Programação C# pela ETEC

Referência rápida de comandos de terminal

Windows (cmd e Powershell) / Linux (bash ou qualquer outro)

|  |  |
| --- | --- |
| Comando | Descrição |
| dir | Lista o conteúdo da pasta atual (arquivos e pastas) |
| cd <pasta> = caminho | Acessa a pasta informada. |
| cls | Limpa o conteúdo da tela, facilitando a leitura. |
| md | Cria uma pasta |

code

|  |  |
| --- | --- |
| Comando | Descrição |
| code | Abre o VsCode. |
| code --version | Mostra a versão instalada do VsCode. |
| code --help | Mostra a tela de ajuda. |
| code . | Abre a pasta atual. |
| code -n . | Abre a pasta atual em uma nova janela. |
| code -r . | Abre a pasta atual na janela atual. |

dotnet

|  |  |
| --- | --- |
| Comando | Descrição |
| dotnet --help | Mostra a tela de ajuda. |
| dotnet --version | Mostra a versão instalada do .NET. |
| dotnet new console | Cria uma nova aplicação de console na pasta atual. |
| dotnet new console -o NomeProjeto | Cria uma nova aplicação de console em uma pasta dentro da pasta atual, com o nome indicado. |
| dotnet new -l | Lista os templates de projeto existentes. |
| dotnet build | Compila o projeto da pasta atual. |
| dotnet run | Compila e executa o projeto da pasta atual. |
| dotnet list reference | Lista os projetos aos quais o projeto atual faz referência. |
| dotnet list package | Lista os pacotes aos quais o projeto atual faz referência. |
| dotnet restore | Baixa os pacotes aos quais o projeto atual faz referência e ainda não existam localmente, bem como suas dependências. |
| dotnet add package <pacote> | Adiciona um pacote no projeto atual. |
| dotnet remove package <pacote> | Remove um pacote do projeto atual. |

Principais templates de aplicação:

|  |  |
| --- | --- |
| Template | Descrição |
| console | Aplicações de linha de comando em console (CLI). |
| classlib | Biblioteca de classes. |
| web | Sites para a web simples. |
| mvc | Sites utilizando o padrão de desenvolvimento MVC. |
| webapp | Sites utilizando o padrão de desenvolvimento MVC, com páginas Razor. |
| webapi | Serviços web. |
| winforms | Aplicações Desktop nativas do Windows. |
| wpf | Aplicações Desktop portáveis. |

Alguns pacotes notáveis:

|  |  |
| --- | --- |
| Pacote | Descrição |
| Humanizer | Permite gerar descrições textuais legíveis por humanos para diversos objetos. |
| Figgle | Cria ASCII-Art a partir de strings. |
| Noda Time | Biblioteca avançada para manipulação de Datas e Horas no .NET. |

git

|  |  |
| --- | --- |
| Comando | Descrição |
| git --help | Mostra a tela de ajuda. |
| git --version | Mostra a versão instalada do git. |
| git init | Cria um repositório local na pasta atual. |
| git clone <endereço> | Cria um repositório local a partir de um repositório remoto. |
| git add <arquivo> | Adiciona um arquivo na lista de alterações executadas. |
| git add . | Adiciona todos os arquivos alterados na lista de alterações executadas. |
| git add -i | Adiciona arquivos interativamente na lista de alterações executadas. |
| git commit -m "comentários das alterações" | Efetiva as alterações no branch atual do repositório local. |
| git push | Envia as alterações do branch atual para o repositório remoto atual. |
| git push origin master | Envia as alterações do branch local master para o repositório remoto origin. |
| git push <nome\_repositorio> <nome\_branch> | Envia as alterações do branch indicado para o repositório remoto indicado. |
| git remote | Lista os repositórios remotos. |
| git remote add <nome\_repositorio> <endereço> | Adiciona um repositório remoto. |
| git branch | Lista os branchs locais. |
| git checkout -b <nome\_branch> | Cria um novo branch baseado no branch atual e o torna ativo. |
| git checkout <nome\_branch> | Ativa um branch local existente. |
| git branch -d <nome\_branch> | Remove um branch local. |
| git push <nome\_remote> --delete <nome\_branch> | Remove um branch remoto. |
| git pull | Atualizar o repositório local com a versão mais recente do repositório remoto. |
| git merge <nome\_branch> | Mescla alterações do branch indicado no branch atual. |
| git diff <nome\_branch\_origem> <nome\_branch\_destino> | Mostra as diferenças entre dois branches. |
| git log | Mostra o histórico de alterações. |
| git log --oneline | Mostra o histórico de alterações de forma resumida, uma alteração por linha. |
| git tag <descricao> <10\_primeiros\_caracteres\_id\_commit> | Cria uma versão nomeada do código a partir de um commit existente. |
| git tag <descricao> -m "comentários" | Cria uma versão nomeada do código no commit atual, com comentários. |
| git checkout -- <arquivo> | Desfaz as alterações locais desde o último commit. |
| git push origin --tags | Envia as tags criadas para o repositório remoto. |
| git clean -d | Lista todos os arquivos e diretórios que não fazem parte do versionamento. |
| git clean -fdx | Exclui todos os arquivos e diretórios que não fazem parte do versionamento. |
| git stash | Adiciona as alterações na área de stash. |
| git stash list | Lista áreas de stash. |
| git stash pop | Recupera as alterações da área de stash. |
| Git config user.name | Verifica o nome que está na maquina |
| Git config user.name | Verifica o email que está na maquina |
| git config --global user.name "Seu Nome" | Configura o nome para a máquina |
| git config --global user.email "Seu\_EMail" | Configura o e-mail para a maquina |

