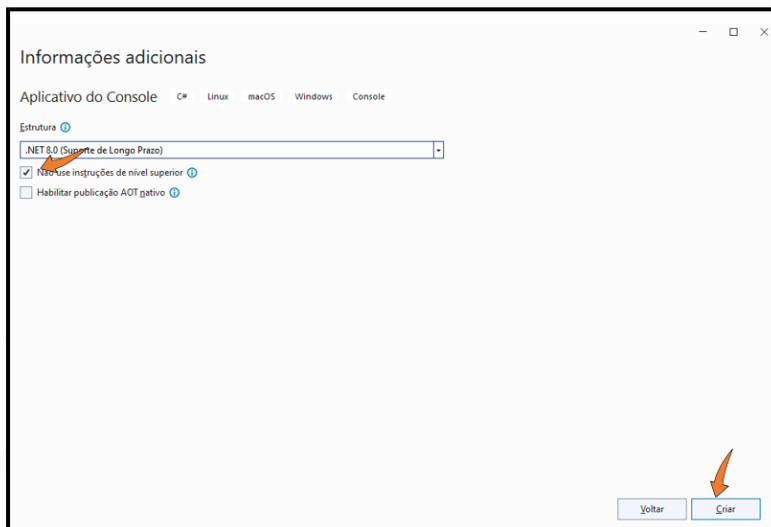
- 3) Em seguida aparecerá a tela para escolher o tipo de projeto e deve ser escolhido “Console” e clicado o Botão Próximo



- 4) Em configurar um novo projeto escolher o nome Aula02 o endereço da pasta criada anteriormente e o nome da solução Aula 02 e clicar o “Botão Próximo”



- 5) Na tela Informações adicionais marcar “**Não use instruções de nível superior**” e clicar o “Botão Criar”

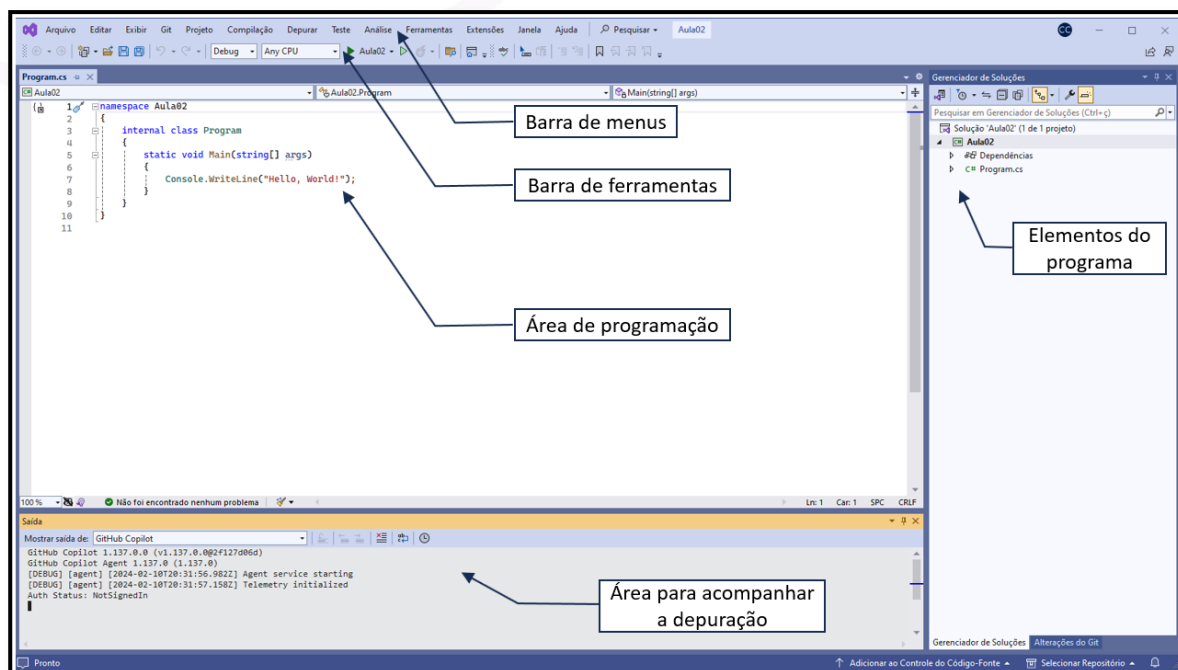


Importante

Atenção: Escolher “Não use instruções de nível superior” é de fundamental importância para o bom acompanhamento do curso. Não inicie qualquer exercício deste curso com esta opção desmarcada.

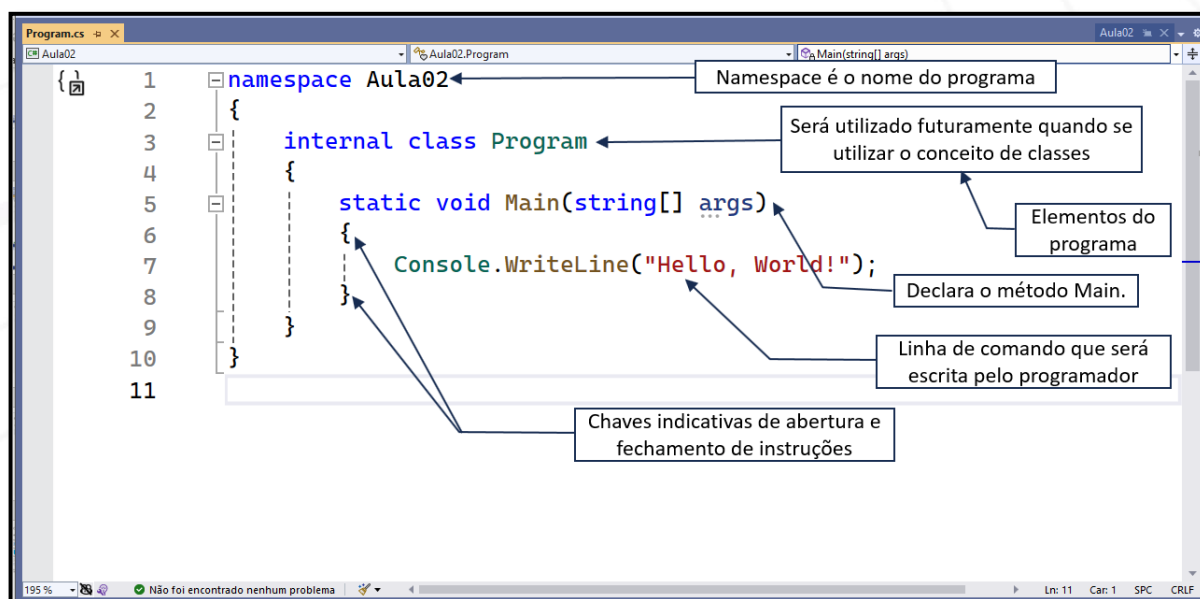
2.4. Área de Programação do C#

Após realizar as operações do item anterior, será aberto o “**Ambiente de Desenvolvimento do C#**” com aspecto conforme abaixo:



2.5. Elementos da programação em C#

Como qualquer programação, a programação em C# deve possuir instruções que devem ser adicionadas seguindo regras e básicas chamadas de Sintaxe. Abaixo explicações sobre o programa “Hello Word!” fornecido como exemplo pela Microsoft



Observações Importantes:

- **Todas as seções** de um programa em C# **iniciam com chave de abertura {e fecham com chave de fechamento}**.
- Após o **namespace** seguido do nome do programa existe uma chave {abrindo o programa que é fechada no final do programa}.
- Após a **internal class program**, existe uma chave {abrindo a classe que é fechada na penúltima linha do programa}.
- Após a declaração do método **Main** – (string[] args), existe uma chave de abertura {e todos os comandos que o programador desejar escrever deverão estar após esta chave e antes da chave de fechamento} que fecha o método.
- Durante a programação o usuário também deverá colocar chaves conforme será orientado nas aulas seguintes.

2.6. Como um programa é executado

Lembrando que, o computador só entende linguagem de máquina, ou seja, sinais elétricos, com objetivo de facilitar a interpretação destes sinais elétricos pelo computador, decidiu-se que se trabalharia unicamente com dois tipos de sinais **ligado** e **desligado** (*que eletricamente constitui-se em “com tensão elétrica” e “sem tensão elétrica”*).

Para facilitar, as informações destes dados para o computador, decidiu-se utilizar o valor **1 para ligado** e o valor **0 para desligado**. Então, qualquer que seja o programa, para que ele passe instruções ao computador ele precisa fornecer conjuntos de zeros (0) e uns (1), portanto, mesmo uma linguagem de programação de alto nível, como é o caso do C#, e qualquer outra linguagem, em última análise precisa fornecer ao computador conjuntos de zeros (0) e uns (1).

No caso do C#, o programa gerado pelo desenvolvedor, que tem o nome de **programa fonte**, e tem a extensão “**cs**”, não é um conjunto de zeros (0) e uns (1) e, portanto, precisa ser convertido para ser executado.

Então, a execução de um programa criado em C# pode ser feita de duas maneiras:

2.6.1. Dentro do ambiente do C#.

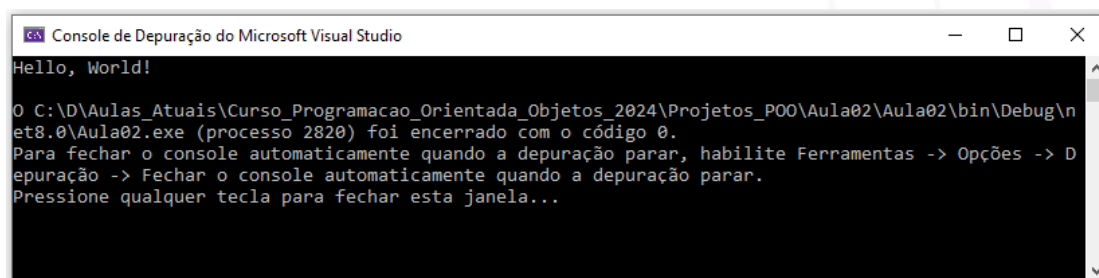
Neste caso o próprio arquivo de extensão “**cs**” sofre um processo chamado de “**debug**”, e caso não existam erros o programa será debugado e executado imediatamente. A operação de debug pode ser disparada das seguintes maneiras:

- Escolhendo no menu **Depurar** o item “Iniciar depuração”;
- Digitando a tecla **F5**;
- Clicando na seta com o nome do programa na Barra de Ferramentas.

2.6.2. Fora do Ambiente C#.

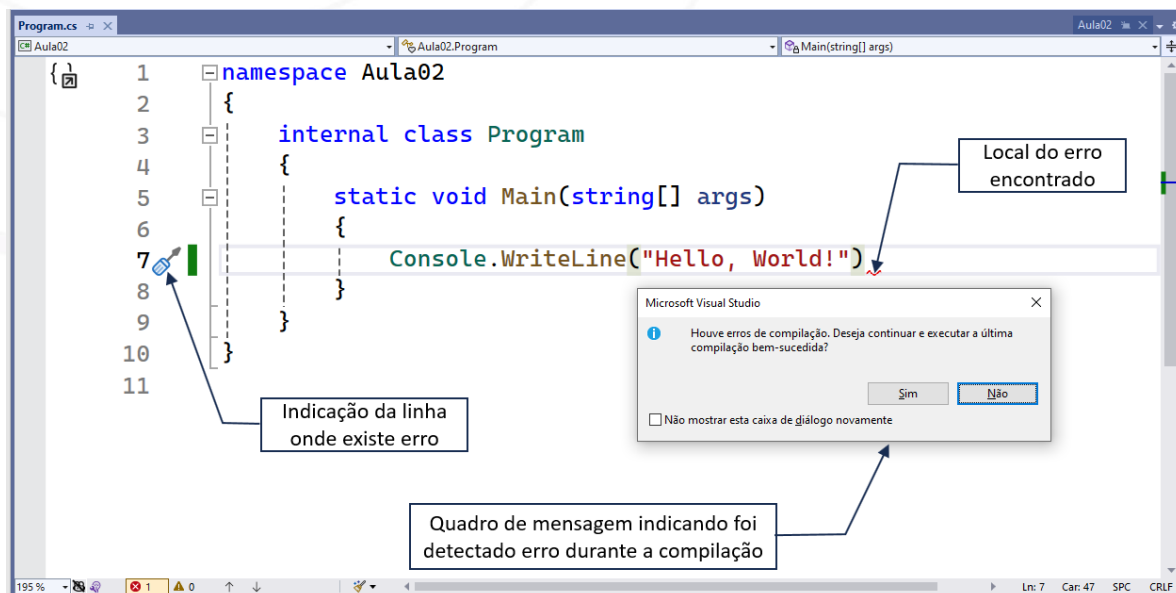
Neste caso o arquivo “**cs**” será compilado, gerando um arquivo executável. Este arquivo executável pode ser executado em qualquer computador que possua o **.NET Framework** instalado. Em versões anteriores, para gerar o programa executável, era necessário utilizar a Build Solution, nas novas versões existe a possibilidade de gerar um pacote inteiro que será apresentado em outras oportunidades.

2.7. Programa Executado



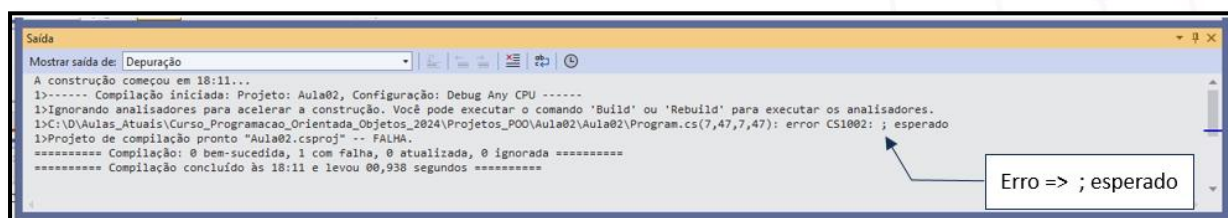
2.8. Caso o programa apresente erros durante a operação de Debug

A operação de debug irá conferir o programa com objetivo de verificar se existem erros de sintaxe (erros na escrituração dos comandos). Caso existam erros, o processo é interrompido e surgem as informações a seguir:



Ao receber o quadro de mensagem o desenvolvedor já sabe que seu programa apresenta um erro, **portanto ele precisa procurar o erro**. Para procurar o erro ele deve **clicar o botão “Não”** (para interromper a compilação) e verificar na tela as mensagens que dão as informações necessárias para que o usuário concerte o erro.

As mensagens deverão ser algo como a figura abaixo, onde é informado:



- A descrição de erro (no exemplo, é esperado um ;).
- O nome do arquivo (no exemplo Aula02.cs).
- A linha em que o erro foi encontrado (no exemplo linha 7).
- A coluna que o erro foi encontrado (no exemplo a coluna 47).
- O nome do projeto (no exemplo Aula02).

2.8.1. Algumas observações importantes sobre o processo de debug e erros

Caso o debug apresente erros, **não adianta clicar o botão Sim**, pois conforme a mensagem **será executada a última versão correta do programa**, o que sem dúvida não interessa ao **desenvolvedor, pois ele precisa encontrar o erro que existe nesta nova versão**.

O fato de o desenvolvedor encontrar **o erro e corrigi-lo não significa que o programa não apresente mais erros**, pois **a operação de debug foi interrompida quando o primeiro erro foi encontrado**, portanto, podem perfeitamente existirem outros erros nas demais linhas do programa que não foram conferidas.

O programa só pode ser considerado encerrado ou mesmo convertido em executável (exe) quando não existirem mais erros. Não adianta manter ou distribuir programas com erros.

3. COMANDO *Write* E *WriteLn*, FORMATAÇÕES EM TELA

3.1. Objetivo

Apresentar comandos: “Comentário”; “Write”; “WriteLn”; seus objetivos, sintaxes e exemplos assim como algumas formatações possíveis aos comandos

3.2. Introdução

Qualquer linguagem de programação, entre elas o C# é composto por **comandos que realizam instruções ou operações** de acordo com os desejos do programador, portanto, cada comando tem um **objetivo, significado ou função**.

Assim como em um idioma, estes **comandos**, que na verdade são **palavras ou conjuntos de palavras**, devem ser escritos de maneira correta, obedecendo rigorosamente o que se convencionou chamar de **sintaxe**.

A partir desta aula, serão apresentados os comandos que serão utilizados no curso de C#, sempre com seu **objetivo**, sua **sintaxe** e **exemplos** de aplicação.

3.3. Como utilizar o Editor de Programa do C#

O C# possui um editor que facilita muito a digitação dos comandos da linguagem, visto que possui a função “Auto Completar”, implementada por um Menu suspenso que surge durante a digitação. Então, para escrever um comando basta:

- Iniciar a digitação do comando, (*lembrar que o C# é case sensitivo – letras maiúsculas são diferentes de letras minúsculas*).
- Imediatamente o “**Menu Suspenso Auto Completar**” com os comandos disponíveis irá surgir. (à medida que mais letras são digitadas, o menu suspenso fica mais restrito, até ficar somente com o comando que se deseja).
- Entretanto, **a ideia não é digitar todo o comando** e sim, logo que ele ficar visível utilizar as teclas “**cursor para baixo**” e “**cursor para cima**”, ou clicar com o Mouse na barra de rolamento do “Menu Suspenso” até encontrar o comando desejado. Quando o comando desejado estiver em evidência, basta digitar a tecla <Enter>.

- É possível também digitar a tecla Tab quando o curso estiver sobre a palavra.
- Caso o comando seja composto, ou seja, mais instruções devam ser adicionadas, basta digitar ponto que novamente o “**Menu Suspenso Auto Completar**” irá aparecer.
- Neste novo menu, basta agir da mesma maneira anteriormente citada, para encontrar as demais partes do comando que se deseja.

Nota: É muito interessante utilizar sempre o recurso Auto Completar, pois ele evita erros, principalmente no caso de diferenças entre letras maiúsculas e minúsculas, muito comum em programadores sem prática.

Nota: É importante observar também que caso o comando não apareça ele não pode ser escrito desta maneira, não adianta insistir, pois com certeza futuramente isto irá gerar erros no momento de executar o debug.

3.4. Comando Comentário

3.4.1. Objetivo:

Permitir que o desenvolvedor escreva comentários durante a programação.

Realmente, o comando comentário, não tem nenhuma função para a linguagem de programação, o que se está fazendo quando se coloca um **comando comentário** é equivalente a falar ao compilador que a linha que inicia com o comando comentário deve ser desconsiderada.

Em resumo, os comentários só servem para dar ao desenvolvedor a oportunidade de escrever informações no programa que lhe serão úteis como lembretes ou explicações de decisões que ele tomou durante a elaboração do programa. Um bom programador deve colocar o máximo possível de comentários, de maneira que estes possam lhe auxiliar principalmente no futuro, quando ele for fazer alterações ou manutenções no programa.

Outra utilização muito comum do **comando comentário** é utilizá-lo para durante o desenvolvimento do programa fazer com que algumas linhas não sejam executadas. Esta ação pode ser útil quando por exemplo se está testando uma instrução com diversas

sintaxes e deseja-se fazer várias tentativas sem apagar as anteriores. Programadores experientes utilizam muito o comando comentário desta maneira. Eles costumam chamar esta operação de “**Comentar essa linha**”

3.4.2. *Sintaxe:*

Existem no C#, duas sintaxes para o comando comentário.

- ✓ Para uma única linha de comentário:

Basta iniciar a linha de comentário com duas barras normais. Não é necessário marcar o final da linha.

//Comentário

- ✓ Para comentários com mais de uma linha:

Deve-se iniciar a linha de comentário com uma barra normal seguida de asterisco e encerrar o comentário com asterisco seguido de uma barra normal.

/*Comentário */

3.4.3. *Exemplo:*

```
1. namespace Aula03_Ex01
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Esta é uma linha de comentário que será desprezada pelo comando
8.             /* Caso se deseje escrever comentários
9.             * com mais de uma linha basta iniciar
10.            * com barra asterisco e fechar com */
11.        }
12.    }
13. }
```

Nota: Este programa não apresentará nada na tela, portanto não precisa ser escrito basta observar e analisar as maneiras de se escrever comentários.

3.5. Write e WriteLine

3.5.1. Objetivo:

Informar dados na tela.

Como o comando é utilizado em programas do tipo “ConsoleApplication” este deve ser precedido de uma das instruções Console descritas a seguir:

- Console.Write => coloca informações na tela e mantém o cursor após o último caractere.
- Console.WriteLine => coloca informações na tela e passa o cursor para a próxima linha.

3.5.2. Sintaxe:

Console.Write(“Texto a ser escrito – deve estar entre aspas duplas”);

Console.WriteLine(“Texto a ser escrito – deve estar entre aspas duplas”);

Obs:

Após o comando deve-se abrir parênteses.

Caso se deseje escrever um texto ele deverá estar entre aspas duplas.

Quando se encerrar o texto deve-se fechar as aspas duplas e o parágrafo.