Desfazer todas as alterações locais, voltando à situação do repositório remoto:

git fetch origin

git reset --hard origin/master

Primeiros passos com C#

Anatomia de um programa C#

Escreveremos nossos programas em arquivos texto contendo comandos da linguagem de programação, chamados códigos-fonte. Esses arquivos são organizados em projetos, sendo um projeto uma unidade de compilação, que gerará um executável, ou assembly (no jargão .NET).

Criemos de um projeto chamado ExemploConsole com o template console:

PS C:\Users\ermogenes\Desktop\code> dotnet new console -o ExemploConsole

Estrutura física

São criados 2 arquivos e uma pasta:



ExemploConsole.csproj contém as configurações do seu projeto.

Program.cs contém o código-fonte principal do seu programa.

obj contém arquivos diversos, na maioria binários, necessários à compilação do projeto.

Vejamos o conteúdo de ExemploConsole.csproj:

<Project *Sdk*="Microsoft.NET.Sdk">

  <PropertyGroup>

    <OutputType>Exe</OutputType>

    <TargetFramework>net7.0</TargetFramework>

    <ImplicitUsings>enable</ImplicitUsings>

    <Nullable>enable</Nullable>

  </PropertyGroup>

</Project>

Este arquivo está escrito em XML, e contém algumas configurações.

* OutputType indica o tipo de assembly resultante.
* TargetFramework indica qual a versão do runtime será necessária para rodar o aplicativo.
* ImplicitUsings e Nullable são configurações de como a linguagem se comportará (não se preocupe com isso agora).

Agora, vejamos Program.cs:

using *System*;

namespace *ProgramaParaTeste*

{

    class Program

    {

        static *void* Main(*string*[] *args*)

        {

           Console.WriteLine("Hola, Mundo!");

        }

    }

}

Executamos o código a cima com o dotnet run , feito isso foi criada uma pasta chamada bin .

Criada a pasta bin:



Dentro dela, em bin\Debug\net7.0 encontramos os assemblies.



Entre eles, ExemploConsole.exe, o nosso programa executável.

Iremos criar diversos outros arquivos .cs em nossos projetos, e podemos livremente organizá-los em pastas.

Estrutura lógica

C# é uma linguagem orientada a objetos, o que significa que (quase) tudo em nossos programas são classes.

Voltemos ao código:

using *System*;

namespace *ProgramaParaTeste*

{

    class Program

    {

        static *void* Main(*string*[] *args*)

        {

           Console.WriteLine("Hola, Mundo!");

        }

    }

}

Nesse código, você está definindo um namespace chamado "ProgramaParaTeste". O namespace é usado para organizar e agrupar classes relacionadas.

Dentro desse namespace, você tem uma classe chamada "Program". Essa é a classe principal do programa.