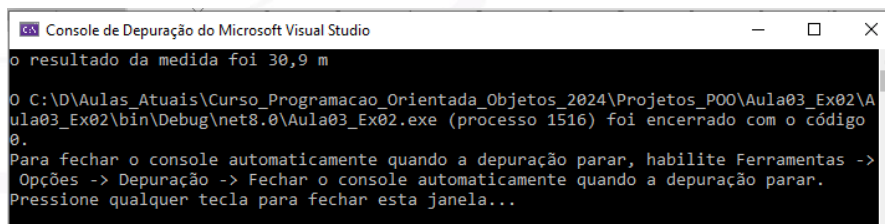
O comando deve ser encerrado com ponto e vírgula

3.5.3. Exemplo com Console.WriteLine:

Obs: Em versões anteriores do C# este programa será executado e encerrado tão rapidamente que nada será visto na tela. Para que seja possível ver na tela o que foi executado é necessário a colocação de um comando Readkey(), conforme próximo item, para manter a tela aberta até que uma tecla qualquer seja digitada.

```
1. namespace Aula03_Ex02
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Coloca um conjunto de caracteres na tela
8.             Console.WriteLine("o resultado da medida foi 30,9 m");
9.         }
10.    }
11. }
```

3.5.4. Tela



```

Microsoft Visual Studio Debug Console
o resultado da medida foi 30,9 m

O C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula03_Ex02\Aula03_Ex02\bin\Debug\net8.0\Aula03_Ex02.exe (processo 1516) foi encerrado com o código 0.
Para fechar o console automaticamente quando a depuração parar, habilite Ferramentas -> Opções -> Depuração -> Fechar o console automaticamente quando a depuração parar.
Pressione qualquer tecla para fechar esta janela...

```

3.6. ReadKey

3.6.1. Objetivo:

Interrompe a programação e espera a digitação de uma tecla. Este comando inclusive evita que a tela de execução coloque todos os comentários que podem ser vistos na tela acima.

3.6.2. Sintaxe:

```
Console.ReadKey();
```

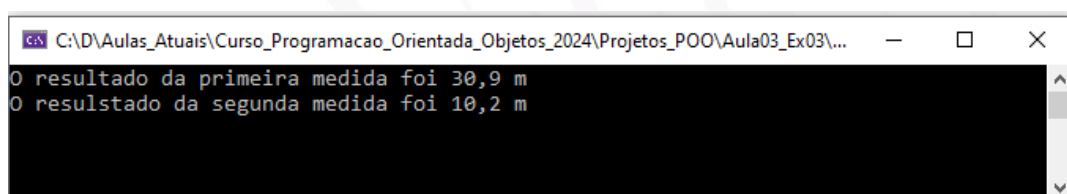
3.6.3. Exemplo com: Console.ReadKey, Console.WriteLine e Console.Write:

```

1. namespace Aula03_Ex03
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Coloca conjuntos de caracteres na tela
8.             Console.WriteLine("O resultado da primeira medida foi 30,9 m");
9.             Console.Write("O resultado da segunda medida foi 10,2 m");
10.            // Mantém a tela aberta e retira os comentários esperando digitar uma
11.            tecla
12.            Console.ReadKey();
13.        }
14.    }

```

3.6.4. Tela:



```

C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula03_Ex03\...
O resultado da primeira medida foi 30,9 m
O resultstado da segunda medida foi 10,2 m

```

Nota: Observar que a primeira linha “O resultado da primeira medida foi 30,9 m” foi escrito com o comando WriteLine, por este motivo, após escrever a mensagem, o cursor passou para a próxima linha, então, a segunda frase “O resultado da segunda medida foi 10,2 m” foi escrito na linha seguinte.

Observar que como a segunda frase foi escrita com o comando Write, o cursor permaneceu no final da linha.

3.7. Concatenando informações com + no comando Write

3.7.1. Objetivo:

Concatenar, consiste em utilizar o mesmo comando Write ou WriteLine para colocar mais de um conjunto de caracteres.

É possível concatenar:

- Utilizando o sinal de + para separar os dois conjuntos.
- Utilizando **parâmetros de formatação**.

3.7.2. Sintaxe utilizando o sinal de +

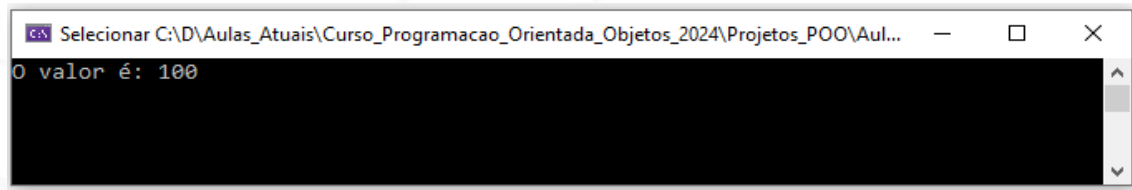
Basta colocar o sinal de + entre os conjuntos de caracteres que serão apresentados na tela.

```
Console.WriteLine("conjunto de caracteres 1" + "conjunto 2");
```

3.7.3. Exemplo:

```
1. namespace Aula03_Ex04
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Concatena utilizando a sintaxe =
8.             Console.WriteLine("O valor é: " + 100);
9.             Console.ReadKey();
10.        }
11.    }
12. }
```


3.7.4. Tela:



3.8. Utilizando parâmetro {0} na formatação do Write.

3.8.1. Objetivo

Utilizar o parâmetro {0} no meio do conjunto de caracteres que é delimitado por aspas duplas. Após fechar aspas deve ser colocada uma vírgula e em seguida o que será apresentado na tela no lugar do parâmetro.

Embora aparentemente esta opção pareça desnecessária ela é muito importante para mostrar valores existentes em variáveis de memória ou mesmo para formatar a apresentação de resultados como será mostrado a seguir:

3.8.2. Sintaxe

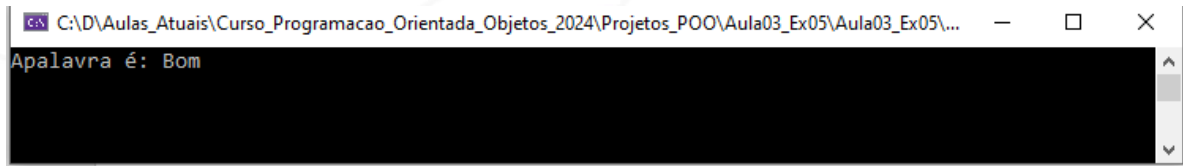
```
Console.WriteLine("A palavra é: {0}", Bom);
```

Obs: o parâmetro {0} será substituído por Bom

3.8.3. Exemplo:

```
1. namespace Aula03_Ex05
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             Console.WriteLine("A palavra é: {0}", "Bom");
8.             Console.ReadKey();
9.         }
10.    }
11. }
```

3.8.4. Tela:

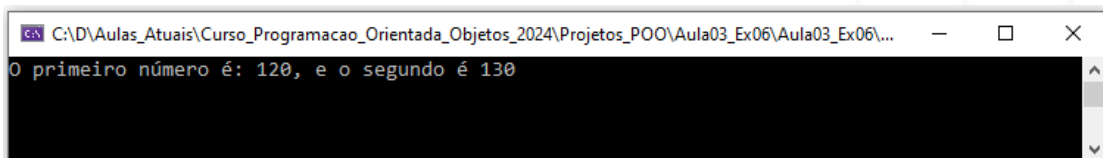


Obs: Caso se deseje mostrar mais de um parâmetro, basta repetir a sintaxe utilizando os números 0, 1, 2, 3 etc.

3.8.5. Exemplo:

```
1. namespace Aula03_Ex06
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             Console.WriteLine("O primeiro número é: {0}, e o segundo é {1}", 120,
8.             130);
9.             Console.ReadKey();
10.        }
11.    }
```

3.8.6. Tela:



3.9. Delimitando largura de informações com o Comando Write

3.9.1. Objetivo:

Caso seja utilizado o Comando Write com a sintaxe de “**parâmetros de formatação**”, é possível delimitar o espaço onde uma informação será colocada.

Para tanto, basta colocar após o “**parâmetro de formatação**”, uma vírgula e em seguida **um número inteiro representando o espaço que será disponibilizado para a apresentação da informação**.

Caso seja colocado um **número positivo o alinhamento será justificado à direita**, caso seja colocado um **número negativo o alinhamento será justificado à esquerda**.

3.9.2. Sintaxe:

`Console.Write("Texto desejado {0,10} novo texto desejado {1,10}", valor0, valor1);`

Obs: valor0 e valor1 são os números que se deseja escrever em 10 espaços justificado à direita.

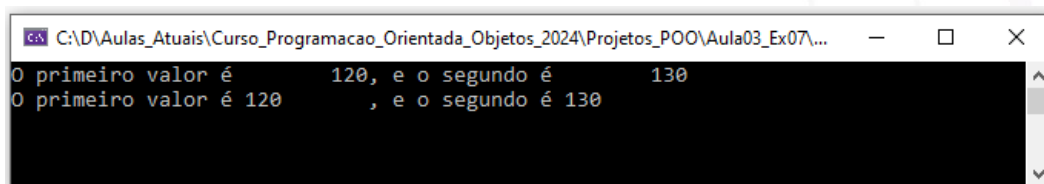
3.9.3. Exemplo:

```

1. namespace Aula03_Ex07
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             Console.WriteLine("O primeiro valor é {0,10}, e o segundo é {1,10}",
120, 130);
8.             Console.WriteLine("O primeiro valor é {0,-10}, e o segundo é {1,-10}",
120, 130);
9.             Console.ReadKey();
10.        }
11.    }
12. }

```

3.9.4. Tela:



3.10. Determinando o número de casas decimais com o comando Write

3.10.1. Objetivo:

Caso seja utilizado o Comando Write com a sintaxe de “**parâmetros de formatação**”, é possível determinar o número de casas decimais que um número será informado.

Para tanto, basta colocar após o “parâmetro de formatação” ou após a indicação de largura disponibilizada **dois pontos e em seguida a letra F e o número de casas desejadas**.

3.10.2. Sintaxe:

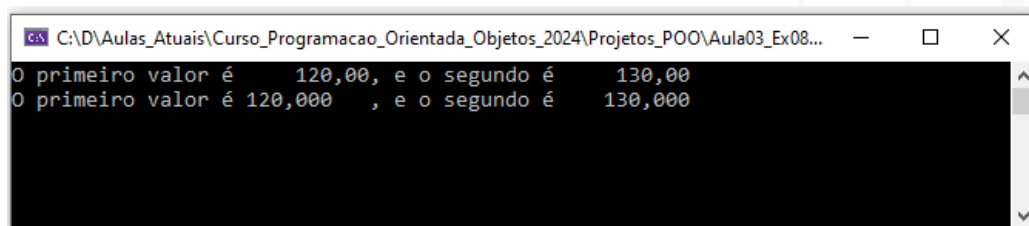
`Console.Write("Texto {0,15:F2}", valor0);`

Obs: O valor0 é o número que se deseja escrever com duas casas após a vírgula e em 15 espaços.

3.10.3. Exemplo:

```
1. namespace Aula03_Ex08
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             Console.WriteLine("O primeiro valor é {0,10:F2}, e o segundo é
8.             {1,10:F2}", 120, 130);
9.             Console.WriteLine("O primeiro valor é {0,-10:F3}, e o segundo é {1,-
10.             10:F3}", 120, 130);
11.             Console.ReadKey();
12.         }
13.     }
14. }
```

3.10.4. Tela:



3.11. Comando Write com tabulação

3.11.1. Objetivo:

O comando \t promove uma tabulação se colocado entre as aspas dentro de um comando Write.

3.11.2. Sintaxe:

```
Console.Write("Texto1 \t Texto2 \t Texto3");
```

Obs: Cada um dos comandos \t coloca uma tabulação.

3.11.3. Exemplo:

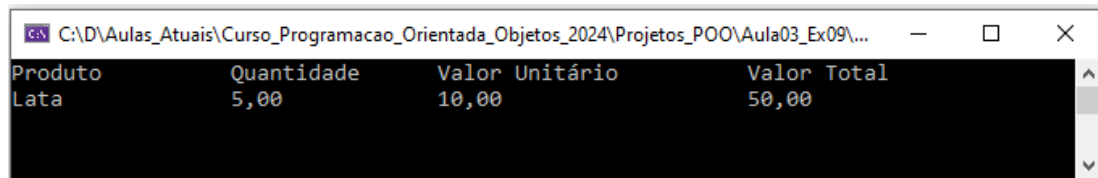
```
1. namespace Aula03_Ex09
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
```

```

7.         Console.WriteLine("Produto \t Quantidade \t Valor Unitário \t Valor
    Total \t");
8.         Console.WriteLine("Lata \t \t 5,00 \t \t 10,00 \t \t 50,00 ");
9.         Console.ReadKey();
10.    }
11. }
12. }

```

3.11.4. Tela:



É importante notar que dependendo da quantidade de caracteres utilizados para lançar os valores na tela, podem ser necessárias mais de uma instrução \t como notado na linha 8 do exemplo acima.

3.12. Inserir linhas em um Comando Write

3.12.1. Objetivo:

O comando \n insere uma linha se colocado entre as aspas de um comando Write

3.12.2. Sintaxe:

```
Console.Write("Texto1 \n ");
```

Obs: Após escrever Texto1 o cursor passará para a próxima linha.

É possível pular mais de uma linha colocando vários \n.

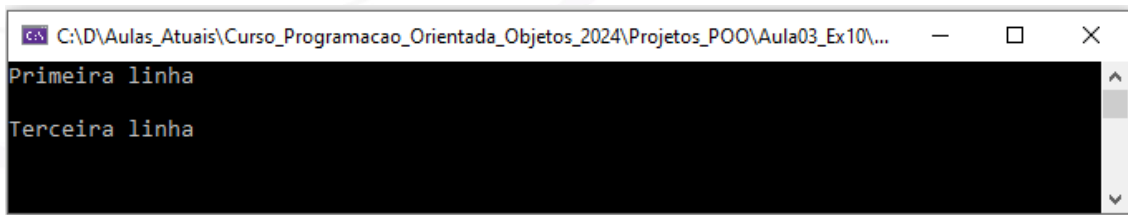
3.12.3. Exemplo:

```

1. namespace Aula03_Ex10
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             Console.WriteLine("Primeira linha \n");
8.             Console.WriteLine("Terceira linha");
9.             Console.ReadKey();
10.        }
11.    }
12. }

```

3.12.4. Tela:



3.13. Exemplo resumindo os itens deste capítulo

O exemplo a seguir apresenta todas as implementações discutidas nesta aula.

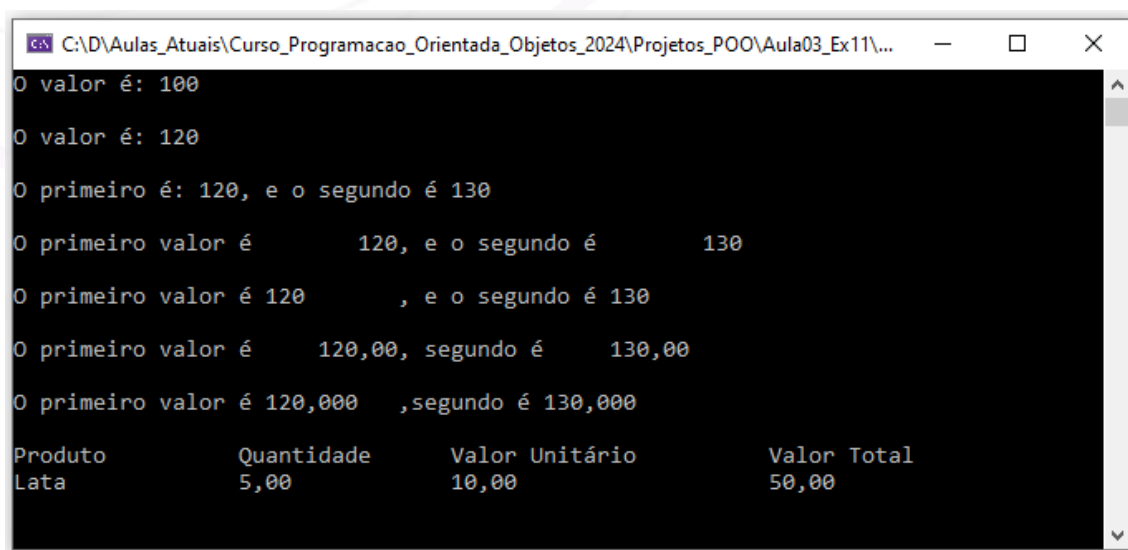
3.13.1. Código:

```

1. namespace Aula03_Ex11
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Wirte concatenando informações utilizando a sintaxe +
8.             Console.WriteLine("O valor é: " + 100);
9.             // Pular uma linha com \n no comando Write
10.            Console.WriteLine("\n");
11.            // Wirte concatenando utilizando parâmetros de formatação
12.            Console.WriteLine("O valor é: {0}", 120);
13.            // Pular uma linha com \n no comando Write
14.            Console.WriteLine("\n");
15.            // Write concatenando utilizando dois parâmetros de formatação
16.            Console.WriteLine("O primeiro é: {0}, e o segundo é {1} \n", 120,
17.            130);
18.            // Concatena utilizando parâmetros de formatação e delimitadores
19.            Console.WriteLine("O primeiro valor é {0,10}, e o segundo é {1,10}
20.            \n", 120, 130);
21.            Console.WriteLine("O primeiro valor é {0,-10}, e o segundo é {1,-10}
22.            \n", 120, 130);
23.            // Concatena utilizando parâmetros de formatação delimitadores e duas
24.            // ou três casas
25.            Console.WriteLine("O primeiro valor é {0,10:F2}, segundo é {1,10:F2}
26.            \n", 120, 130);
27.            Console.WriteLine("O primeiro valor é {0,-10:F3},segundo é {1,-10:F3}
28.            \n", 120, 130);
29.            // Comando Write com tabulação
30.            Console.WriteLine("Produto \t Quantidade \t Valor Unitário \t Valor
31.            Total \t");
32.            Console.WriteLine("Lata \t \t 5,00 \t \t 10,00 \t \t 50,00");
33.            // Mantem a tela aberta esperando a digitação de uma tecla
34.            Console.ReadKey();
35.        }
36.    }
37. }

```


3.13.2. Tela:



```
C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula03_Ex11\...
O valor é: 100
O valor é: 120
O primeiro é: 120, e o segundo é 130
O primeiro valor é      120, e o segundo é      130
O primeiro valor é 120      , e o segundo é 130
O primeiro valor é      120,00, segundo é      130,00
O primeiro valor é 120,000 ,segundo é 130,000

Produto      Quantidade      Valor Unitário      Valor Total
Lata          5,00          10,00              50,00
```

4. VARIÁVEIS TIPOS E UTILIZAÇÕES, CONCEITO DE ALIAS E CONSTANTES

4.1. Objetivo

Apresentar o conceito de variáveis

4.2. Introdução

Variáveis são atribuídos que serão manipulados durante a programação.

Uma variável possui sempre:

- Identificador
- Conteúdo

4.2.1. Identificador:

É o **nome** atribuído à variável pelo programador. Em princípio deve ser um nome simples que não seja igual a nenhuma das palavras já utilizadas pelo C# e deve ser facilitar a identificação da variável. As regras para nomear identificadores são as seguintes:

- Iniciar com uma letra.
- Pode utilizar letras e números.
- Não utilizar caracteres especiais.
- Não deixar espaço entre as letras.
- Não colocar letras ou palavras sublinhadas.
- Iniciar preferivelmente com letras minúsculas.
- Se representar duas palavras a segunda palavra deve iniciar com maiúscula.

Obs: Algumas destas regras são obrigatórias e se for tentado nomes fora do permitido, o C# não aceitará, outras fazem parte de convenções normalmente utilizadas por programadores e serão seguidas neste curso.

4.2.2. Conteúdo:

É o **valor** atribuído à variável durante a programação.

Este valor pode ser atribuído diretamente pelo programador, ou pode ser resultado de manipulações ou cálculos efetuados pelo programa durante sua execução.

Como na maioria das linguagens, os conteúdos das variáveis em C# podem ser do tipo:

4.2.2.1. *Numéricas:*

São variáveis que aceitam somente valores numéricos e estes podem sofrer operações aritméticas. Estes números podem ser:

- Inteiros
- Decimais ou com ponto flutuante

4.2.2.2. *Caracteres:*

São variáveis que aceitam letras e números se forem somente números não podem sofrer operações aritméticas.

4.2.2.3. *Booleano:*

São variáveis que aceitam somente dois valores **Verdadeiro (True)** ou **Falso (False)**.

4.3. Tipos das Variáveis em C#

Existem diversos tipos de variáveis, cabe ao programador escolher o tipo de variável que vai utilizar, dependendo dos valores que a variável pode receber e do tamanho que ela vai ocupar na memória. Na medida do possível, o programador deve utilizar a variável que atende suas necessidades com o menor tamanho possível de maneira a não ocupar mais memória do que realmente é necessário.

A tabela abaixo fornece os nomes, tipos, valores possíveis e tamanhos das variáveis em C#.