Dentro da classe "Program", você tem o método estático "Main". O método "Main" é o ponto de entrada do programa, onde a execução do programa começa. Ele recebe um array de strings chamado "args", que pode ser usado para passar argumentos para o programa a partir da linha de comando.

Dentro do método "Main", você pode escrever o código que deseja executar. Você pode adicionar outras classes, métodos e lógica conforme necessário para alcançar o objetivo do seu programa.

Lembre-se de fechar qualquer recurso que você tenha aberto, como arquivos ou conexões de banco de dados, para garantir que seu programa funcione corretamente e não deixe recursos em uso.

Quando o programa atingir o final do método "Main" ou encontrar uma instrução explícita de saída, a execução será encerrada e o programa será finalizado.

Saída em console

Comandos disponíveis na classe System.Console.

Exibir um texto, sem quebrar linha

Console.Write

O código Console.Write("Primeiro texto"); é usado para exibir o texto "Primeiro texto" no console em uma aplicação C#.

O método Console.Write() é responsável por exibir um valor ou uma mensagem no console, sem pular para uma nova linha. Isso significa que, após exibir o texto, o cursor permanecerá na mesma linha.

Exemplo:

{

  Console.Write("Primeiro texto");

  Console.Write("Segundo texto");

}

Resultado:

Primeiro textoSegundo texto

Exibir um texto, com quebra de linha no final

Console.WriteLine

O código Console.WriteLine("Segundo texto"); é usado para exibir o texto "Segundo texto" no console em uma aplicação C#.

O método Console.WriteLine() é responsável por exibir um valor ou uma mensagem no console, acrescentando automaticamente uma quebra de linha após a exibição. Isso significa que, após exibir o texto, o cursor será movido para uma nova linha.

No caso específico do código fornecido, o texto "Segundo texto" será exibido no console e o cursor será movido para uma nova linha. O resultado será algo como:

{

  Console.Write("Primeiro texto");

  Console.WriteLine("Segundo texto");

  Console.Write("Terceiro texto");

}

Resultado:

Primeiro textoSegundo texto

Terceiro texto

Limpar todos os textos da tela

Console.Clear();

O código Console.Clear(); é usado para limpar o conteúdo do console em uma aplicação C#. Ele remove todas as mensagens e caracteres exibidos anteriormente, deixando o console vazio.

Quando você chama o método Console.Clear(), a tela do console é apagada e o cursor é movido para a posição superior esquerda, aguardando novas instruções. Isso proporciona uma "limpeza" visual no console, removendo qualquer texto ou informações anteriores.

A utilização do Console.Clear() é útil quando você deseja começar uma nova seção de exibição no console, ou quando precisa atualizar a interface de usuário do console sem a interferência de conteúdo anterior.

Por exemplo, se você tiver várias mensagens exibidas no console e quiser "limpar" a tela antes de mostrar um novo conjunto de informações, pode usar o Console.Clear() da seguinte maneira:

{

  Console.Clear();

  Console.WriteLine("Novas informações");

}

Tocar um sinal sonoro

Console.Beep();

⚠ Beep é dependente de plataforma, ou seja, só funciona em ambiente Windows. Isso causa uma aviso do compilador, em amarelo. Para desativar esse aviso, marque o trecho com a notação pragma:

// ...

// a partir daqui o aviso não será dado pelo compilador

#pragma warning disable CA1416

// ...

Console.Beep();

// ...

#pragma warning restore CA1416

// reativa o aviso daqui em diante

// ...

{

  Console.Beep();

}

Para alterar a frequência e o tempo do sinal sonoro, passamos dois argumento à Beep. No exemplo, tocamos a nota dó por 200 milésimos de segundo:

Console.Beep(1320, 200);

Algumas notas:

|  |  |
| --- | --- |
| Nota | Frequência |
| Dó | 1320 |
| Ré | 1485 |
| Mi | 1650 |
| Fá | 1759 |
| Sol | 1980 |
| Lá | 2200 |
| Si | 2475 |

{

  Console.Beep(1320,200);

  Console.WriteLine("dó") ;

  Console.Beep(1485,200);

   Console.WriteLine("ré");

}

Assuntos sobre frequência musical

<http://www.das.inpe.br/~alex/FisicadaMusica/fismus_escalas.htm>

<https://learn.microsoft.com/pt-br/dotnet/api/system.console.beep?view=net-6.0>

<https://pages.mtu.edu/~suits/notefreqs.html>

<https://pages.mtu.edu/~suits/notefreqs.html>

Fazer uma pausa na execução do programa

Thread.Sleep();

É necessário passar a duração da pausa em milésimos de segundo (use 1000 para 1 segundo).