Nome C#	Tipo	Valores possíveis de armazenar	Tamanho
bool	Booleano	Verdadeiro ou Falso	8 bits
byte	Inteiro	0 a 255	8 bits

sbyte	Inteiro	-128 a 127	8 bits
ushort	Inteiro	0 a 65,535	16 bits
short	Inteiro	-32,768 a 32,767	16 bits
uint	Inteiro	0 a 4,294,967,295	32 bits
int	Inteiro	-2,147,483,648 a 2,147,483,647	32 bits
ulong	inteiro	0 a 18,446,744,073,709,551,615	64 bits
long	inteiro	9,223,372,036,854,775,808 a 9,223,372,036,854,775,807	64 bits
float	Real	$\pm 1.5 \times 10^{-45}$ a $\pm 3.4 \times 10^{38}$	32 bits
double	Real	$\pm 5.0 \times 10^{-324}$ a $\pm 1.7 \times 10^{308}$	64 bits
decimal	Real	$\pm 1.0 \times 10^{-28}$ a $\pm 7.9 \times 10^{28}$	128 bits
char	Caracter	Um caracter	16 bits
string	Caracter	Sequência de caracter (16 bits por caráter)	16 bits

Além da faixa de valores possíveis, a principal diferença entre as variáveis do tipo **float** **double** e **decimal** está na precisão, expressa pelo número de casas decimais conforme indicado na tabela abaixo.

Nome C#	Tipo	Valores possíveis de armazenar	Precisão	Tamanho
Float	Real	$\pm 1.5 \times 10^{-45}$ a $\pm 3.4 \times 10^{38}$	7 dígitos	32 bits
Double	Real	$\pm 5.0 \times 10^{-324}$ a $\pm 1.7 \times 10^{308}$	15 a 16 dígitos	64 bits
Decimal	Real	$\pm 1.0 \times 10^{-28}$ a $\pm 7.9 \times 10^{28}$	28 a 29 dígitos	128 bits

Outra importante diferença está na maneira como os dados são armazenados, pois dados do tipo float e double são armazenados na forma binária, enquanto que dados do tipo decimal, são armazenados na forma decimal. Isto faz com que, embora utilizando mais memória os dados na forma decimal ficam mais precisos.

4.4. Declaração de Variáveis

4.4.1. Como declarar:

Declarar variáveis consiste em informar ao programa que pretende utilizar uma variável e fornecer suas características em C#, basta informar o nome da variável e o identificador.

Em princípio uma variável pode ser declarada em qualquer local do programa, basta lembrar que antes de ser utilizada a variável precisa ser declarada. **Entretanto por questões de estética e organização a grande maioria dos programadores declaram as variáveis logo no início do programa.**

Neste curso as variáveis serão declaradas sempre no início do programa precedido do comentário “Declaração de Variáveis”

4.4.2. Sintaxe:

```
// Declaração de Variáveis
```

```
byte n1;
```

4.4.3. Como declarar diversas variáveis do mesmo tipo:

É possível declarar mais de uma variável em uma mesma expressão, desde que elas sejam do mesmo tipo.

4.4.4. Sintaxe:

```
// Declaração de Variáveis
```

```
byte n1, n2, n3, a1, a2;
```

4.5. Atribuição de valores a variáveis

4.5.1. Como atribuir valores:

Após declarada, é possível atribuir valores a uma variável. A atribuição de valores é feita informando o identificador da variável, o sinal de igual e o valor que será atribuído à variável.

4.5.2. Sintaxe:

```
// Atribuir valores
```

```
N1 = 10;
```

4.6. Declaração e atribuição simultânea

É possível atribuir valor à variável no mesmo momento que ela é declarada, para fazê-lo basta obedecer à sintaxe abaixo:

4.6.1. Sintaxe:

```
// Declaração de Variáveis
```

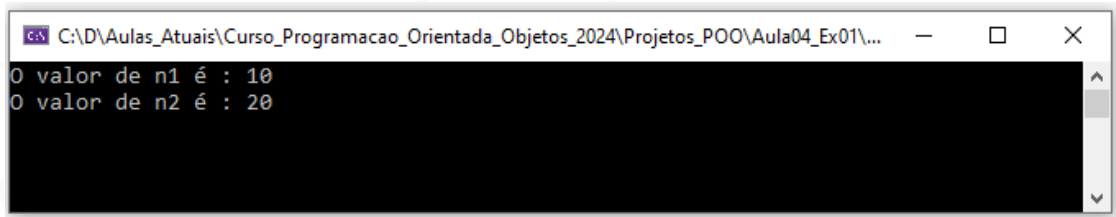
```
byte n2 = 20;
```

4.6.2. Exemplo Variável Inteira:

O programa abaixo declara duas variáveis inteiras do tipo byte e em seguida os mostra com instruções Write.

```
1. namespace Aula04_Ex01
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declaração de Variáveis do tipo Inteiro
8.             byte n1;
9.             byte n2 = 20;
10.
11.             // Atribui valor
12.             n1 = 10;
13.
14.             // Mostra os valores
15.             Console.WriteLine("O valor de n1 é : " + n1);
16.             Console.WriteLine("O valor de n2 é : " + n2);
17.
18.             // Mantem a tela aberta esperando a digitação de uma tecla
19.             Console.ReadKey();
20.         }
21.     }
22. }
```

4.6.3. Tela:



```
C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula04_Ex01\...
O valor de n1 é : 10
O valor de n2 é : 20
```

4.7. Variável Real do tipo float:

O programa abaixo declara duas variáveis reais do tipo float e em seguida os mostra com instruções Write.

(MICROSOFT, 2010) Por padrão, um literal numérico real no lado direito do operador de atribuição é tratado como double. Portanto, para inicializar uma variável float, use o sufixo `f` ou `F`,

4.7.1. Sintaxe

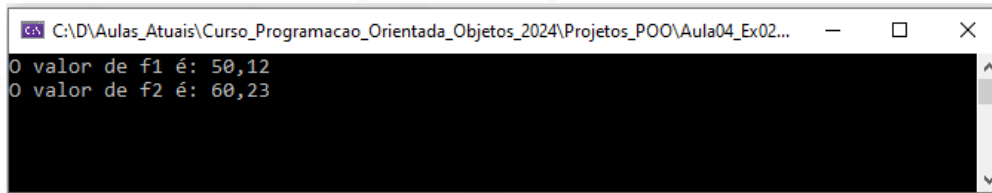
`float Identificador_da_variável;` *// cria a variável sem atribuir valor*

`float Identificador_da_variável = valor numérico;` *// cria a variável e atribui um valor*

4.7.2. Exemplo

```
1. namespace Aula04_Ex02
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declaração de Variáveis do tipo float
8.             float f1;
9.             float f2 = 60.229F;
10.
11.            // Atribui valor
12.            f1 = 50.1249F;
13.
14.            // Mostra os valores
15.            Console.WriteLine("O valor de f1 é: {0:F2} ", f1);
16.            Console.WriteLine("O valor de f2 é: {0:F2} ", f2);
17.            Console.ReadKey();
18.        }
19.    }
20. }
```


4.7.3. Tela



Nota: Ao atribuir valores do tipo float, é **obrigatório o uso do sufixo F**. Caso o F seja omitido o C# tentará utilizar a variável como double, mas o compilador apresentará um erro pois a variável foi declarada como float. **Por este motivo, como será observado a seguir, somente variáveis reais do tipo double não precisam de sufixo.**

4.8. Variável Real do tipo double

O programa abaixo declara duas variáveis reais do tipo Double e em seguida os mostra com instruções Write.

4.8.1. Sintaxe

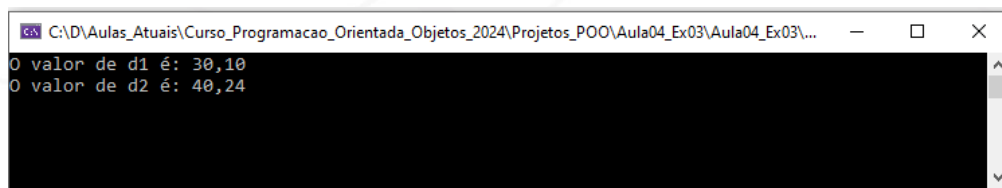
`double Identificador_da_variável; // cria a variável sem atribuir valor`

`double Identificador_da_variável = valor numérico; // cria a variável e atribui um valor`

4.8.2. Exemplo:

```
1. namespace Aula04_Ex03
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declaração Variáveis do tipo double
8.             double d1;
9.             double d2 = 40.238;
10.
11.             // Atribui valor
12.             d1 = 30.1;
13.
14.             // Mostra os valores
15.             Console.WriteLine("O valor de d1 é: {0:F2} ", d1);
16.             Console.WriteLine("O valor de d2 é: {0:F2} ", d2);
17.
18.             // Mantem a tela aberta esperando a digitação de uma tecla
19.             Console.ReadKey();
20.         }
21.     }
22. }
```

4.8.3. Tela



```

C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula04_Ex03\Aula04_Ex03\...
O valor de d1 é: 30,10
O valor de d2 é: 40,24
  
```

4.9. Variável Real do tipo decimal

O programa abaixo declara duas variáveis reais do tipo Decimal e em seguida os mostra com instruções Write.

(MICROSOFT, 2010) A palavra-chave decimal indica um tipo de dados de 128-bits. Comparado aos tipos de ponto flutuante, o tipo decimal tem mais precisão e um intervalo pequeno, o que torna apropriado para cálculos financeiros e monetários.

(MICROSOFT, 2010) Se você quiser que um numérico real literal seja tratado como decimal, use o sufixo m ou M, como no exemplo

4.9.1. Sintaxe

decimal Identificador_da_variável; *(cria a variável sem atribuir valor)*

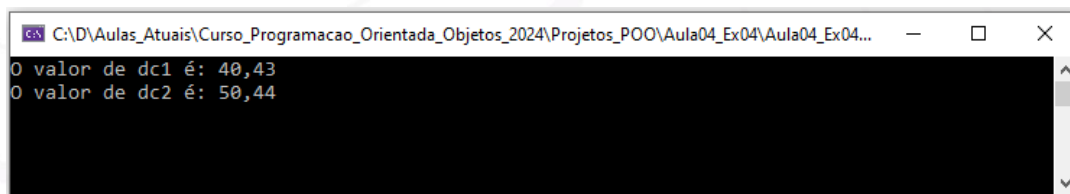
decimal identficador_da_variável = valor numéricoM; *(cria a variável e atribui um valor)*

4.9.2. Exemplo

```

1. namespace Aula04_Ex04
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declaração Variáveis do tipo decimal
8.             decimal dc1;
9.             decimal dc2 = 50.4367M;
10.
11.            // Atribui valor
12.            dc1 = 40.4348M;
13.
14.            // Mostra os valores
15.            Console.WriteLine("O valor de dc1 é: {0:F2} ", dc1);
16.            Console.WriteLine("O valor de dc2 é: {0:F2} ", dc2);
17.
18.            // Mantem a tela aberta esperando a digitação de uma tecla
19.            Console.ReadKey();
20.        }
21.    }
22. }
  
```

4.9.3. Tela



Nota: A exemplo do tipo float, o tipo Decimal também obriga a utilização de sufixo que no caso deve ser a letra M. Este sufixo não pode ser omitido pois caso ele não esteja presente o compilador apresentará erro.

4.10. Variáveis do tipo char e string

Variáveis do tipo char e string aceitam caracteres. Mesmo que sejam fornecidos números para seus conteúdos, eles serão utilizados como caracteres.

Variáveis tipo char só aceitam um caracter, pode ser interessante para respostas do tipo “S”, “N”; “V”, “F” etc.

Quando se atribui valores a uma variável tipo char, estes devem ser atribuído entre aspas simples.

Quando se atribui valores a variáveis tipo string estes devem ser atribuídos entre aspas duplas.

4.10.1. Sintaxe

```
char Identificador_da_variável; // cria a variável tipo char
string Identificador_da_variável; // cria a variável do tipo string
```

4.10.2. Exemplo:

```
1. namespace Aula04_Ex05
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis do tipo Char e String
8.             char c1;
9.             string c2;
```

```

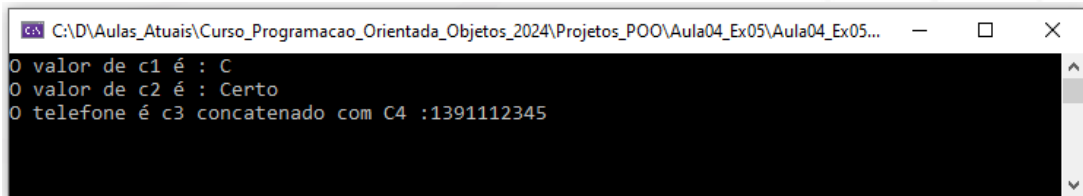
10.         string c3, c4;
11.
12.         // Atribui valores
13.         c1 = 'C';
14.         c2 = "Certo";
15.         c3 = "13";
16.         c4 = "91112345";
17.
18.         // Mostra os valores
19.         Console.WriteLine("O valor de c1 é : " + c1);
20.         Console.WriteLine("O valor de c2 é : " + c2);
21.         Console.WriteLine("O telefone é c3 concatenado com C4 : " + c3 + c4);
22.
23.         // Mantem a tela aberta esperando a digitação de uma tecla
24.         Console.ReadKey();
25.     }
26. }
27. }

```

Nota: É importante observar como as variáveis são string, na linha “O telefone é C3 concatenado com C4 :” resulta nos dois números um após o outro e não a soma.

Nota: Conforme pode ser visto na declaração string c3, c4, é possível criar diversas variáveis do mesmo tipo somente separando os indicadores por vírgula

4.10.3. Tela



4.11. Variável do tipo booleano

Variável do tipo booleano pode assumir somente dois valores True (Verdadeiro) ou False (Falso).

4.11.1. Sintaxe

```
bool identificador;
```

4.11.2. Exemplo

```

1. namespace Aula04_Ex06
2. {
3.     internal class Program
4.     {

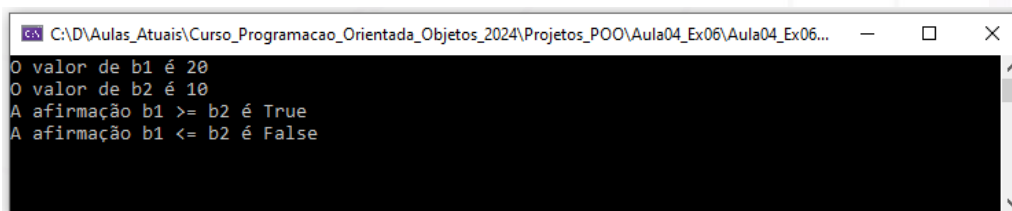
```

```

5.      static void Main(string[] args)
6.      {
7.          // Declaração de Variáveis
8.          byte b1, b2;
9.          bool compara;
10.
11.         // Atribuição de valores
12.         b1 = 20;
13.         b2 = 10;
14.         // Mostrar valores
15.         Console.WriteLine("O valor de b1 é " + b1);
16.         Console.WriteLine("O valor de b2 é " + b2);
17.
18.         // Atribui valores
19.         compara = b1 >= b2;
20.         // Mostrar valores
21.         Console.WriteLine("A afirmação b1 >= b2 é " + compara);
22.
23.         // Atribui valores
24.         compara = b1 <= b2;
25.         // Mostrar valores
26.         Console.WriteLine("A afirmação b1 <= b2 é " + compara);
27.
28.         // Mantém a tela aberta aguardando a digitação de uma tecla
29.         Console.ReadKey();
30.     }
31. }
32. }

```

4.11.3. Tela



4.12. Informações adicionais:

4.12.1. Alias:

Esta palavra que é utilizada em diversos momentos no ramo de informática, representa na verdade uma apelido que se dá a algo, ou seja um segundo nome. Este alias normalmente tende a ser um nome mais simples para referenciar algo que tem um nome ou endereço complicado ou muito grande.

4.12.2. Tipo de variável com letra minúscula ou maiúscula.

Ao declarar uma variável, você já deve ter reparado que o “Menu suspenso auto completar” sugere os tipos iniciando com letras minúsculas e letra maiúsculas (byte e Byte;

double e Double, ... etc.). na verdade elas representam o mesmo tipo de dados, de modo que o código resultante é idêntico, veja a definição abaixo:

(CERTIFICATION, 2008) Minúsculas "byte" é construído em um tipo em C#. System.Byte é uma classe construída no frame .NET que representa um byte. O segredo é que o tipo "byte" é construído em um alias para a classe "System.Byte". O .NET framework possui aliases diferentes baseados na semântica da linguagem particular, mas estes mapeiam todos os mesmos tipos de objetos específicos do .NET framework. Isso permite que o código escrito fique mais limpo e mais fácil de compilar para o tipo correto do .NET framework.

A diferença é que byte com "b" minúsculo é um tipo do C#, enquanto que Byte com "B" maiúsculo é uma referência à classe do .NET

O mesmo acontece com os demais tipos de variáveis como float e Float, double e Double, string e String, etc.

O programa abaixo demonstra estas informações, pois declara variáveis do tipo byte e Byte e do tipo string e String. Em seguida ele compara os conteúdos e mostra que eles são diferentes, mas ao comparar os tipos mostra que eles são iguais. O programa pode ser repetido para outros tipos de variáveis, como float e Float, double e Double, string e String, etc.

4.12.2.1. Sintaxes

A instrução `Indicador_da_variável.GetType()`, fornece o tipo de variável

4.12.2.2. Exemplo:

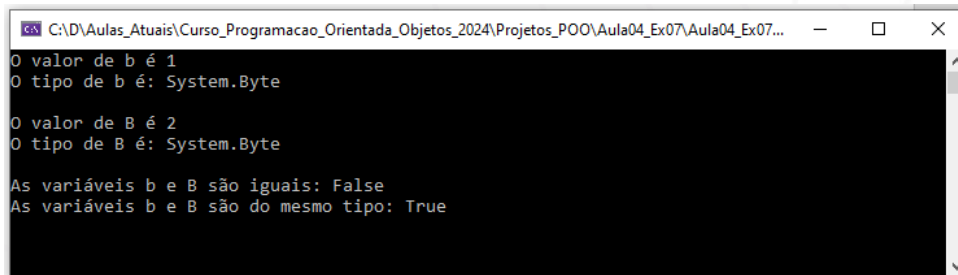
```
1. namespace Aula04_Ex07
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // declaração de variáveis iniciando com minúscula e maiúscula
8.             byte b = 1;
9.             Byte B = 2;
10.
11.             // variáveis booleanas para executar a comparação
12.             bool comparavariaveis;
13.             bool comparatipos;
14. }
```

```

15.         // Mostra os valores e os tipos de variáveis
16.         Console.WriteLine("O valor de b é " + b);
17.         Console.WriteLine("O tipo de b é: " + b.GetType());
18.         Console.WriteLine("\nO valor de B é " + B);
19.         Console.WriteLine("O tipo de B é: " + B.GetType());
20.
21.         // Compara as variáveis e os tipos de variáveis
22.         comparavariaveis = b == B;
23.         comparatipos = b.GetType() == B.GetType();
24.         Console.WriteLine("\nAs variáveis b e B são iguais: " +
comparavariaveis);
25.         Console.WriteLine("As variáveis b e B são do mesmo tipo: " +
comparatipos);
26.
27.         // Mantém a tela aberta aguardando a digitação de uma tecla
28.         Console.ReadKey();
29.
30.     }
31. }
32. }

```

4.12.2.3. Tela



4.13. Declaração de constante

Caso se deseje que uma variável receba um valor inicial e não possa ser alterada durante a programação ela pode ser declarada como constante. Para declarar uma constante basta colocar antes do tipo de variável a palavra `const` conforme sintaxe a seguir:

4.13.1. Sintaxe

```
const byte valor = 10;
```


5. OPERADORES ARITMÉTICOS E CONVERSÃO DE VARIÁVEIS – Read E ReadLine

5.1. Objetivo

Apresentar os operadores aritméticos, discutir conversões de variáveis e os comandos Read e ReadLine.

5.2. Introdução

Operadores Aritméticos são símbolos que representam as operações aritméticas, normalmente utilizadas na programação de computadores.

É importante observar que, além das 4 operações básicas é muito comum utilizar em programas de computação **números elevados a expoentes**, **raiz quadrada de números** e operações que fornecem o **resto** e o **cociente** da divisão inteira.

O resto e o cociente de divisões são operações também importantes, não só pelos resultados, mas também para propiciar alguns tipos de análise, tais como: verificar se a divisão é exata (resto da divisão = 0); obter a parte inteira de uma divisão; saber se um número é par (divisível por 2 = resto da divisão por 2 resulta em 0).

As operações Aritméticas utilizadas são as indicadas na tabela a seguir:

Operação	Função	Significado	Sintaxe em C#
+	Adição	Soma valores	$a + b$
-	Subtração	Subtrai valores	$a - b$
*	Produto	Multiplica valores	$a * b$
/	Divisão	Divide valores	a / b
%	Resto	Resto da divisão	$a \% b$
++	Incremento	Acrescenta 1 ao valor	$a++$ ou $++a$
--	Decremento	Subtrai 1 ao valor	$a--$ ou $--a$

Notas: O C# possui a operação %, que fornece o resto da divisão, mas não possui exatamente um operador para obter a parte inteira da divisão. Caso se deseje a parte

inteira, basta declarar os valores como inteiros. O exemplo Ex10 demonstra o operador % e a obtenção da parte inteira da divisão, assim como a maneira de obter o resultado real da divisão entre números inteiros.

Os operadores de incremento e decremento funcionam da seguinte maneira:

Se o incremento for colocado após a variável (a++), o incremento será feito depois de usar o valor da variável na expressão.

Se o incremento for colocado antes da variável (++a), o incremento será feito antes do uso do valor na expressão.

5.3. Precedência de Operações

No momento de efetuar as expressões aritméticas, os softwares obedecem às precedências indicadas na tabela a seguir.

Estas precedências, são relativamente simples de entender, entretanto, são muito perigosas no momento da escrituração de fórmulas para serem resolvidas por programas de computadores.

Uma excelente maneira de garantir que as operações sejam executadas da maneira como se deseja é a utilização de parênteses, portanto, o conselho é que, em caso de dúvidas, sejam utilizados parentes, visto que, parentes em excesso, desde que colocados corretamente, não prejudicam as operações, mas uma operação executada de maneira diferente do que o programador pretende, sem dúvida irá colocar a perder todo o trabalho de programação.

Prioridade	Operadores
1	Parênteses
2	Potenciação e Radiciação
3	% * /
4	+ -

5.4. Comando Read

Este comando interrompe o fluxo do programa e aguarda que o usuário digite algo e encerre com a tecla <Enter>.

Ao receber a informação da tecla <Enter>, o comando atribui o **código decimal do primeiro caracter digitado** à variável cujo identificador foi colocado antes do comando Read.

5.4.1. Sintaxe:

```
foidigitado = Console.Read();
```

Notas:

- 1) É importante lembrar que **este comando recebe somente o primeiro caracter digitado**. Uma das utilidades deste comando pode ser garantir que o programa irá receber somente um caracter, mesmo que o usuário digite diversos caracteres.
- 2) **O código decimal deste caracter é atribuído à variável cujo identificador é colocado antes do sinal de igual** (no exemplo a variável **foidigitado**).
- 3) O valor que é atribuído à variável (**foi digitado no exemplo**) **é um código decimal, portanto um valor numérico**, que **precisa ser declarado como int**.

Obs: A tabela ASCII com caracteres e a referência decimal pode ser encontrada em:

<https://web.fe.up.pt/~ee96100/projecto/Tabela%20ascii.htm>

5.5. Conversão de um valor numérico para caracter

Caso se tenha o valor decimal de um caracter da tabela ASCII e se deseje saber qual este caracter, basta fazer a conversão com o comando Convert.ToChar, com sintaxe conforme a seguir:

5.5.1. Sintaxe:

```
referenciaAscii = Convert.ToChar(codigoDecimal);
```

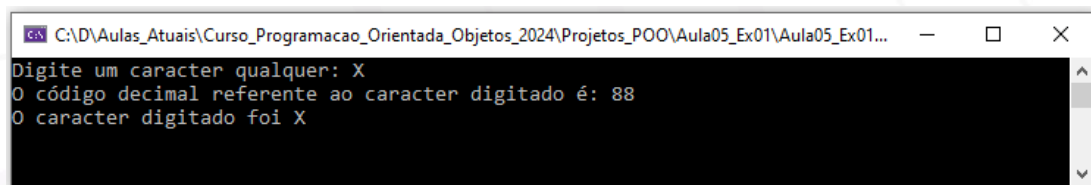
Notas:

- 1) A variável “referenciaAscii” da sintaxe acima deve ser declarada como caracter.
- 2) A variável “codigoDecimal” da sintaxe acima deve ser declarada como int.

5.5.2. Exemplo: (como é Read() só recebe o primeiro caracter digitado)

```
1. namespace Aula05_Ex01
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // declaração de variáveis
8.             int codDec;
9.             char referencia;
10.
11.             // Recebe valor
12.             Console.WriteLine("Digite um caracter qualquer: ");
13.             codDec = Console.Read();
14.
15.             // Converte o valor recebido para caracter
16.             referencia = Convert.ToChar(codDec);
17.
18.             Console.WriteLine("O código decimal referente ao caracter digitado é:
19. " + codDec);
20.             Console.WriteLine("O caracter digitado foi " + referencia);
21.
22.             // Mantem a tela aguardando a digitação de uma tecla
23.             Console.ReadKey();
24.         }
25. }
```

5.5.3. Tela



5.6. Comando ReadLine

Este comando interrompe o fluxo do programa e aguarda que o usuário digite algo e encerre com a tecla <Enter>.

Ao receber a informação da tecla <Enter> o comando atribui o **conteúdo digitado** à variável cujo identificador foi colocado antes do comando ReadLine.

5.6.1. Sintaxe:

```
foidigitado = Console.ReadLine();
```

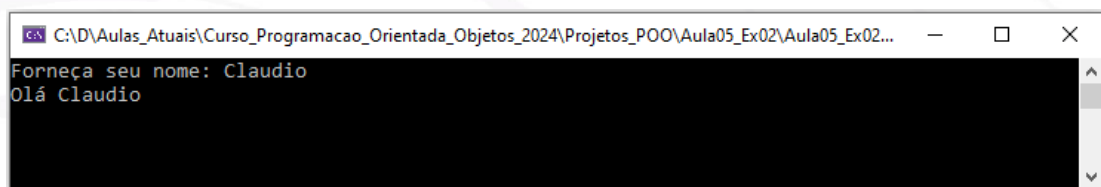
Notas:

- 1) Este comando **recebe todos os caracteres digitados** pelo usuário.
- 2) Os caracteres digitados são atribuídos à variável cujo identificador é colocado antes do sinal de igual (*no exemplo a variável **foidigitado***).
- 3) O valor que é atribuído à variável (**foidigitado** no exemplo) é sempre uma string, portanto a variável deve ser declarada como string.

5.6.2. Exemplo (como é ReadLine, recebe todos os caracteres uma variável string)

```
1. namespace Aula05_Ex02
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis
8.             string nome;
9.
10.            // Recebe o nome
11.            Console.Write("Forneça seu nome: ");
12.            nome = Console.ReadLine();
13.
14.            // Informa os valores
15.            Console.WriteLine("Olá {0}", nome);
16.
17.            // Mantem a tela aberta esperando a digitação de uma tecla
18.            Console.ReadKey();
19.        }
20.    }
21. }
```

5.6.3. Tela:



5.7. Como receber um conteúdo do tipo char

Como o comando **ReadLine** recebe sempre conteúdo do tipo string, caso a variável à qual se deseja atribuir um conteúdo digitado pelo usuário seja do tipo char, é necessário

converter o que foi digitado para char antes de atribuir o conteúdo ao identificador da variável.

Caso a conversão não seja feita o programa não será executado.

5.7.1. Sintaxe:

Nesta primeira parte do curso a conversão será feita com o comando **Parse**, portanto a sintaxe ficará conforme a seguir:

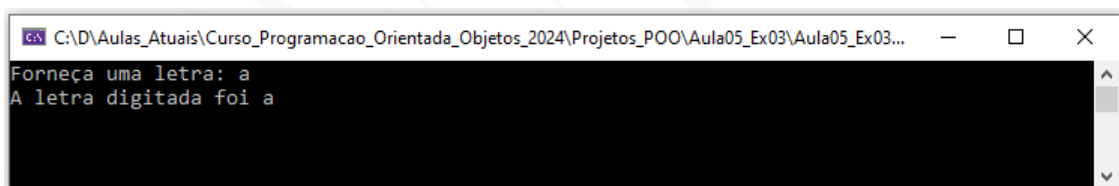
```
foidigitado = char.Parse(Console.ReadLine());
```

Obs: O comando `char.Parse`, converte a variável string recebida no comando `ReadLine` em uma variável do tipo `char`.

5.7.2. Exemplo recebendo uma variável char:

```
1. namespace Aula05_Ex03
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis
8.             char letra;
9.
10.            // Recebe uma letra
11.            Console.Write("Forneça uma letra: ");
12.            letra = char.Parse(Console.ReadLine());
13.
14.            // Informa os valores
15.            Console.WriteLine("A letra digitada foi {0}", letra);
16.
17.            // Mantem a tela aberta esperando a digitação de uma tecla
18.            Console.ReadKey();
19.
20.        }
21.    }
22. }
```

5.7.3. Tela



5.8. Como receber somente um caractere

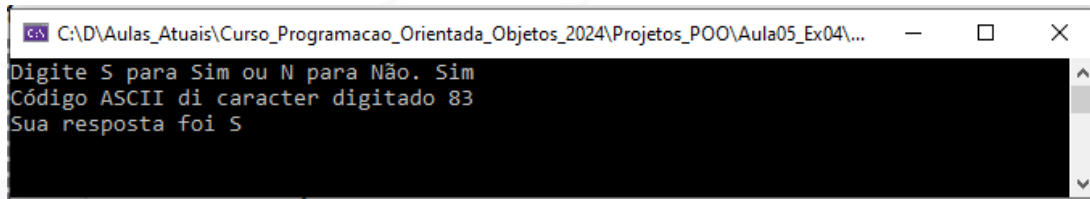
Conforme pode ser observado no exemplo anterior, caso o usuário digite mais de um caractere em uma resposta que será lida pelo comando `ReadLine`, o programa apresentará uma mensagem de erro no momento da execução, pois ele está esperando um conteúdo do tipo `char` (somente um caractere). Este tipo de erro, que ocorreu em tempo de execução, chamado por muitos livros de “Runtime error”, não é um erro de compilação, pois o programa pode funcionar corretamente, porém, indica que, o programador não tomou o devido cuidado para evitar que o programa apresentasse um erro devido a uma digitação descuidada do usuário.

Uma das maneiras de evitar este tipo de erro é utilizar o comando `Read`, que pega somente o primeiro caractere digitado convertido para seu valor decimal conforme tabela ASCII e em seguida o converte para o caractere equivalente, conforme exemplo a seguir.

5.8.1. Exemplo

```
1. namespace Aula05_Ex04
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis
8.             int res;
9.             char resposta;
10.
11.             // Recebe digitação - com comando Read e converte para decimal da
ASCII
12.             Console.WriteLine("Digite S para Sim ou N para Não. ");
13.             res = Console.Read();
14.             // Converte o valor recebido para caractere
15.             resposta = Convert.ToChar(res);
16.
17.             // O código ASCII do primeiro caractere digitado é:
18.             Console.WriteLine("Código ASCII di caractere digitado {0} ", res);
19.
20.             // Informa a primeira letra digitada
21.             Console.WriteLine("Sua resposta foi {0} ", resposta);
22.
23.             // Mantem a tela aberta esperando a digitação de uma tecla
24.             Console.ReadKey();
25.         }
26.     }
27. }
```


5.8.2. Tela



Obs: Conforme pode ser observado, embora tenham sido digitados diversos caracteres “Sim”, à variável resposta foi atribuído o valor “S”. Essa atribuição foi feita pelo comando `Confert.ToChar(res)`

5.9. Como receber um número inteiro:

Como o comando **ReadLine** recebe sempre variáveis do tipo **string**, caso a variável à qual se deseja atribuir um conteúdo digitado pelo usuário seja do tipo inteiro (byte, sbyte, ushort, short, uint, int, ulong, long), é necessário converter o que foi digitado para um dos tipos inteiros antes de atribuir o conteúdo ao identificador da variável.

Caso a conversão não seja feita o programa não será executado.

5.9.1. Sintaxe:

```
foi digitado = int.Parse(Console.ReadLine());
```

Obs: O comando `int.Parse`, converte a variável `string` recebida no comando `ReadLine` em uma variável do tipo `int`.

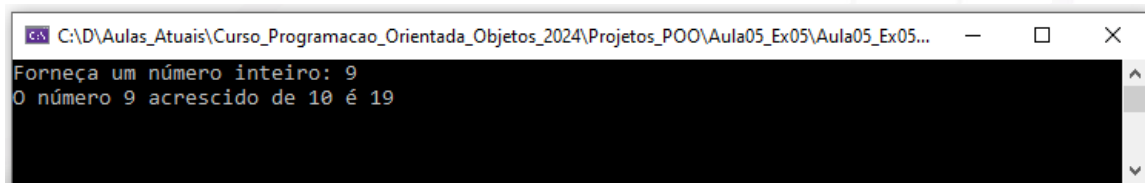
5.9.2. Exemplo recebendo uma variável `int`:

```
1. namespace Aula05_Ex05
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis inteira
8.             int ni, resInt;
9.
10.            // Recebe um valor
11.            Console.WriteLine("Forneça um número inteiro: ");
12.            ni = int.Parse(Console.ReadLine());
13.
14.            // Atribui valor
15.            resInt = ni + 10;
```

```
16.  
17.         // Informa os valores  
18.         Console.WriteLine("O número {0} acrescido de 10 é {1}", ni, resInt);  
19.  
20.         // Mantem a tela aberta esperando a digitação de uma tecla  
21.         Console.ReadKey();  
22.  
23.     }  
24. }  
25. }
```

Nota: Caso o usuário digite caracteres ao invés de números o programa apresentará erro em tempo de execução. Para resolver este problema pode-se utilizar a instrução `int.TryParse(Console.ReadLine(), out variável)`. Entretanto esta utilização necessita de uma interpretação do valor atribuído à variável, pois ele será zero (0) se o caracter fornecido não for numérico. Como, os comandos necessários para este tratamento serão vistos somente a partir da próxima aula, estas implementações só serão feitas futuramente.

5.9.3. Tela



5.10. Como receber um número Real

Como o comando `ReadLine` recebe sempre variáveis do tipo `string`, caso a variável à qual se deseja atribuir um conteúdo digitado pelo usuário seja do tipo real (`float`, `double`, `decimal`), é necessário converter o que foi digitado para um dos tipos reais antes de atribuir o conteúdo ao identificador da variável.

Caso a conversão não seja feita o programa não será executado.

5.10.1. Sintaxe para converter para `double`:

```
foidigitado = double.Parse(Console.ReadLine());
```

Obs: O comando `double.Parse`, converte a variável `string` recebida no comando `ReadLine` em uma variável do tipo `double`.

5.10.2. Exemplo recebendo uma variável double:

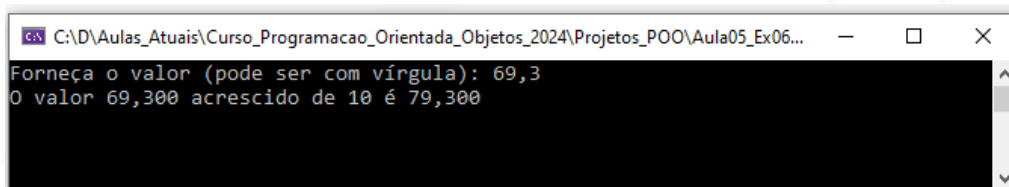
```

1. namespace Aula05_Ex06
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis double
8.             double nd, resDouble;
9.
10.            // Recebe um valor
11.            Console.WriteLine("Forneça o valor (pode ser com vírgula): ");
12.            nd = double.Parse(Console.ReadLine());
13.
14.            // Atribui valor
15.            resDouble = nd + 10;
16.
17.            // Informa os valores
18.            Console.WriteLine("O valor {0:F3} acrescido de 10 é {1:F3}", nd,
19.            resDouble);
20.
21.            // Mantem a tela aberta esperando a digitação de uma tecla
22.            Console.ReadKey();
23.        }
24.    }

```

Nota: O número e o resultado serão apresentados com 3 casas decimais utilizando a instrução F3.

5.10.3. Tela



5.10.4. Sintaxe para converter para float:

```
foidigitado = float.Parse(Console.ReadLine());
```

Obs: O comando float.Parse, converte a variável string recebida no comando ReadLine em uma variável do tipo float.

5.10.5. Exemplo recebendo uma variável float:

```

1. namespace Aula05_Ex07
2. {
3.     internal class Program
4.     {

```

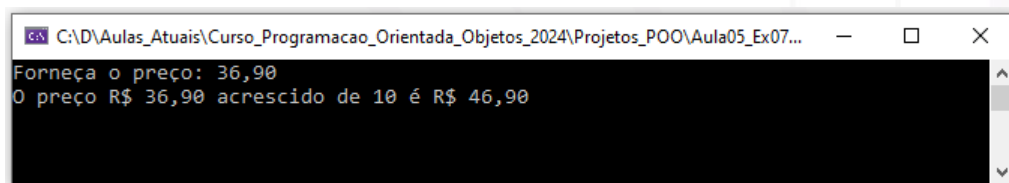
```

5.      static void Main(string[] args)
6.      {
7.          // Declara Variáveis float
8.          float nf, resFloat;
9.
10.         // Recebe um valor
11.         Console.Write("Forneça o preço: ");
12.         nf = float.Parse(Console.ReadLine());
13.
14.         // Atribui valor
15.         resFloat = nf + 10;
16.
17.         // Informa os valores
18.         Console.WriteLine("O preço {0:C} acrescido de 10 é {1:C}", nf,
19.         resFloat);
20.
21.         // Mantem a tela aberta esperando a digitação de uma tecla
22.         Console.ReadKey();
23.     }
24. }

```

Nota: O C força o resultado do tipo moeda (com R\$ e duas casas após a vírgula)

5.10.6. Tela



5.10.6.1. Observação com relação à atribuição sem F

Conforme pode ser observado, na linha “`resFloat = nf + 10;`”, não foi colocado a letra F após o 10 conforme no exemplo da aula anterior e a operação debug, não apresentou o erro “*Cannot implicitly convert type ‘double’ to ‘float’. No explicit conversion exist (are you missing a cast?)*”. Isto aconteceu porque o valor 10 é inteiro. Caso se desejasse utilizar um valor fracionário o sufixo F deveria ser acrescido.

5.10.7. Sintaxe para converte para decimal:

```
foidigitado = decimal.Parse(Console.ReadLine());
```

Obs: O comando `decimal.Parse`, converte a variável string recebida no comando `ReadLine` em uma variável do tipo decimal.

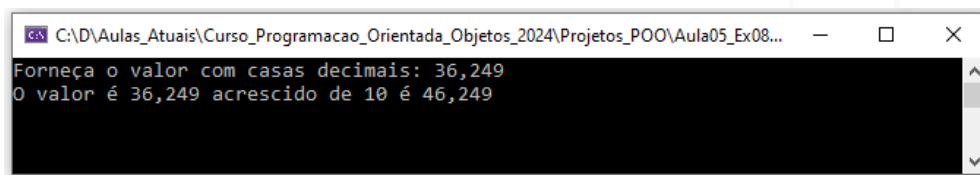
5.10.8. Exemplo recebendo uma variável decimal:

```

1. namespace Aula05_Ex08
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis decimal
8.             decimal nd, resDecimal;
9.
10.            // Recebe um valor
11.            Console.WriteLine("Forneça o valor com casas decimais: ");
12.            nd = decimal.Parse(Console.ReadLine());
13.
14.            // Atribui valor
15.            resDecimal = nd + 10;
16.
17.            // Informa os valores
18.            Console.WriteLine("O valor é {0} acrescido de 10 é {1}", nd,
19.            resDecimal);
20.
21.            // Mantem a tela aberta esperando a digitação de uma tecla
22.            Console.ReadKey();
23.        }
24.    }

```

5.10.9. Tela



5.10.9.1. Observação com relação à atribuição sem M

Conforme pode ser observado, na linha “`resDecimal = nd + 10;`”, não foi colocado a letra M após o 10 conforme no exemplo da aula anterior e a operação debug, não apresentou erro.

É importante ficar atento, pois assim como no exemplo anterior (utilizando variável tipo float), o erro não ocorreu porque o valor 10 é inteiro, caso se utilizasse valores fracionários o sufixo M, (assim como o F do exemplo anterior) torna-se obrigatório.

5.11. Conversão de conteúdo

Algumas vezes pode ser necessário converter conteúdo dentro de um programa para que seja possível operar com estes valores. Isto pode ser feito com uma sintaxe de atribuição mesclada com conversão que também pode simultaneamente estar declarando

a variável. O programa a seguir fornece exemplos destas declarações com atribuição e conversão simultânea.