O método Thread.Sleep() é usado para suspender a execução de uma thread (ou seja, uma unidade de execução dentro de um programa) por um determinado período de tempo em milissegundos. Ele faz com que a thread atual pare de executar temporariamente antes de continuar.

A sintaxe do método Thread.Sleep() é a seguinte:

Thread.Sleep(milissegundos);

O parâmetro "milissegundos" indica a quantidade de tempo que a thread deve ficar suspensa, em milissegundos. Durante esse período, a thread não executará nenhuma instrução.

O Thread.Sleep() é útil em situações em que você deseja adicionar atrasos ou pausas na execução do programa. Por exemplo, pode ser usado para criar uma espera entre a exibição de mensagens no console, para simular um processo de espera ou para controlar a taxa de execução de um loop.

Aqui está um exemplo de como usar o Thread.Sleep() para adicionar um atraso de 1 segundo (1000 milissegundos) na execução:

{

  Console.WriteLine("inicio");

  Thread.Sleep(1000);

  Console.WriteLine("fim");

}

Nesse exemplo, a mensagem "Início" será exibida imediatamente, e em seguida, a thread será suspensa por 1 segundo usando Thread.Sleep(1000). Após o atraso, a mensagem "Fim" será exibida.

É importante ter cuidado ao usar Thread.Sleep() em threads principais (como a thread principal de uma aplicação de console ou de uma interface gráfica). Se a thread principal for suspensa por um longo período de tempo, a aplicação pode parecer congelada ou não responsiva para o usuário. Em alguns casos, é preferível usar técnicas de assincronia para evitar bloqueios.

Mudar a cor da letra

Console.ForegroundColor();

Cores disponíveis no enumerador System.ConsoleColor:

O método Console.ForegroundColor() é usado para definir a cor do texto exibido no console em uma aplicação C#. Ele permite alterar a cor do texto para tornar a exibição mais visualmente interessante ou significativa.

A sintaxe do método Console.ForegroundColor() é a seguinte:

Console.ForegroundColor = ConsoleColor.Valor;

O parâmetro "Valor" indica a cor desejada para o texto. Ele deve ser um valor da enumeração ConsoleColor, que possui uma variedade de opções de cores pré-definidas.

Aqui estão alguns exemplos de como usar o Console.ForegroundColor():

{

  Console.ForegroundColor = *ConsoleColor*.DarkBlue;

  Console.WriteLine("Texto em Azul");

  Console.ForegroundColor = *ConsoleColor*.DarkRed;

   Console.WriteLine("Texto em Vermelho");

   Console.ForegroundColor = *ConsoleColor*.DarkYellow;

   Console.WriteLine("Texto em Amarelo");

   Console.ResetColor();

}

Nesses exemplos, cada chamada de Console.ForegroundColor define a cor do texto e, em seguida, o texto correspondente é exibido com a cor selecionada.

Após a exibição do texto com uma cor específica, se você deseja restaurar a cor padrão do console, pode usar o método Console.ResetColor(); como mostra no exemplo acima.

Isso fará com que a cor do texto volte à cor padrão do console.

Lembre-se de que a aparência e as cores disponíveis podem variar dependendo do terminal ou do ambiente em que o programa está sendo executado.

Usando o Numerador:

|  |  |  |
| --- | --- | --- |
| Constante | Valor | Cor |
| Black | 0 | A cor preta. |
| Blue | 9 | A cor azul. |
| Cyan | 11 | A cor ciano (verde-azul). |
| DarkBlue | 1 | A cor azul-escuro. |
| DarkCyan | 3 | A cor ciano-escuro (azul escuro-verde). |
| DarkGray | 8 | A cor cinza-escuro. |
| DarkGreen | 2 | A cor verde-escuro. |
| DarkMagenta | 5 | A cor magenta-escuro (arroxeado-escuro-vermelho). |
| DarkRed | 4 | A cor vermelho-escuro. |
| DarkYellow | 6 | A cor amarelo-escuro (ocre). |
| Gray | 7 | A cor cinza. |
| Green | 10 | A cor verde. |
| Magenta | 13 | A cor magenta (arroxeado-vermelho). |
| Red | 12 | A cor vermelha. |
| White | 15 | A cor branca. |
| Yellow | 14 | A cor amarela. |