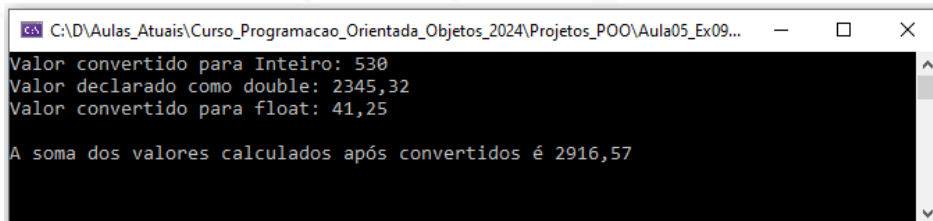
5.11.1. Exemplo:

```

1. namespace Aula05_Ex09
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // declara e inicializa variáveis do tipo string
8.             string valor_string1 = "530";
9.             string valor_string2 = "2345,32";
10.            string valor_string3 = "41,25";
11.            // declara variável para pegar o resultado
12.            double resultado;
13.
14.            // converte string para inteiro (declara também a variável int)
15.            int valor_integer = int.Parse(valor_string1);
16.            // exibe o resultado
17.            Console.WriteLine("Valor convertido para Inteiro: {0}",
18.                valor_integer);
19.
20.            // converte string para double (declara também a variável double)
21.            double valor_double = double.Parse(valor_string2);
22.            // exibe o resultado
23.            Console.WriteLine("Valor declarado como double: {0}", valor_double);
24.
25.            // converte string para float (declara também a variável float)
26.            float valor_float = float.Parse(valor_string3);
27.            // exibe o resultado (o \n antes de fechar as aspas acrescenta uma
28.            // linha em branco)
29.            Console.WriteLine("Valor convertido para float: {0} \n", valor_float);
30.
31.            // Calcula e fornece o resultado
32.            resultado = valor_integer + valor_double + valor_float;
33.            Console.WriteLine("A soma dos valores calculados após convertidos é
34.                {0} ", resultado);
35.
36.            //Mantem a tela aberta esperando a digitação de uma tecla
37.            Console.ReadKey();
38.        }
39.    }
40. }

```

5.11.2. Tela



Nota: Observar que os valores atribuídos nos momentos das declarações das variáveis do tipo string que posteriormente serão convertidas para números reais, têm suas partes decimais separadas por vírgulas, isto se dá, devido ao fato destes valores estarem no mesmo status de digitações do teclado, que são assimiladas no idioma do sistema operacional do usuário.

5.12. Divisão de Números declarados como inteiros

No C#, cada tipo de dados possui seus próprios operadores, isto significa que, quando se divide dois números inteiros, o resultado será sempre a parte inteira da divisão.

Caso se deseje que o resultado seja um número Real (do tipo float ou double), é melhor que os valores que serão divididos, sejam também números reais, entretanto, caso por qualquer razão os valores que serão divididos precisarem ser declarados como inteiros e os resultados das divisões precisarem ser números reais, é possível no momento da divisão informar o compilador, que o resultado da divisão deve ser tratado como real no momento da divisão, isto é feito com a sintaxe a seguir.

5.12.1. Sintaxes:

```
double(resultado) = double(dividendo)/double(divisor);
```

```
float(resultado) = float(dividendo)/float(divisor);
```

Obs: Não é necessário colocar a instrução (double ou float) em todos os identificadores, basta colocar somente em um dos identificadores que o compilador irá entender.

5.12.2. Exemplo:

```
1. namespace Aula05_Ex10
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara variaveis
8.             int v1, v2, restoInt, divInt;
9.             float restoReal, divReal;
10.
11.            // Solicita valores
12.            Console.Write("Forneça o primeiro valor: ");
13.            v1 = int.Parse(Console.ReadLine());
```

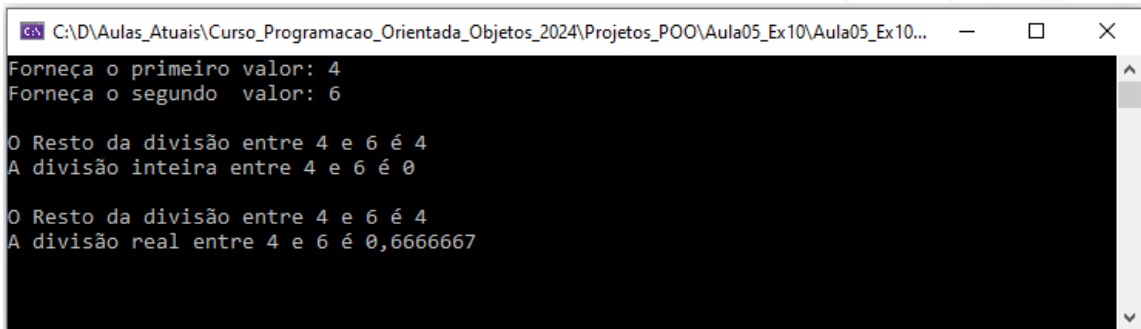


```

14.
15.         Console.WriteLine("Forneça o segundo valor: ");
16.         v2 = int.Parse(Console.ReadLine());
17.
18.         restoInt = v1 % v2;
19.         divInt = v1 / v2;
20.         restoReal = (float)v1 % (float)v2;
21.         divReal = (float)v1 / (float)v2;
22.         // Fornece os valores
23.         Console.WriteLine();
24.         Console.WriteLine("O Resto da divisão entre {0} e {1} é {2} ", v1, v2,
    restoInt);
25.         Console.WriteLine("A divisão inteira entre {0} e {1} é {2} ", v1, v2,
    divInt);
26.         Console.WriteLine();
27.         Console.WriteLine("O Resto da divisão entre {0} e {1} é {2} ", v1, v2,
    restoReal);
28.         Console.WriteLine("A divisão real entre {0} e {1} é {2} ", v1, v2,
    divReal);
29.
30.         // Mantem a tela aberta esperando a digitação de uma tecla
31.         Console.ReadKey();
32.     }
33. }
34. }

```

5.12.3. Tela



```

C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula05_Ex10\Aula05_Ex10...
Forneça o primeiro valor: 4
Forneça o segundo valor: 6

O Resto da divisão entre 4 e 6 é 4
A divisão inteira entre 4 e 6 é 0

O Resto da divisão entre 4 e 6 é 4
A divisão real entre 4 e 6 é 0,6666667

```

6. OPERADORES RELACIONAIS E LÓGICOS – ESTRUTURAS DE DECISÃO

6.1. Objetivo

Apresentar os operadores relacionais e lógicos e as estruturas de decisão.

6.2. Introdução

Muitas vezes em um programa é necessário decidir entre diferentes grupos de comandos a serem executados.

Estas decisões podem ser causadas por resultados obtidos em atribuições feitas pelo próprio programa, ou ainda respostas fornecidas pelos usuários. Como exemplo, o programa pode escrever “Aprovado” para um aluno cujo cálculo de sua média resultou em um valor igual ou superior a 7, ou escrever “Em Exame” para aquele que obteve média inferior a 7.

Estas decisões podem ser feitas basicamente com as estruturas:

- if...else
- if...else if
- switch...case

6.3. Operadores

As estruturas de decisão trabalham com os Operadores Relacionais e os Operadores lógicos. Em C# estes operadores são representados conforme a seguir:

6.3.1. Operadores relacionais:

Descrição	Operador
Igual	==
Diferente	!=
Menor que	<
Maior que	>
Menor ou igual	<=
Maior ou igual	>=

6.3.2. Operadores lógicos:

Operação	Operador
E	&&
OU	
Não	!

6.4. Estrutura if...Else

6.4.1. Finalidade

Executar o bloco de comandos que vem após a “condição”, quando essa “condição” for verdadeira. Caso a “condição” não seja verdadeira (seja falsa), serão executados os comandos após a palavra else.

6.4.2. Sintaxe

```
if (condição)
{
    <comando>;
    <comando>;
    :
    :
}
else
{
    <comando>;
    <comando>;
    :
    :
}
```

Obs: Caso seja colocado somente um comando após a (condição) ou após o else, as chaves abrindo e fechando os comandos podem ser dispensadas.

Obs: É possível utilizar o comando **If** sem a clausula **else**, neste caso os comandos somente serão executados se a condição for verdadeira, caso contrário nenhum comando será executado.

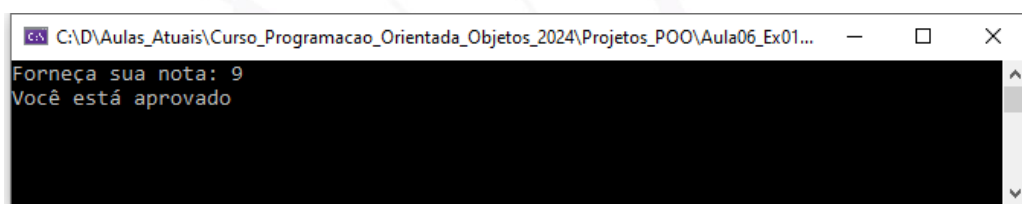
6.4.3. Exemplo:

```
1. namespace Aula06_Ex01
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis
8.             double nota;
9.
10.            // Recebe a nota
11.            Console.Write("Forneça sua nota: ");
12.            nota = double.Parse(Console.ReadLine());
13.
14.            // Verifica se o usuário está aprovado ou em exame
15.            if (nota >= 7)
16.                Console.WriteLine("Você está aprovado");
17.            else
18.                Console.WriteLine("Você está em exame");
19.
20.            //Mantem a tela aberta esperando a digitação de uma tecla
21.            Console.ReadKey();
22.
23.        }
24.    }
25. }
```

Comentários:

Observe que se a nota for **maior ou igual a 7 a condição (nota>=7) resultará em verdadeira, sendo executada a instrução que vem imediatamente após a “condição”**. Caso contrário, será executado o comando que está após a cláusula else.

6.4.4. Tela



6.5. Comando if com clausula else if

Pode acontecer de existirem mais desmembramentos nas condições testadas por uma estrutura if. No exemplo dado, com média inferior a 7, mas superior a 3 o aluno tem direito a recuperação, e somente com nota inferior a 3 é que ele está reprovado.

Então, nova condição deve ser testada. Para tanto, utiliza-se o comando else if.

6.5.1. Sintaxe

```
if (condição)
{
    <comando>;
    <comando>;
    :
    :
}
else if (condição)
{
    <comando>;
    <comando>;
    :
    :
}
else
{
    <comando>;
    <comando>;
    :
    :
}
```

6.5.2. Exemplo

```
1. namespace Aula06_Ex02
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis
8.             double nota;
9.
10.            // Recebe a nota
11.            Console.Write("Forneça sua nota: ");
12.            nota = double.Parse(Console.ReadLine());
13.
14.            if (nota >= 7)
15.                // Verifica se o usuário está aprovado
16.                Console.WriteLine("Você está aprovado");
17.
18.            else if (nota >= 3)
19.                // Verifica se o usuário está em recuperação
20.                // Observar que a nota com certeza não é maior ou igual a 7
```

```

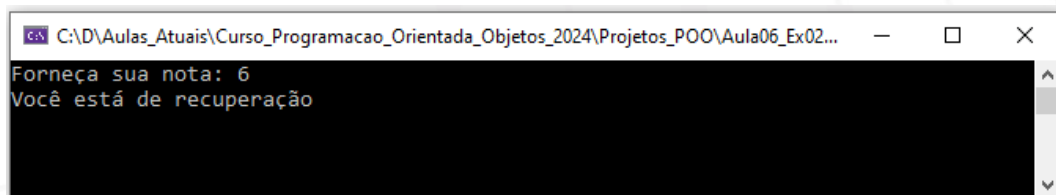
21.         Console.WriteLine("Você está de recuperação");
22.
23.     else
24.         // Informa que o usuário está reprovado
25.         // Observar que a nota com certeza não é maior ou igual a 7
26.         // Nem maior ou igual a 5
27.         Console.WriteLine("Você reprovado");
28.
29.         //Mantem a tela aberta esperando a digitação de uma tecla
30.         Console.ReadKey();
31.     }
32. }
33. }

```

Comentários:

- 1) Observe que **se a nota for igual ou superior a 7 será executado o comando após a primeira condição (nota >=7)**. Neste caso a instrução if será interrompida, não havendo hipótese de serem executadas as demais cláusulas desta instrução.
- 2) Observe que **para executar o else if a nota precisa ser maior ou igual a 3, mas com certeza ela é menor que 7 pois se fosse maior ou igual a 7 o primeiro if seria executado** e o comando if encerrado antes de chegar ao else if.
- 3) Observe que se a **nota não for maior ou igual a 7**, nem maior ou igual a 3 nem o if nem o else if serão executados, **então, o comando executará a cláusula else**.

6.5.3. Tela



6.6. Comando if com operadores lógicos

Pode haver necessidade de um teste ser feito dentro de um intervalo ou com intervalos alternativos. Neste caso torna-se necessário a utilização de operadores lógicos na cláusula if, de maneira que mais de uma condição seja testada.

6.6.1. Sintaxes

```
if ((condição1)&& (condição2)) <=
{
    <comando>
    <comando>
    :
    :
}
```

Neste caso os comandos após o if só serão executados **se as duas condições forem verdadeiras**

```
if (condição3) || (condição4) <=
{
    <comando>
    <comando>
    :
    :
}
```

Neste caso os comandos após o if serão executados **se qualquer uma das duas condições for verdadeira**

6.6.2. Exemplo:

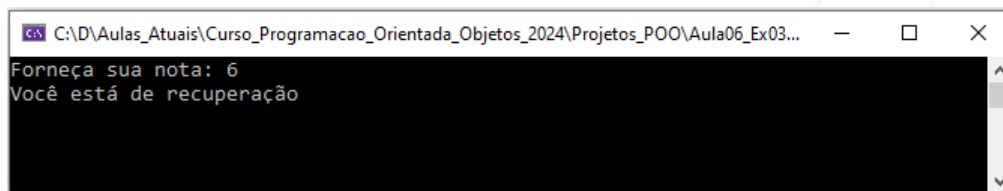
```
1. namespace Aula06_Ex03
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis
8.             double nota;
9.
10.            // Recebe a nota
11.            Console.WriteLine("Forneça sua nota: ");
12.            nota = double.Parse(Console.ReadLine());
13.
14.            if ((nota < 0) || (nota > 10))
15.                // Executado se a nota for menor que zero ou maior que 10
16.                Console.WriteLine("Forneça um valor entre 0 e 10");
17.
18.            else if (nota >= 7)
19.                // Verifica se o usuário está aprovado
20.                Console.WriteLine("Você está aprovado");
21.
22.            else if (nota >= 3)
23.                // Verifica se o usuário está em recuperação
24.                Console.WriteLine("Você está de recuperação");
25.
26.            else
27.                // Informa que o usuário está reprovado.
28.                // Observar que neste caso, a nota.
29.                // não é menor que zero nem maior que 10.
30.                // não está entre 7 e 10 e
31.                // não está entre 3 e 7)
32.                Console.WriteLine("Você está reprovado");
33.
34.            //Mantem a tela aberta esperando a digitação de uma tecla
35.            Console.ReadKey();
36.
37.        }
38.    }
39. }
```

Comentários:

No exemplo dado é importante observar que:

- 1) No primeiro caso, **testado com o operador ou (||)** a nota não pode ser menor que zero ou maior que 10.
- 2) O primeiro else if, só será executado se a nota estiver entre 7 e 10 (visto que obrigatoriamente ela é maior ou igual a 0 e menor ou igual a 10).
- 3) No segundo else if, só será executado, se a nota estiver entre 3 e 7 (visto que se fosse maior que 7 o else if anterior teria sido executado).
- 4) O else só será executado se nenhuma das condições anteriores forem verdadeiras portando a nota será menor que 3.

6.6.3. Tela



6.7. Comandos ifs aninhados

Aninhar comandos consiste em colocar comandos dentro de comandos, no caso de “comandos ifs”, consiste em colocar “estruturas ifs” dentro de “estruturas ifs”.

6.7.1. Sintaxe

```
if condição1)
{
    If (condição 2)
    {
        <comando>
        <comando>
        :
        :
    }
}
```

Obs: No exemplo o segundo if (aquele com a condição 2) está aninhado com o primeiro (aquele com a condição 1).

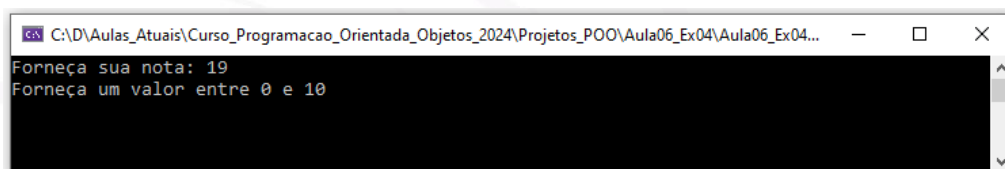
6.7.2. Exemplo

```

1. namespace Aula06_Ex04
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis
8.             double nota;
9.
10.            // Recebe a nota
11.            Console.WriteLine("Forneça sua nota: ");
12.            nota = double.Parse(Console.ReadLine());
13.
14.            if ((nota >= 0) && (nota <= 10))
15.            {
16.                // Executado se a nota for maior ou igual a zero
17.                // e menor ou igual a 10
18.                if (nota >= 7)
19.                {
20.                    // Verifica se o usuário está aprovado
21.                    Console.WriteLine("Você está aprovado");
22.
23.                }
24.                else if (nota >= 3)
25.                {
26.                    // Verifica se o usuário está em recuperação
27.                    Console.WriteLine("Você está de recuperação");
28.
29.                }
30.                else
31.                {
32.                    // Informa que o usuário está reprovado.
33.                    // Observar que neste caso, a nota.
34.                    // é menor que 7 e menor que 3.
35.                    Console.WriteLine("Você está reprovado");
36.                }
37.            }
38.            // Executado se a nota não estiver entre 0 e 10
39.            Console.WriteLine("Forneça um valor entre 0 e 10");
40.
41.            //Mantem a tela aberta esperando a digitação de uma tecla
42.            Console.ReadKey();
43.        }
44.    }
45. }

```

6.7.3. Tela



6.8. Testes de digitação de valores numéricos

Quando um usuário deve digitar um valor numérico este valor precisa ser recebido com o comando `Console.WriteLine()`. Como este comando só aceita valores string, a digitação do usuário deverá ser recebida, convertida para um valor numérico tal como byte,

int, double e os demais tipos numéricos. Porém, se no momento da digitação o usuário colocar algum caractere especial ou letra, o programa apresentará um erro de runtime se for convertido diretamente com instruções do tipo double.Parse.

Uma das maneiras de evitar este tipo de erro é receber os valores com instruções do tipo:

```
digi = double.TryParse(Console.ReadLine(), out valor);
```

Onde:

digi é uma variável do tipo bool que deverá receber um dos valores

Esta instrução atribuirá:

À variável **digi**, que deve ser do tipo bool, o conteúdo **true** se o usuário digitar um número e o conteúdo **false** se o usuário digitar caracteres.

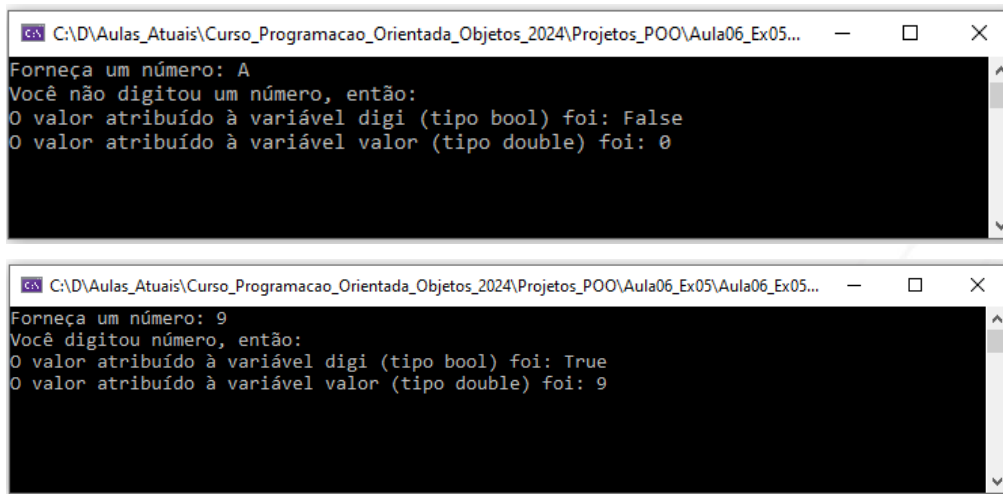
À variável **valor**, que deve ser de um tipo numérico, o **valor que o usuário digitar** se o usuário digitar um número e o **valor 0 (zero)** se o usuário digitar caracteres.

6.8.1. Exemplo

```
1. namespace Aula06_Ex05
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara Variáveis
8.             double valor;
9.             bool digi;
10.
11.             // Recebe valores digitados no teclado
12.             // Se a digitação for numérica, atribui true para a variável digi
13.             // Se a digitação não for numérica, atribui false para a variável digi
14.             Console.WriteLine("Forneça um número: ");
15.             digi = double.TryParse(Console.ReadLine(), out valor);
16.
17.             // Testa se o valor digitado é numérico ou possui caracteres
18.             if (digi == false)
19.             {
20.                 Console.WriteLine("Você não digitou um número, então:");
21.             }
22.             else
23.             {
24.                 Console.WriteLine("Você digitou número, então:");
25.             }
26.             Console.WriteLine("O valor atribuído à variável digi (tipo bool) foi:
27. " + digi);
28.             Console.WriteLine("O valor atribuído à variável valor (tipo double)
29. foi: " + valor);
```

```
29.         //Mantem a tela aberta esperando a digitação de uma tecla
30.         Console.ReadKey();
31.     }
32. }
33. }
```

6.8.2. Telas:



6.9. Comando switch...case

Assim como a estrutura “if...else if... else” (if com a clausula else if) do item anterior, o comando “**switch...case**” é também um comando de decisão. Normalmente deve-se dar preferência ao comando “switch.. .case”, quando existem mais que duas (várias) decisões a serem tomadas.