Código:

{

  Console.ForegroundColor = (*ConsoleColor*)9;

  Console.WriteLine("Texto em Azul");

  Console.ForegroundColor = (*ConsoleColor*)4;

  Console.WriteLine("Texto em Vermelho");

  Console.ForegroundColor = (*ConsoleColor*)6;

  Console.WriteLine("Texto em Amarelo");

  Console.ResetColor();

}

Mudar a cor do fundo

Console.BackgroundColor();

Cores disponíveis no enumerador System.ConsoleColor:

O método Console.BackgroundColor() é usado para definir a cor de fundo do console em uma aplicação C#. Ele permite alterar a cor de fundo do console para tornar a exibição mais visualmente interessante ou significativa.

A sintaxe do método Console.BackgroundColor() é a seguinte:

Console.BackgroundColor = ConsoleColor.Valor;

Usando o enumerador:

{

  Console.BackgroundColor = *ConsoleColor*.Blue;

  Console.ForegroundColor = *ConsoleColor*.DarkYellow;

  Console.WriteLine("Texto em Amarelo");

  Console.ResetColor();

}

Usando o código numérico:

{

  Console.BackgroundColor = (*ConsoleColor*)15;

Console.ForegroundColor = (*ConsoleColor*)9;

  Console.WriteLine("Texto em Azul");

Console.ResetColor();

}

O parâmetro "Valor" indica a cor desejada para o fundo do console. Assim como o método Console.ForegroundColor(), ele deve ser um valor da enumeração ConsoleColor, que possui uma variedade de opções de cores pré-definidas.,

Nesses exemplos, cada chamada de Console.BackgroundColor define a cor de fundo do console e, em seguida, o texto correspondente é exibido com o fundo na cor selecionada.

Assim como a cor do texto, se você deseja restaurar a cor de fundo padrão do console, pode usar o método Console.ResetColor();

Isso fará com que a cor de fundo do console volte à cor padrão.

Lembre-se de que a aparência e as cores disponíveis podem variar dependendo do terminal ou do ambiente em que o programa está sendo executado.

Cursor

Esconder/mostrar o cursor:

{

  Console.CursorVisible = false; // cursor não visível

  Console.CursorVisible = true;  // cursor visível

}

Os comandos Console.CursorVisible = false; e Console.CursorVisible = true; são usados para controlar a visibilidade do cursor no console em uma aplicação C#.

Quando você define Console.CursorVisible = false;, o cursor fica invisível no console. Isso é útil quando você não deseja que o cursor seja exibido durante a exibição de informações ou quando quer criar uma interface mais limpa no console.

É importante mencionar que o estado de visibilidade do cursor é específico para cada aplicação e não é afetado por comandos ou alterações feitas em outros aplicativos no console.

Posição atual do cursor:

Console.CursorTop

A propriedade Console.CursorTop é usada para obter ou definir a posição atual do cursor na coordenada vertical (linha) no console em uma aplicação C#.

A propriedade CursorTop retorna um valor inteiro que representa a posição da linha atual do cursor no console. A primeira linha tem o valor 0, a segunda linha tem o valor 1 e assim por diante.

Você também pode atribuir um novo valor a Console.CursorTop para definir a posição do cursor em uma linha específica.

Aqui está um exemplo:

{

*int* linhacursor = Console.CursorTop;

 Console.WriteLine("posição atual do cursor " + linhacursor);

  Console.CursorTop = 5;

Console.WriteLine("Nova posição do cursor " + Console.CursorTop);

}

Console.CursorLeft

A propriedade Console.CursorLeft é usada para obter ou definir a posição atual do cursor na coordenada horizontal (coluna) no console em uma aplicação C#.

A propriedade CursorLeft retorna um valor inteiro que representa a posição da coluna atual do cursor no console. A primeira coluna tem o valor 0, a segunda coluna tem o valor 1 e assim por diante.