Se por exemplo deseja-se criar um programa para fornecer uma mensagem para cada dia da semana, como existem 7 opções possíveis, é melhor utilizar o comando “switch...case”, embora o programa também pudesse ser implementado com os comandos “if...else if...else”.

Existem vantagem e desvantagens da utilização do “switch...case” sobre “if...else if...else”

6.9.1. Vantagens:

A variável a ser testada é lida uma só vez;

Uma mesma condição ou condições pode ou podem ser utilizadas para diversas clausulas (exemplo será mostrado a seguir);

6.9.2. Desvantagens:

Com o comando “switch...case” só é possível testar igualdade, ou seja, diferentemente da sintaxe do if e else...if não é possível verificar se o valor a ser comparado é maior, menor, diferente etc.

O comando “switch...case” só opera com variáveis Boleana (**bool**), Inteiras (**byte, sbyte, short, etc**) e caracteres (**char e string**). Não aceita variáveis reais tais como float e double etc.

6.9.3. Finalidade

Executar o bloco de comandos que vem após o “case”, caso o valor fornecido após a palavra case (valorTestado), seja igual ao valor com o conteúdo da variável indicada após a palavra switch (valorEsperado).

Após cada uma das condições, deve-se colocar a instrução break, para que o programa não verifique os demais “cases”.

6.9.4. Sintaxe

```
switch (valorEsperado)
{
    case valorTestado1:
        <comando>;
        break;
    case valorTestado2:
        <comando>;
        break;
    case valorTestado3:
        <comando>;
        break;
    :
    :
    :
}
```

6.9.5. Detalhes

Após o “valorEsperado”, o conjunto de cases deve ser aberto com uma chave {.

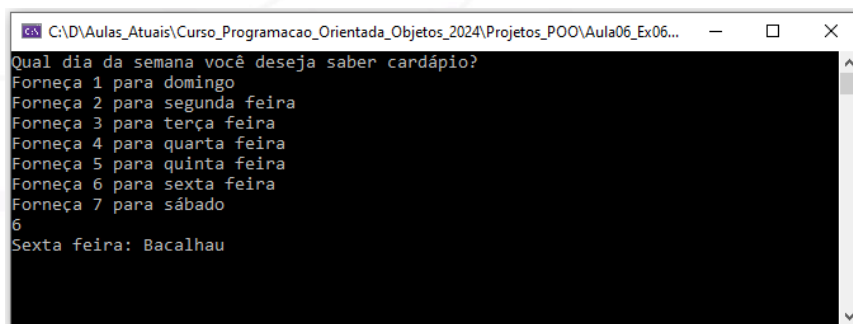
Ao encerrar o conjunto de cases o comando deverá ser fechado com uma chave }.

O comando break interrompe a execução do case, fazendo com que o programa passe para os comandos após a chave que encerra o case.

6.9.6. Exemplo

```
1. namespace Aula06_Ex06
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara variáveis
8.             char dia;
9.
10.            // Recebe o dia da semana
11.            Console.WriteLine("Qual dia da semana você deseja saber cardápio? ");
12.            Console.WriteLine("Forneça 1 para domingo");
13.            Console.WriteLine("Forneça 2 para segunda feira");
14.            Console.WriteLine("Forneça 3 para terça feira");
15.            Console.WriteLine("Forneça 4 para quarta feira");
16.            Console.WriteLine("Forneça 5 para quinta feira");
17.            Console.WriteLine("Forneça 6 para sexta feira");
18.            Console.WriteLine("Forneça 7 para sábado");
19.            dia = char.Parse(Console.ReadLine());
20.            // Fornece o prato do dia
21.            switch (dia)
22.            {
23.                case ('1'):
24.                    Console.WriteLine("Domingo: Macarrão a Bolonhesa");
25.                    break;
26.                case ('2'):
27.                    Console.WriteLine("Segunda feira: Virado a Paulista");
28.                    break;
29.                case ('3'):
30.                    Console.WriteLine("Terça feira: Dobradinha");
31.                    break;
32.                case ('4'):
33.                    Console.WriteLine("Quarta feira: Feijoada");
34.                    break;
35.                case ('5'):
36.                    Console.WriteLine("Quinta feira: Rabada");
37.                    break;
38.                case ('6'):
39.                    Console.WriteLine("Sexta feira: Bacalhau");
40.                    break;
41.                case ('7'):
42.                    Console.WriteLine("Sábado: Feijoada especial");
43.                    break;
44.            }
45.            // Mantem a tela aberta esperando a digitação de uma tecla
46.            Console.ReadKey();
47.        }
48.    }
49. }
```

6.9.7. Tela



```

C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula06_Ex06...
Qual dia da semana você deseja saber cardápio?
Forneça 1 para domingo
Forneça 2 para segunda feira
Forneça 3 para terça feira
Forneça 4 para quarta feira
Forneça 5 para quinta feira
Forneça 6 para sexta feira
Forneça 7 para sábado
6
Sexta feira: Bacalhau
  
```

6.10. Comando switch...case com clausula default

Caso nenhum dos “cases” forem executados, devem ser executados os comandos existentes após a cláusula default.

A cláusula default não é obrigatória, mas sempre que for utilizada deverá ser colocada após todos os cases.

6.10.1. Sintaxe

```

switch (valorEsperado)
{
    case valorTestado1:
        <comando>;
        break;
    case valorTestado2:
        <comando>;
        break;
    case valorTestado3:
        <comando>;
        break;
    default:
        <comando>;
        Break;
}
  
```

6.10.2. Exemplo:

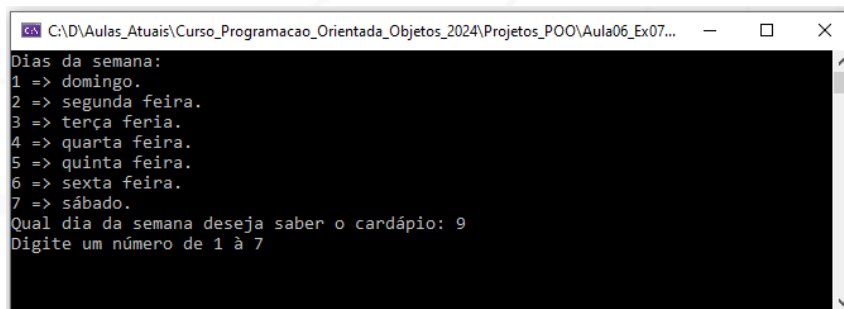
```

1. namespace Aula06_Ex07
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             char dia;
8.             int dig;
9.
10.            Console.WriteLine("Dias da semana:");
11.            Console.WriteLine("1 => domingo.");
  
```

```
12. Console.WriteLine("2 => segunda feira.");
13. Console.WriteLine("3 => terça feria.");
14. Console.WriteLine("4 => quarta feira.");
15. Console.WriteLine("5 => quinta feira.");
16. Console.WriteLine("6 => sexta feira.");
17. Console.WriteLine("7 => sábado.");
18. Console.Write("Qual dia da semana deseja saber o cardápio: ");
19. dig = Console.Read();
20. dia = Convert.ToChar(dig);
21.
22. switch (dia)
23. {
24.     case ('1'):
25.         Console.WriteLine("Domingo: Macarrão a bolonhesa.");
26.         break;
27.     case ('2'):
28.         Console.WriteLine("Segunda: Virado a paulista.");
29.         break;
30.     case ('3'):
31.         Console.WriteLine("Terça: Dobradinha.");
32.         break;
33.     case ('4'):
34.         Console.WriteLine("Quarta: Feijoadada.");
35.         break;
36.     case ('5'):
37.         Console.WriteLine("Quinta: Rabada.");
38.         break;
39.     case ('6'):
40.         Console.WriteLine("Sexta: Bacalhau.");
41.         break;
42.     case ('7'):
43.         Console.WriteLine("Sábado: Feijoadada especial.");
44.         break;
45.     default:
46.         Console.WriteLine("Digite um número de 1 à 7");
47.         break;
48. }
49. // Mantem a tela aberta esperando a digitação de uma tecla
50. Console.ReadKey();
51.
52. }
53. }
54. }
```

Nota: Neste exemplo foi utilizada na tomada de dados os comandos apresentados na aula anterior, de maneira que se o usuário digitar mais de um caractere, somente o primeiro é assumido evitando que o programa apresente um erro de runtime.

6.10.3. Tela



```

C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula06_Ex07...
Dias da semana:
1 => domingo.
2 => segunda feira.
3 => terça feira.
4 => quarta feira.
5 => quinta feira.
6 => sexta feira.
7 => sábado.
Qual dia da semana deseja saber o cardápio: 9
Digite um número de 1 à 7
  
```

6.11. Diversos cases devem executar os mesmos comandos

Uma cláusula case **não pode testar mais que um valor**, entretanto, **caso se deseje que os mesmos comandos sejam executados para diversas clausulas cases**, basta deixar os cases em sequência e sem comandos (inclusive sem break), atribuindo os comandos desejados somente ao último case da série. Desta maneira, será executado somente o case que tem comandos (que deve ser o último da sequência) e consequentemente um break ao final.

Para entender este funcionamento, vamos supor que se deseja fazer um programa que fornece três mensagens para os 7 dias da semana, conforme a seguir:

- Para **segunda e terça** a mensagem é “**Força a semana está começando**”.
- Para **quarta, quinta e sexta** a mensagem é “**Aguente, o fim de semana já vai chegar**”.
- Para **sábado e domingo** a mensagem é “**Que bom, estamos no fim de semana**”.

Nota: Neste exemplo está sendo utilizado também o comando ToLower() que converte letras maiúsculas para letra minúsculas.

6.11.1. Exemplo

```

1. namespace Aula06_Ex08
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declaração de variáveis
8.             char dia;
9.             // Recebe o dia da semana
10.            Console.WriteLine("Que dia da semana é hoje? ");
11.            Console.WriteLine("Forneça s para segunda feira");
12.            Console.WriteLine("Forneça t para terça feira");
  
```

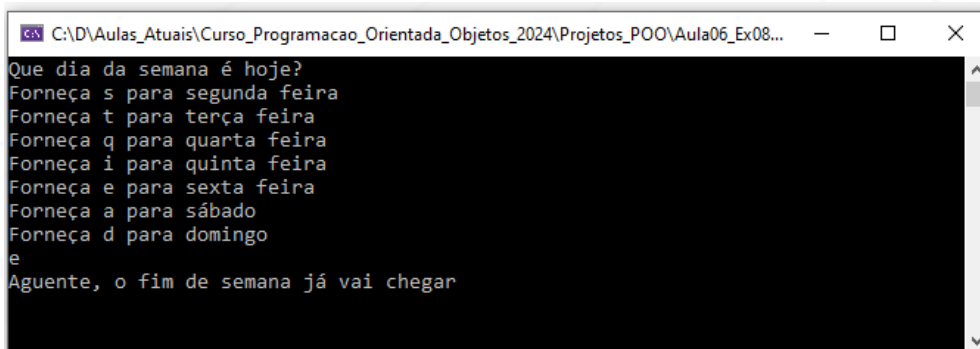


```

13. Console.WriteLine("Forneça q para quarta feira");
14. Console.WriteLine("Forneça i para quinta feira");
15. Console.WriteLine("Forneça e para sexta feira");
16. Console.WriteLine("Forneça a para sábado");
17. Console.WriteLine("Forneça d para domingo");
18. dia = char.Parse(Console.ReadLine().ToLower());
19.
20. // Fornece a mensagem
21. switch (dia)
22. {
23.     case 's':
24.     case 't':
25.         Console.WriteLine("Força a semana está começando");
26.         break;
27.     case 'q':
28.     case 'i':
29.     case 'e':
30.         Console.WriteLine("Aguente, o fim de semana já vai chegar");
31.         break;
32.     case 'a':
33.     case 'd':
34.         Console.WriteLine("Que bom, estamos no fim de semana");
35.         break;
36.     default:
37.         Console.WriteLine("Por favor digite s,t,q,i,e,a ou d");
38.         break;
39. }
40. // Mantem a tela aberta esperando a digitação de uma tecla
41. Console.ReadKey();
42.
43. }
44. }
45. }

```

6.11.2. Tela



```

C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula06_Ex08...
Que dia da semana é hoje?
Forneça s para segunda feira
Forneça t para terça feira
Forneça q para quarta feira
Forneça i para quinta feira
Forneça e para sexta feira
Forneça a para sábado
Forneça d para domingo
e
Aguente, o fim de semana já vai chegar

```

7. ESTRUTURAS DE REPETIÇÃO

7.1. Objetivo

Apresentar as estruturas de repetição

7.2. Introdução

Muitas vezes em um programa é necessário repetir uma série de comandos um determinado número de vezes, ou quantas vezes forem necessárias, enquanto ou até que uma determinada condição seja verdadeira.

Para executar estas rotinas o C# possui 3 estruturas:

- for valor inicial; valor final; incremento;
- do while;
- while.

Estas estruturas, também chamadas de “estruturas de loop” ou “estruturas de laço”, são muitas vezes classificados como:

- Estrutura de laço contado (que é a estrutura do tipo for).
- Estrutura de laço condicional (que são as estruturas do tipo “do while” e “while”).

Fundamentalmente a diferença entre estas estruturas é que na estrutura do tipo **for**, existe um contador interno e **o programador determina quantas vezes o loop será executado**, enquanto que nas estruturas do tipo **while** e **do while** os comandos são executados **enquanto ou até que determinada condição seja verdadeira**.

7.3. Estruturas for

7.3.1. Finalidade

Executar o bloco de comandos entre os limitadores {...}, **enquanto um contador é acrescido de um valor inicial até um valor final** obedecendo a um incremento pré-determinado.

7.3.2. Sintaxe

```
for (valor_inicial; valor_final; incremento)
{
    <comando>;
    <comando>;
    <comando>;
    :
    :
    :
}
```

7.3.3. Detalhes

O valor_inicial na verdade **declara e atribui** um valor a uma variável que deve ser um número inteiro.

O valor_final que também deve ser um número inteiro é o valor que será testado pelo comando para ver se foi atingido. Caso o valor seja atingido, o grupo de comandos entre os marcadores { } não será mais executado, e o programa seguirá após o primeiro comando após o marcador de encerramento da estrutura.

Incremento é uma instrução que informa como o valor inicial deve ser incrementado a cada teste. São possíveis incrementos unitários ou múltiplos, positivos, negativos. É possível até que o incremento seja uma equação matemática, lembrando que ele tem que ser um número inteiro.

7.3.4. Exemplo:

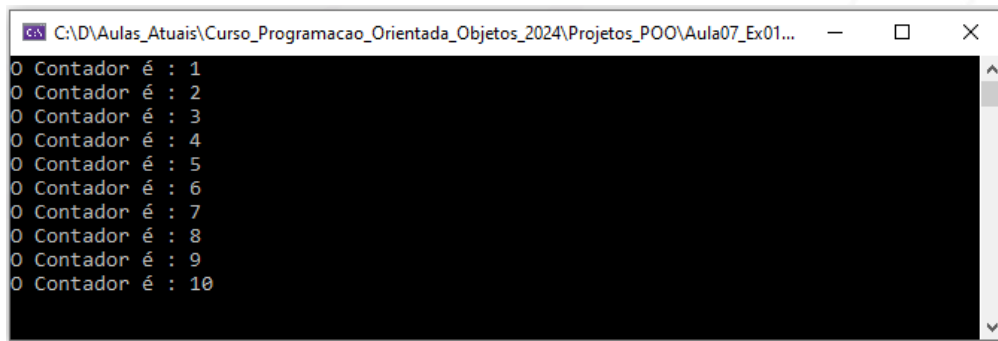
```
1. namespace Aula07_Ex01
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Delclara variáveis
8.             int contador;
9.
10.            // Executa um contador de 1 até 10
11.            for (contador = 1; contador <= 10; contador++)
12.            {
13.                Console.WriteLine("O Contador é : " + contador);
14.            }
15.            // Mantem a tela aberta esperando a digitação de uma tecla
16.            Console.ReadKey();
17.        }
18.    }
19. }
```

Obs: Olhando atentamente a sintaxe do comando “for”, é possível verificar duas informações:

No início quando é colocado **contador = 1** (existe somente um sinal de igual), portanto isto na verdade é uma atribuição, ou seja: ao contador é atribuído o valor inicial 1

Quando na continuação da sintaxe se coloca **contador =< 10** o que está sendo feito é uma comparação (existem dois sinais =<), ou seja, está se comparando se o valor atual do contador é menor ou igual a 10.

7.3.5. Tela



```
C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula07_Ex01...
O Contador é : 1
O Contador é : 2
O Contador é : 3
O Contador é : 4
O Contador é : 5
O Contador é : 6
O Contador é : 7
O Contador é : 8
O Contador é : 9
O Contador é : 10
```

7.4. Estrutura do...while

7.4.1. Finalidade

Executar o bloco de comandos entre as os limitadores {...}, **enquanto uma determinada condição, testada pelo while for verdadeira.**

É importante notar que o teste é executado no final do comando, então o grupo de comandos **será executado pelo menos uma vez, independente da condição ser ou não satisfeita.**

7.4.2. Sintaxe

```
Do
{
    <comando>;
    <comando>;
    <comando>;
    :
    :
```

```
        :  
    }  
    while (condicao == teste)
```

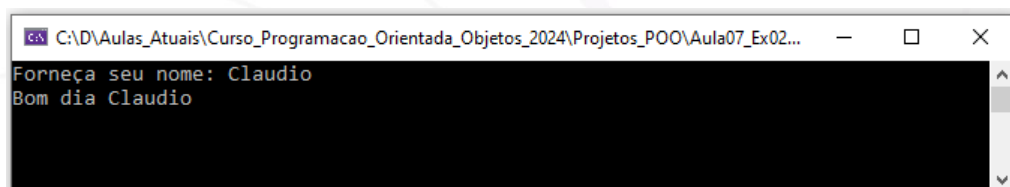
7.4.3. Exemplo

```
1. namespace Aula07_Ex02  
2. {  
3.     internal class Program  
4.     {  
5.         static void Main(string[] args)  
6.         {  
7.             // Declara variável  
8.             string nome;  
9.  
10.            // Solicita o nome do usuário  
11.            do  
12.            {  
13.                Console.Write("Forneça seu nome: ");  
14.                nome = Console.ReadLine();  
15.            }  
16.            while (nome == "");  
17.  
18.            // Cumprimenta o usuário  
19.            Console.WriteLine("Bom dia " + nome);  
20.  
21.            // Mantem a tela aberta esperando a digitação de uma tecla  
22.            Console.ReadKey();  
23.        }  
24.    }  
25. }
```

Obs: Conforme foi falado anteriormente este comando **é executando pelo menos uma vez**, independentemente do valor da variável nome.

Este é um programa clássico de verificação de digitação do usuário, pode ser implementado para conferir praticamente todos os tipos de digitação, só dando continuidade ao programa se a digitação do usuário for coerente com o que se deseja.

7.4.4. Tela



7.5. Alguns comandos para manipulações de string

Existem diversos comandos que auxiliam, quando se pretende executar testes de digitação. A seguir, serão fornecidos alguns destes.

7.5.1. *Calcula o tamanho de uma string*

Permite determinar número de caracteres de uma string.

7.5.1.1. *Sintaxe*

```
variavel = identificador_string.length
```

7.5.1.2. *Exemplo:*

```
tamanho = nome.Length;
```

7.5.2. *Identifica caracteres de uma string*

Identifica cada um dos caracteres de uma string

7.5.2.1. *Sintaxe*

```
variavel = new string(new char[n])
```

Onde:

- a) n é o número de cada um dos caracteres da string.
- b) Os caracteres ficam no formato char.
- c) À primeira letra é atribuído ao char[0], a segunda ao char[1] e assim sucessivamente, até a última letra que é atribuída ao char[n-1] (sendo n o total de caracteres da string original).

7.5.2.2. *Exemplo:*

```
nome = new string(new char[30]);
```

7.5.3. *Captura conjuntos de caracteres em uma string*

Este comando cria uma string a partir de outra, pegando somente uma parte dos caracteres da string.

7.5.3.1. Sintaxe

variavel = variavel.Substring(inicio,fim)

Onde:

- a) **inicio** é o número do caracter da string a partir do qual será iniciada a captura.
- b) **fim** é o número do caracter da string onde será encerrada a considerada.

7.5.3.2. Exemplo

nome = nome.Substring(0, 9);

Pega os 10 primeiros caracteres a string nome.

7.5.4. Exemplo:

```

1. namespace Aula07_Ex03
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declarações de Variáveis
8.             string nome;
9.             int tamanho;
10.            char plettra, ulettra;
11.
12.            // Cria a variável nome indexada de 0 à 29
13.            nome = new string(new char[30]);
14.
15.            // Impede que seja informado um nome em branco
16.            do
17.            {
18.                Console.Write("Forneça seu nome: ");
19.                nome = Console.ReadLine();
20.                if (nome == "")
21.                    Console.WriteLine("Por favor não deixe em branco");
22.            }
23.            while (nome == "");
24.
25.            // Mostra nome
26.            Console.WriteLine("Olá Sr(a) " + nome + " Bem vindo");
27.
28.            // Calcula e mostra a quantidade de letras
29.            tamanho = nome.Length;
30.            Console.WriteLine("Seu nome possui " + tamanho + " letras");
31.
32.            // Verifica e mostra a primeira e última letra
33.            plettra = nome[0];
34.            ulettra = nome[tamanho - 1];
35.            Console.WriteLine("A primeira letra de seu nome é " + plettra);
36.            Console.WriteLine("A última letra de seu nome é " + ulettra);
37.
38.            // Mantem a tela aberta esperando a digitação de uma tecla
39.            Console.ReadKey();

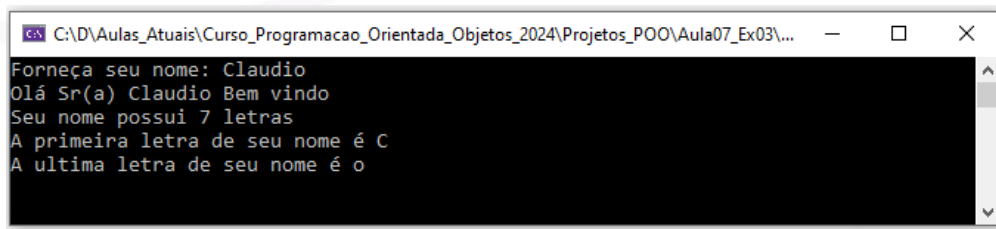
```

```

40.     }
41. }
42. }

```

7.5.5. Tela



7.5.6. Exemplo:

Este exemplo separa o nome e o sobrenome de um usuário

```

1. namespace Aula07_Ex04
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declaração de variáveis
8.             string nome;
9.             char letra;
10.            int c, tamanho;
11.
12.            // Cria a variável nome indexada de 0 à 29
13.            nome = new string(new char[30]);
14.
15.            // Solicita o nome completo do usuário
16.            do
17.            {
18.                Console.Write("Forneça seu nome e sobrenome ");
19.                nome = Console.ReadLine();
20.            }
21.            while (nome == "");
22.
23.            // Verifica e informa quantas letras tem o nome completo do usuário
24.            tamanho = nome.Length;
25.
26.            // Separa o primeiro nome
27.            Console.Write("Seu primeiro nome é: \t");
28.            c = 0;
29.            do
30.            {
31.                letra = nome[c];
32.                Console.Write(letra);
33.                c = c + 1;
34.            }
35.            while ((letra != ' ') && (c < tamanho));
36.            if (c < tamanho)
37.            {
38.                // Separa o segundo nome
39.                Console.Write("\nSeu sobrenome nome é: \t");
40.                do
41.                {

```

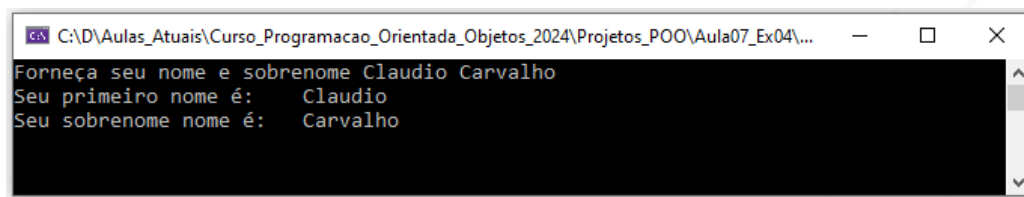


```

42.         letra = nome[c];
43.         Console.WriteLine(letra);
44.         c = c + 1;
45.     }
46.     while ((letra != ' ') && (c < tamanho));
47. }
48. // Mantem a tela aberta esperando a digitação de uma tecla
49. Console.ReadKey();
50. }
51. }
52. }

```

7.5.7. Tela



7.5.8. Exemplo:

O programa anterior só admite nome e um sobrenome, se desejarmos que o usuário forneça o nome completo e ele separe cada um dos nomes devemos utilizar loops encadeados conforme o exemplo a seguir:

```

1. namespace Aula07_Ex05
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declaração de variáveis
8.             string nome;
9.             int tamanho, c;
10.            char letra;
11.
12.            // Cria a variável nome indexada de 0 à 49
13.            nome = new string(new char[50]);
14.
15.            // Solicita o nome completo do usuário
16.            do
17.            {
18.                Console.WriteLine("Qual seu nome completo? (máx. 50 caracteres) ");
19.                nome = Console.ReadLine();
20.                if (nome == "")
21.                    Console.WriteLine("Por favor não deixe em branco");
22.            }
23.            while (nome == "");
24.
25.            // Verifica e informa quantas letras tem o nome fornecido
26.            tamanho = nome.Length;
27.
28.            // Separa o nome e sobrenomes fornecidos
29.            Console.WriteLine("Seu nome e sobrenomes são:");
30.            c = 0;

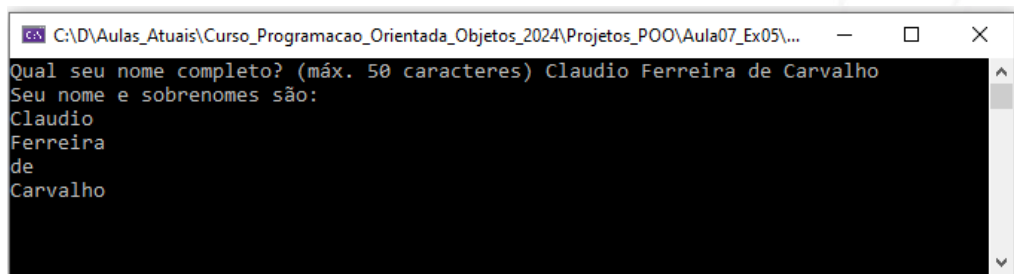
```

```

31.         do
32.         {
33.             do
34.             {
35.                 letra = nome[c];
36.                 Console.Write(letra);
37.                 c = c + 1;
38.             }
39.             while ((letra != ' ') && (c < tamanho));
40.             Console.WriteLine();
41.         }
42.         while (c < tamanho);
43.
44.         // Mantem a tela aberta esperando a digitação de uma tecla
45.         Console.ReadKey();
46.     }
47. }
48. }

```

7.5.9. Tela



7.6. Estrutura while

7.6.1. Finalidade

Executar o bloco de comandos entre as os limitadores {...}, **enquanto uma determinada condição, testada pelo while for verdadeira.**

Note que o Comando while, testa a condição **antes de entrar no loop**. Caso a condição não seja satisfeita, o loop não será executado nenhuma vez. Por este motivo é importante garantir que **antes do comando while, seja atribuído um valor para a variável e este valor deve atender as condições de teste.**

7.6.2. Sintaxe

```

while (condição == teste)
{
    <comando>;
    <comando>;
    <comando>;
    :
}

```

```

:
:
}

```

7.6.3. Exemplo

```

1. namespace Aula07_Ex06
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Delclara variáveis
8.             double saldo;
9.             double saque;
10.
11.             // Solicita o saldo atual
12.             Console.WriteLine("Forneça o saldo atual: ");
13.             saldo = double.Parse(Console.ReadLine());
14.
15.             // Permite retiradas enquanto existir saldo
16.             while (saldo > 0)
17.             {
18.                 Console.WriteLine("Forneça o valor do saque: ");
19.                 saque = double.Parse(Console.ReadLine());
20.
21.                 // Informa o Saldo ou avisa que não pode sacar
22.                 if (saque <= saldo)
23.                 {
24.                     saldo = saldo - saque;
25.                     Console.WriteLine("Seu saldo é: " + saldo);
26.                 }
27.                 else
28.                 {
29.                     Console.WriteLine("Você não pode sacar este valor ");
30.                 }
31.             }
32.
33.             // Informa que o saldo encerrou (fechou o comando While)
34.             Console.WriteLine("Você já fez todas as retiradas possíveis");
35.
36.             // Mantem a tela aberta esperando a digitação de uma tecla
37.             Console.ReadKey();
38.         }
39.     }
40. }

```

Obs: Neste programa, o “valor do saldo” está sendo atribuído antes de se entrar no comando while.

Este programa não permite que o usuário interrompa os saques enquanto não encerrar o saldo, esta deficiência será sanada no próximo exemplo.

7.6.4. Tela

```

C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula07_Ex06\...
Forneça o saldo atual: 600
Forneça o valor do saque: 200
Seu saldo é: 400
Forneça o valor do saque: 100
Seu saldo é: 300
Forneça o valor do saque: 50
Seu saldo é: 250
Forneça o valor do saque: 250
Seu saldo é: 0
Você já fez todas as retiradas possíveis

```

7.6.5. Exemplo

Utilizando “do...while” e “while” (as duas estruturas estudadas nesta aula, no mesmo programa).

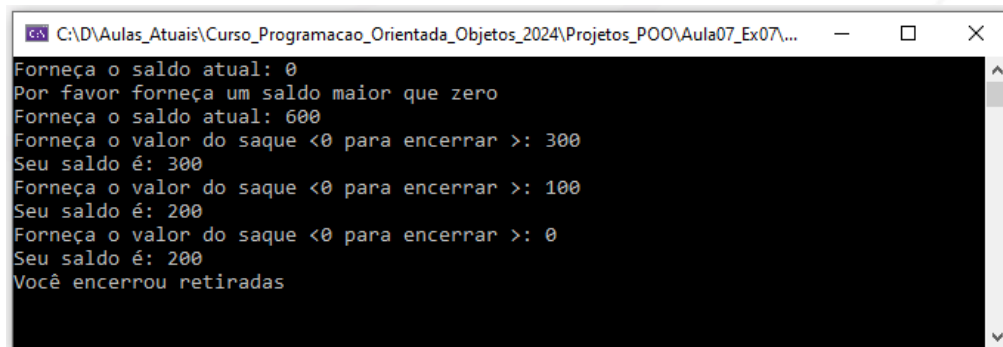
```

1. namespace Aula07_Ex07
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara variáveis
8.             double saldo;
9.             double saque;
10.
11.            // Solicita o saldo atual e não permite saldo negativo ou nulo
12.            do
13.            {
14.                Console.WriteLine("Forneça o saldo atual: ");
15.                saldo = double.Parse(Console.ReadLine());
16.
17.                // fornece mensagem indicando que o saldo deve ser positivo
18.                if (saldo <= 0)
19.                {
20.                    Console.WriteLine("Por favor forneça um saldo maior que
zero");
21.                }
22.            }
23.            while (saldo <= 0);
24.
25.            // Testa se existe saldo, caso exista permite mais um saque.
26.            while (saldo > 0)
27.            {
28.                Console.WriteLine("Forneça o valor do saque <0 para encerrar >: ");
29.                saque = double.Parse(Console.ReadLine());
30.                // Não permite saque sem saldo.
31.                if (saque <= saldo)
32.                {
33.                    saldo = saldo - saque;
34.                    Console.WriteLine("Seu saldo é: " + saldo);
35.
36.                    // Encerra o programa
37.                    if (saque == 0)
38.                    {
39.                        break;
40.                    }
41.                }
42.            }
43.        }
44.    }
45. }

```

```
43.         {  
44.             Console.WriteLine("Você não pode sacar este valor");  
45.         }  
46.     }  
47.  
48.     Console.WriteLine("Você encerrou retiradas");  
49.  
50.     // Mantem a tela aberta esperando a digitação de uma tecla  
51.     Console.ReadKey();  
52. }  
53. }  
54. }
```

7.6.6. Tela



8. MÉTODOS – MÉTODOS *Main()* VARIÁVEIS GLOBAIS

8.1. Objetivo

Apresentar conceitos de Procedimentos e Funções evoluindo para Objetos Atributos e Métodos que são elementos da Programação Orientada a Objetos.

8.2. Introdução

A linguagem C# foi concebida sob o paradigma de “Programação Orientada a Objetos” (POO). Esta filosofia de programação pode ser considerada como uma evolução do conceito de “Programação Estruturada”.

Por sua vez, os conceitos de programação estruturada que começaram a ser desenvolvidos no início dos anos 60 (Século XX - 1960), facilitou a escrituração de programas razoavelmente complexos dividindo-os em módulos ou subprogramas.

Os módulos ou subprogramas são normalmente chamados em “Programação Estruturada” de **Procedimentos e Funções**.

- **Procedimentos**, (Procedures) são subprogramas ou sub-rotinas que tem como objetivo realizar ações. Podem ser utilizados para automatizar algum tipo de apresentação de tela como, por exemplo, traçar linhas horizontais, colocar determinadas frases na tela etc.
- **Funções** (Functions) são subprogramas ou sub-rotinas que normalmente calculam e fornecem valores.

Em “Programação Orientada a Objetos”, utiliza-se o conceito de **Métodos** ou “Funções Membros”, que possui conceitos e implementações semelhantes aos Procedimentos e as Funções da “Programação Estruturada”.

8.3. Método

8.3.1. Definição

Método é um conjunto de códigos de uma linguagem de programação que realizam operações predefinidas pela linguagem ou criadas pelo programador.

Basicamente existem dois tipos de Métodos:

- **Métodos Internos:** São aqueles já existentes na linguagem. Estes métodos podem ser considerados como recursos da linguagem.
- **Métodos externos:** São aqueles criados pelos programadores. Programadores caprichosos podem criar seus métodos e formar suas próprias bibliotecas, assim como incorporar à suas bibliotecas, módulos criados por outros programadores.

Em C# um namespace é um pacote de métodos.

8.3.2. Funcionamento

A construção de um método é feita com os mesmos comandos disponibilizados pela linguagem para escrituração de programas, ou seja, tudo que é executado por um método pode ser executado diretamente na linguagem. A utilização de Métodos tem como objetivo auxiliar na elaboração do trabalho do programador fazendo com que este, seja menos repetitivo, conseqüentemente mais produtivo.

8.3.3. Sintaxe

```
<qualificador> <tipo> <identificador> (parâmetro, parâmetro....)  
<comando>;  
<comando>;
```

Onde:

Qualificador: Define como será o método para o C#. Pode ser:

- **private:** Só pode ser acessado dentro da classe que foi criado.
- **public:** Pode ser acessado dentro ou fora da classe que foi criado.
- **protect:** Pode ser acessado dentro da classe que foi criado assim como pelas classes filhos das classes que estejam herdando as características da classe pai (classe que criou).

Tipo: Define o tipo de dado que irá retornar pela classe (Função). São utilizados os tipos de variáveis apresentados em aulas anteriores. Se for um Procedimento (não manipula valores), neste caso, deve ser utilizada a palavra **void**.

Identificador: É o nome do Método, segue as mesmas regras dos identificadores de variáveis.

Parâmetro: São os **tipos** e **identificadores** que são passados pelo programa que chamou o método.

8.3.4. Método tipo Procedimento

Um método tipo procedimento não recebe e não passa valores para o programa principal.

O exemplo fornecido possui dois Métodos que podem ser classificados como procedimentos. Um deles cujo identificador é **Linha()** tem a função de traçar uma linha horizontal na tela e é chamado diversas vezes durante o programa, o outro, cujo identificador é **Aguarda()**, tem a função de manter a tela aberta aguardando a digitação de uma tecla.

```
1. namespace Aula08_Ex01
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declaração de variáveis
8.             string nome;
9.             char continuar;
10.            double compra, venda;
11.
12.            // solicitação de dados
13.            do
14.            {
15.                Console.Write("Forneça o nome do produto: ");
16.                nome = Console.ReadLine();
17.                Console.Write("Forneça o custo do produto: ");
18.                compra = double.Parse(Console.ReadLine());
19.                // Chama subprograma Linha
20.                Linha();
21.
22.                // Calcula o preço de venda
23.                if (compra <= 100)
24.                    venda = compra * 1.8;
25.                else
26.                    venda = compra * 1.5;
27.
28.                // Informa o preço de venda
29.                Console.WriteLine("Preço venda será: {0} ", venda);
30.
31.                // Chama subprograma linha
32.                Linha();
33.
34.                Console.Write("Deseja calcular outro produto (S/N) ");
```



```

35.         continuar = char.Parse(Console.ReadLine());
36.         if ((continuar == 'S') || (continuar == 's'))
37.         {
38.             // Chama subprograma linha
39.             Linha();
40.             // Chama subprograma linha
41.             Linha();
42.         }
43.     }
44.     while ((continuar == 'S') || (continuar == 's'));
45.
46.     // Mantem tela aberta
47.     Aguarda();
48. }
49.
50. // Subprograma linha
51. public static void Linha()
52. {
53.     Console.WriteLine("-----");
54. }
55.
56. // Mantém a tela aberta aguardando a digitação de uma tecla
57. public static void Aguarda()
58. {
59.     Console.WriteLine("\nDigite qualquer tecla para encerrar o
60. programa.");
61.     Console.ReadKey();
62. }
63. }

```

8.3.5. Tela



```

C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula08_Ex01\...
Forneça o nome do produto: Sapato
Forneça o custo do produto: 100
-----
Preço venda será: 180
-----
Deseja calcular outro produto (S/N) S
-----
Forneça o nome do produto: Camisa
Forneça o custo do produto: 200
-----
Preço venda será: 300
-----
Deseja calcular outro produto (S/N)

```

8.3.6. Método tipo Função

Um método tipo função recebe e pode passar valores para o programa principal.

Este é o mesmo exemplo do item anterior, a diferença é que o cálculo do valor de venda é feito por uma função cujo identificador é **Calculo()**.

É importante notar que o “Método Calculo” recebe do programa principal que é o main o valor do preço de compra que é identificado por compra. Portanto, a sintaxe da chamada da função é Calculo(compra).

Os métodos **linha()** e **aguarda** continuam a ser utilizados neste exemplo.

```

1. namespace Aula08_Ex02
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declaração de variáveis
8.             string nome;
9.             char continuar;
10.            double compra;
11.
12.            // solicitação de dados
13.            do
14.            {
15.                Console.Write("Forneça o nome do produto: ");
16.                nome = Console.ReadLine();
17.                Console.Write("Forneça o custo do produto: ");
18.                compra = double.Parse(Console.ReadLine());
19.
20.                // Chama subprograma Linha
21.                Linha();
22.
23.                // Chama subprograma Calculo
24.                Calculo(compra);
25.
26.                Console.Write("Deseja calcular outro produto (S/N) ");
27.                continuar = char.Parse(Console.ReadLine());
28.                if ((continuar == 'S') || (continuar == 's'))
29.                {
30.                    // Chama subprograma Linha
31.                    Linha();
32.                    // Chama subprograma Linha
33.                    Linha();
34.                }
35.            }
36.            while ((continuar == 'S') || (continuar == 's'));
37.
38.            // Mantém tela aberta
39.            Aguarda();
40.        }
41.        // Subprograma Linha
42.        public static void Linha()
43.        {
44.            Console.WriteLine("-----");
45.        }
46.        // Subprograma Calculo
47.        public static void Calculo(double compra)
48.        {
49.            double venda;
50.            if (compra <= 100)
51.                venda = compra * 1.8;
52.            else
53.                venda = compra * 1.5;
54.        }
55.    }
56. }

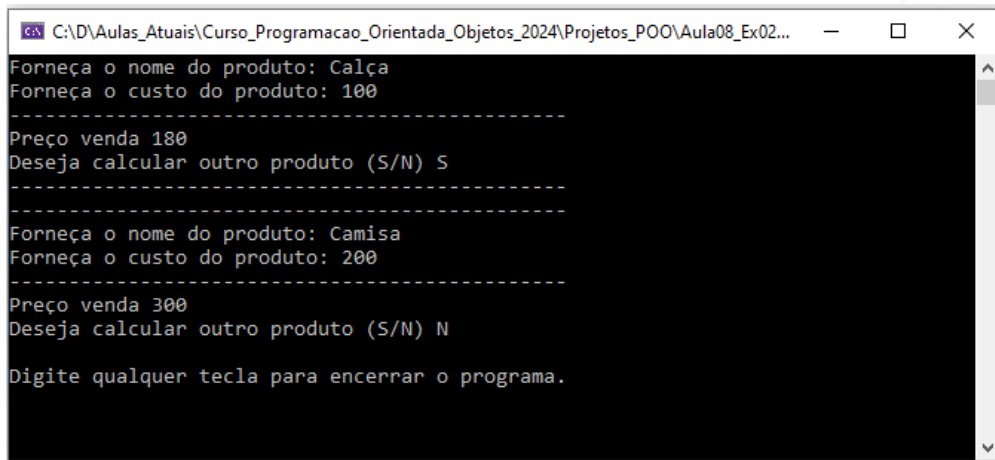
```

```

55.         Console.WriteLine("Preço venda {0} ", venda);
56.     }
57.
58.     // Mantém a tela aberta aguardando a digitação de uma tecla
59.     public static void Aguarda()
60.     {
61.         Console.WriteLine("\nDigite qualquer tecla para encerrar o
programa.");
62.         Console.ReadKey();
63.     }
64.
65. }
66. }

```

8.3.7. Tela



8.4. Variáveis Locais e Variáveis Globais

Variáveis locais só podem ser utilizadas no programa ou subprograma que as declara, ou seja, se a variável local é declarada no programa principal `main()`, ela só pode ser utilizada neste programa, caso se deseje utilizar o conteúdo destas variáveis em subprogramas elas tem que ser passadas para o subprograma e este deverá atribuir os conteúdos a outras variáveis locais a este subprograma.

Variáveis globais são aquelas que podem ser utilizadas tanto no programa principal `main()` como nos subprogramas do mesmo namespace. As variáveis globais são declaradas como **public** de maneira a tornarem-se públicas, após o item `class` (que abre a classe), **antes do início do programa principal** (`static void main(string[] args)`).

8.4.1. Sintaxe da declaração de Variáveis Globais

```
public static <tipo> <identificador>
```

Onde:

public: determina que o recurso se torna público ou seja visível de forma global.

Nota: caso a variável não seja global não é necessário digitar o domínio (public). Esta é a maneira como as variáveis foram declaradas nos programas anteriores do curso.

static: indica que o conteúdo permanecerá em uma parte da memória, sendo visível a outros métodos deste namespace.

Tipo: Define o tipo de dado que será declarado.

Identificador: É o nome da variável.

8.4.2. Exemplo

Transformando a variável “**vendas**” do exemplo anterior em variável global, o valor de venda pode voltar a ser informado diretamente pelo programa principal, visto que ele é calculado no método de nome “Calculo”, mas como “venda” é uma variável global ela pode ser utilizada tanto no “Método Principal” (main) como no “Método calculo”.

```

1. namespace Aula08_Ex03
2. {
3.     internal class Program
4.     {
5.         // Declaração de variável global
6.         public static double venda;
7.
8.         static void Main(string[] args)
9.         {
10.            // Declaração de variáveis
11.            string nome;
12.            char continuar;
13.            double compra;
14.
15.            // solicitação de dados
16.            do
17.            {
18.                Console.Write("Forneça o nome do produto: ");
19.                nome = Console.ReadLine();
20.                Console.Write("Forneça o custo do produto: ");
21.                compra = double.Parse(Console.ReadLine());
22.                Console.WriteLine("Produto: {0}, Preço Compra {1:F2}, ", nome,
compra);
23.
24.                // Chama subprograma Linha
25.                Linha();
26.
27.                // Chama subprograma calculo

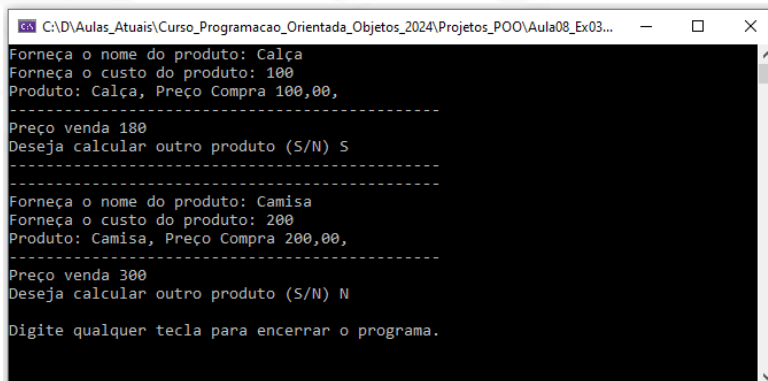
```

```

28.         Calculo(compra);
29.         Console.WriteLine("Preço venda {0} ", venda);
30.
31.         Console.Write("Deseja calcular outro produto (S/N) ");
32.         continuar = char.Parse(Console.ReadLine());
33.
34.         if ((continuar == 'S') || (continuar == 's'))
35.         {
36.             // Chama subprograma Linha
37.             Linha();
38.             // Chama subprograma Linha
39.             Linha();
40.         }
41.
42.     }
43.     while ((continuar == 'S') || (continuar == 's'));
44.
45.     // Manté tela aberta
46.     Aguarda();
47. }
48.
49. // Subprograma Linha
50. public static void Linha()
51. {
52.     Console.WriteLine("-----");
53. }
54.
55. // Subprograma Calculo
56. public static void Calculo(double compra)
57. {
58.     if (compra <= 100)
59.         venda = compra * 1.8;
60.     else
61.         venda = compra * 1.5;
62. }
63.
64. // Mantém a tela aberta aguardando a digitação de uma tecla
65. public static void Aguarda()
66. {
67.     Console.WriteLine("\nDigite qualquer tecla para encerrar o
68. programa.");
69.     Console.ReadKey();
70. }
71. }
72. }

```

8.4.3. Tela



```

C:\DVAulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula08_Ex03...
Forneça o nome do produto: Calça
Forneça o custo do produto: 100
Produto: Calça, Preço Compra 100,00,
-----
Preço venda 180
Deseja calcular outro produto (S/N) S
-----
Forneça o nome do produto: Camisa
Forneça o custo do produto: 200
Produto: Camisa, Preço Compra 200,00,
-----
Preço venda 300
Deseja calcular outro produto (S/N) N
-----
Digite qualquer tecla para encerrar o programa.

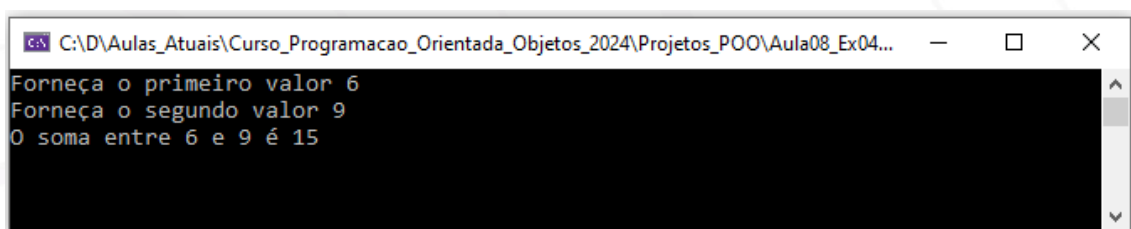
```

8.4.4. Exemplo

A função return, devolve o valor para o sistema e é necessária para algumas variáveis.

```
1. namespace Aula08_Ex04
2. {
3.     internal class Program
4.     {
5.         // Declaração de variáveis globais
6.         public static float r;
7.
8.         static void Main(string[] args)
9.         {
10.            // Declaração de variáveis locais
11.            float v1, v2;
12.
13.            // Tomada de valores
14.            Console.Write("Forneça o primeiro valor ");
15.            v1 = float.Parse(Console.ReadLine());
16.            Console.Write("Forneça o segundo valor ");
17.            v2 = float.Parse(Console.ReadLine());
18.            r = calculo(v1, v2);
19.            Console.WriteLine("O soma entre {0} e {1} é {2} ", v1, v2, r);
20.
21.            Console.ReadKey();
22.        }
23.        public static float Calculo(float a, float b)
24.        {
25.            r = a + b;
26.            return r;
27.        }
28.    }
29. }
```

8.4.5. Tela



```
C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula08_Ex04...
Forneça o primeiro valor 6
Forneça o segundo valor 9
O soma entre 6 e 9 é 15
```

9. VARIÁVEIS INDEXADAS

9.1. Objetivo

Apresentar e discutir o conceito de Variáveis Indexadas.

9.2. Introdução

Muitas vezes, um programa precisa receber diversos dados do mesmo tipo como, por exemplo, o nome e a idade de diversas pessoas.

Para receber estes dados o programador precisa criar diversas variáveis. Neste caso, a tendência dos desenvolvedores é criar variáveis com o mesmo nome possuindo no final um número, tal como: nome1, nome2, nome3. etc.

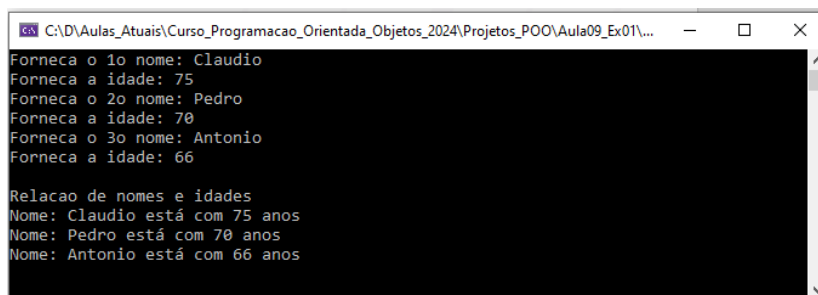
O exemplo a seguir demonstra este procedimento.

9.2.1. Codificação sem uso de variável indexada

```
1. namespace Aula09_Ex01
2. {
3.     internal class Program
4.     {
5.
6.         static void Main(string[] args)
7.         {
8.             // Declara variáveis
9.             string nome1, nome2, nome3;
10.            byte idade1, idade2, idade3;
11.
12.            // Solicita os nomes e idade
13.            Console.WriteLine("Forneca o 1o nome: ");
14.            nome1 = Console.ReadLine();
15.            Console.WriteLine("Forneca a idade: ");
16.            idade1 = byte.Parse(Console.ReadLine());
17.            Console.WriteLine("Forneca o 2o nome: ");
18.            nome2 = Console.ReadLine();
19.            Console.WriteLine("Forneca a idade: ");
20.            idade2 = byte.Parse(Console.ReadLine());
21.            Console.WriteLine("Forneca o 3o nome: ");
22.            nome3 = Console.ReadLine();
23.            Console.WriteLine("Forneca a idade: ");
24.            idade3 = byte.Parse(Console.ReadLine());
25.
26.            // Fornece os nomes e idades
27.            Console.WriteLine("\nRelacao de nomes e idades");
28.            Console.WriteLine("Nome: {0} está com {1} anos", nome1, idade1);
29.            Console.WriteLine("Nome: {0} está com {1} anos", nome2, idade2);
30.            Console.WriteLine("Nome: {0} está com {1} anos", nome3, idade3);
31.        }
32.    }
33. }
```

```
32.          // Mantém a tela aberta esperando a digitação de uma tecla
33.          Console.ReadKey();
34.      }
35.  }
36. }
```

9.2.2. Tela



Conforme pode ser observado, a programação se torna muito trabalhosa e extensa, visto que o programa acima só recebe 3 nomes e 3 idades, imagine se a necessidade fosse receber 100 nomes.

Para facilitar esta tarefa, o C#, assim como a maioria das linguagens de programação possui o que se chama de Variável Indexada ou arranjos³.

Estes índices podem ser unidimensionais ou multidimensionais

9.3. Variável Indexada Unidimensional

9.3.1. Definição

Consiste em criar um conjunto de variáveis indexadas numericamente, deste 0 até o valor desejado.

9.3.2. Sintaxe

A sintaxe para criação de variáveis indexadas é:

tipo[] identificador = new tipo:[dimensão];

onde:

tipo: é o tipo de variável que será criada;

³ Alguns autores também uso os termos matrizes ou vetores.

identificador: é o identificador da variável;

dimensão: número de itens que serão criados a partir do 0 até o número indicado.

9.3.3. Exemplo

```

1. namespace Aula09_Ex02
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             string[] nome = new string[10];
8.             int[] idade = new int[10];
9.             int c, f;
10.            char cont;
11.
12.            c = 0;
13.            // Solicita os nomes e idade
14.            do
15.            {
16.                // Declara variáveis
17.                Console.WriteLine("Forneça o {0}o nome: ", c + 1);
18.                nome[c] = Console.ReadLine();
19.                Console.WriteLine("Forneça a idade do {0}o: ", c + 1);
20.                idade[c] = int.Parse(Console.ReadLine());
21.                Console.WriteLine("Deseja fornecer outro nome? (s/n) ");
22.                cont = char.Parse(Console.ReadLine().ToUpper());
23.                if (cont == 'N')
24.                    break;
25.                c = c + 1;
26.            }
27.            while (c <= 9);
28.
29.            // Atribui ao f o valor de c que encerrou o loop
30.            f = c;
31.            c = 0;
32.
33.            // Fornece os nomes e idades
34.            Console.WriteLine("\nRelação de nomes e idades:");
35.            do
36.            {
37.                Console.WriteLine("Nome {0} , está com {1} anos de idade",
38.                nome[c], idade[c]);
39.                c = c + 1;
40.            }
41.            while (c <= f);
42.
43.            // Mantém a tela aberta esperando a digitação de uma tecla
44.            Console.ReadKey();
45.        }
46.    }

```

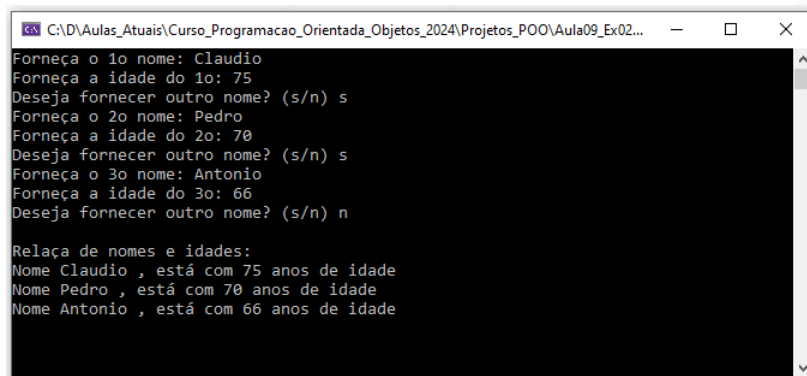
Conforme pode ser visto com a utilização de variáveis indexadas:

- Podem ser recebidos quantos nomes e idade o usuário desejar (No exemplo foram criadas variáveis indexadas de 0 a 9, porém, é possível facilmente

aumentar este número). É possível inclusive criar sub-rotinas ou classes para solicitar diretamente ao usuário quantos nomes e idades ele pretende lançar.

- A tomada de dados que foi feita com duas linhas por nome e idade no exemplo inicial, foi feita com apenas duas linhas para quantos usuários forem necessários.
- O fornecimento dos dados foi feito dentro de um loop for com uma única linha, independentemente do número de nomes e idades.

9.3.4. Tela:



```

C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objeto_2024\Projetos_POO\Aula09_Ex02...
Forneça o 1o nome: Claudio
Forneça a idade do 1o: 75
Deseja fornecer outro nome? (s/n) s
Forneça o 2o nome: Pedro
Forneça a idade do 2o: 70
Deseja fornecer outro nome? (s/n) s
Forneça o 3o nome: Antonio
Forneça a idade do 3o: 66
Deseja fornecer outro nome? (s/n) n

Relação de nomes e idades:
Nome Claudio , está com 75 anos de idade
Nome Pedro , está com 70 anos de idade
Nome Antonio , está com 66 anos de idade
  
```

9.3.5. Exemplo

O exemplo abaixo foi criado para discutir os índices das variáveis e como eles variam ao longo do programa.

```

1. namespace Aula09_Ex03
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declara variáveis
8.             double[] valor = new double[4];
9.             double soma = 0;
10.            double media = 0;
11.            byte c;
12.            // Solicita os valores
13.            for (c = 0; c <= 3; c++)
14.            {
15.                Console.WriteLine("Forneça o {0}o Valor: ", c + 1);
16.                valor[c] = double.Parse(Console.ReadLine());
17.                soma = soma + valor[c];
18.            }
19.            // Fornece a média
20.            media = soma / c;
21.            Console.WriteLine("A média dos {0} valores é {1} ", c, media);
22.
23.            // Mantem a tela aberta aguardando a digitação de uma tecla
24.            Console.ReadKey();
  
```

```
25.     }  
26.     }  
27. }
```

Neste programa deve ser observado:

- 1) Quando é criada a variável indexada com a instrução:

```
double[] valor = new double[4];
```

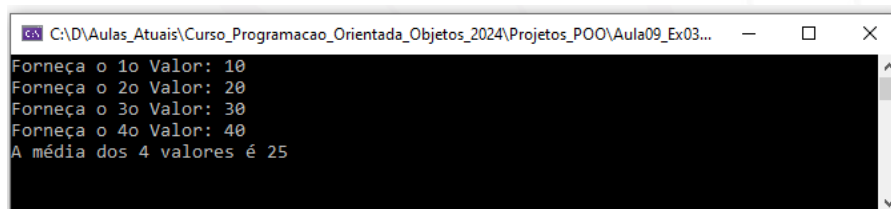
são criadas 4 variáveis do tipo double com os seguintes identificadores:

valor[0], valor[1], valor[2], valor[3], portanto o primeiro valor fornecido é atribuído à variável valor[0] e o quarto valor fornecido à variável valor[3], motivo pelo qual na linha `Console.WriteLine("Forneça o {0}o Valor: ", c + 1);`, o valor a ser atribuído à {0} é c+1

- 2) O “loop for”, é executado 4 vezes com o contador c variando de 0 até 3, mas o contador c é incrementado no final do loop, então, ao sair do loop o c vale 4. Isto pode ser confirmado pois a média é obtida pela expressão

```
media = soma / c;
```

9.3.6. Tela



```
C:\D\Aulas_Atuais\Curso_Programacao_Orientada_Objetos_2024\Projetos_POO\Aula09_Ex03...  
Forneça o 1o Valor: 10  
Forneça o 2o Valor: 20  
Forneça o 3o Valor: 30  
Forneça o 4o Valor: 40  
A média dos 4 valores é 25
```

9.4. Posicionar o curso na tela

9.4.1. Introdução

A utilização dos comandos `Write` e `WriteLine` permitem escrever sequencialmente na tela. Associados às instruções `\t` permitem ao programador diagramar a tela, de maneira a melhorar o aspecto dos programas, entretanto, caso se deseje melhorar o posicionamento das informações na tela, pode-se utilizar o comando “`SetCursorPosition`”, visto que este facilita a montagem de layouts mais trabalhosos.

9.4.2. Sintaxe

```
SetCursorPosition(c,l);
```

Onde:

c é o número da coluna;

l é o número da linha.

Exemplo

SetCursorPosition(15,10);

Coloca o cursor na coluna 15 e linha 10.

9.5. Variável Indexada Multidimensional

9.5.1. Definição

Consiste em criar um conjunto de variáveis indexadas numericamente com diversas dimensões. O comportamento é semelhante ao das variáveis indexadas unidimensionais, porém com possibilidades de criação de duas ou mais sequências de numerações.

9.5.2. Sintaxe

A sintaxe para criação de variáveis indexadas é:

tipo[,] identificador = new tipo:[dimensão1,dimesnsão2];

onde:

tipo: é o tipo de variável que será criada;

identificador: é o identificador da variável;

dimensões: são os números de itens de cada uma das dimensões que serão criados a partir do 0 até os números indicados.

9.5.3. Exemplo

```

1. namespace Aula09_Ex04
2. {
3.     internal class Program
4.     {
5.         static void Main(string[] args)
6.         {
7.             // Declaração de Variáveis
8.             int[,] A = new int[2, 2];
9.             byte c, l;
10.
11.             // Recebe os valores da matriz
12.             for (l = 0; l <= 1; l++)
13.             {
14.                 for (c = 0; c <= 1; c++)
15.                 {
16.                     Console.Write("Forneça o valor A {0}{1} ", l+1, c+1);
17.                     A[l, c] = int.Parse(Console.ReadLine());
18.                 }

```

```
19.         }
20.
21.         //Monta a matriz
22.         // Barra esquerda
23.         for (l = 9; l <= 13; l++)
24.         {
25.             Console.SetCursorPosition(8, l);
26.             Console.WriteLine("|");
27.         }
28.         // Barra direita
29.         for (l = 9; l <= 13; l++)
30.         {
31.             Console.SetCursorPosition(18, l);
32.             Console.WriteLine("|");
33.         }
34.
35.         // Coloca os números na matriz
36.         Console.SetCursorPosition(10, 10);
37.         Console.Write(A[0, 0]);
38.         Console.SetCursorPosition(15, 10);
39.         Console.Write(A[0, 1]);
40.         Console.SetCursorPosition(10, 12);
41.         Console.Write(A[1, 0]);
42.         Console.SetCursorPosition(15, 12);
43.         Console.Write(A[1, 1]);
44.
45.         //Mantém a tela aberta esperando a digitação de uma tecla
46.         Console.ReadKey();
47.
48.     }
49. }
50. }
```

9.5.4. Tela





Referências

CERTIFICATION, M. MCTS.NET Certification. MCTS.NET Certification, 2008. Disponível em: <<http://mctscertification.blogspot.com.br/2008/05/what-is-difference-between-byte-and.html>>

MICROSOFT. Biblioteca. MSDN, 2010. Disponível em: <<http://msdn.microsoft.com/pt-br/library/3ewxz6et>>. Acesso em: 20 ago. 2012.

SINTES, A. Aprenda programação orientada a objetos em 21 dias. São Paulo. Pearson, 2002.

MELO, A. V. SILVA, F. S. C. Princípios de linguagem de programação. São Paulo, Blucher, RS, 2014.

DEITEL, P. J. DEITEL, H. M. LISTFIELD, J. A. C#: como programar. São Paulo: Pearson 2003.

ASCENCIO, A.F. G.; CAMPOS, E. A. V. Fundamentos da programação de computadores: algoritmos, PASCAL, C/C++ e JAVA. 2ª ed. São Paulo: Pearson, 2007.

SOUZA, S G. Lógica de Programação Algorítmica. 1ª ed. São Paulo: Pearson, 2014.

SILVA, E. L. Programação de computadores. 1ª ed. São Paulo: Pearson, 2015.

DEITEL, P. J. DEITEL, H. M. C como programar. São Paulo: Pearson 2011.

ARAÚJO, S. Linguagem de programação (ADS). Curitiba. Contentus, 2020. (BV)