Você também pode atribuir um novo valor a Console.CursorLeft para definir a posição do cursor em uma coluna específica.

Aqui está um exemplo:

{

*int* colunaCursor = Console.CursorLeft;

 Console.WriteLine("posição atual do cursor " + linhacursor);

  Console.CursorLeft = 10;

  Console.WriteLine("Nova posição do cursor " + Console.CursorTop);

}

No exemplo acima, colunaAtual armazena o valor atual da coluna do cursor. Em seguida, é exibida a posição atual do cursor na coluna. Após isso, Console.CursorLeft é definido como 10 para mover o cursor para a coluna 10. Por fim, é exibida a nova posição do cursor na coluna.

Console.SetCursorPosition();

O método Console.SetCursorPosition() é usado para definir a posição do cursor em uma coordenada específica no console em uma aplicação C#.

A sintaxe do método Console.SetCursorPosition() é a seguinte:

Console.SetCursorPosition(coluna, linha);

Os parâmetros coluna e linha especificam a posição desejada do cursor no console. A primeira coluna tem o valor 0 e a primeira linha também tem o valor 0. Você pode fornecer valores inteiros para definir a posição do cursor na coluna e linha desejadas.

Aqui está um exemplo:

{

   Console.SetCursorPosition(10, 5);

 Console.WriteLine("o cursor está aqui ");

}

Nesse exemplo, Console.SetCursorPosition(10, 5) define a posição do cursor na coluna 10 e linha 5. Em seguida, a mensagem "Cursor está aqui!" é exibida nessa posição específica do console.

Ao usar Console.SetCursorPosition(), é importante garantir que os valores fornecidos estejam dentro dos limites do console. Caso contrário, ocorrerão exceções.

Você também pode combinar o uso de Console.SetCursorPosition() com a propriedade Console.CursorLeft e Console.CursorTop para obter a posição atual do cursor e, em seguida, modificá-la conforme necessário.

Pacotes para renderização em terminal

Caso busque algo com mais recursos, tente:

<https://spectreconsole.net/>

<https://github.com/gui-cs/Terminal.Gui>

<https://github.com/Thraka/SadConsole>

<https://bitbucket.org/clarktravism/rlnet>

<https://docs.monogame.net/>

Aguardando a entrada do usuário.

Console.ReadKey();

O método Console.ReadKey() é usado para ler a entrada do teclado no console em uma aplicação C#. Ele aguarda até que o usuário pressione uma tecla e, em seguida, retorna um objeto do tipo ConsoleKeyInfo que contém informações sobre a tecla pressionada.

Aqui está um exemplo de como usar o Console.ReadKey():

{

 Console.WriteLine("Pressione uma tecla");

*ConsoleKeyInfo* teclapressionada =  Console.ReadKey();

 Console.WriteLine(" você pressionou a tecla " + teclapressionada.Key);

 Console.WriteLine(" Caractere da tecla pressionada " +

teclapressionada.KeyChar);

 Console.WriteLine(" Tecla modificafora " + teclapressionada.Modifiers);

}

Nesse exemplo, o programa exibe uma mensagem para o usuário pressionar uma tecla. Em seguida, o método Console.ReadKey() é chamado, aguardando a entrada do usuário. Quando o usuário pressiona uma tecla, as informações sobre a tecla pressionada são armazenadas em um objeto ConsoleKeyInfo chamado teclaPressionada.

Você pode acessar diferentes propriedades do objeto ConsoleKeyInfo. Alguns exemplos incluem:

* Console.ReadKey(true); Para não exibir a tecla pressionada
* teclaPressionada.Key: retorna a tecla específica pressionada, representada por um valor da enumeração ConsoleKey.
* teclaPressionada.KeyChar: retorna o caractere associado à tecla pressionada.
* teclaPressionada.Modifiers: retorna as teclas modificadoras (como Shift, Ctrl, Alt) pressionadas simultaneamente com a tecla principal.

Você pode usar essas informações para executar diferentes ações com base na tecla pressionada pelo usuário.

Observe que Console.ReadKey() aguardará indefinidamente até que o usuário pressione uma tecla. Se você deseja limitar o tempo de espera por uma entrada, pode usar outras técnicas, como combinar Console.ReadKey() com um temporizador ou a utilização de threads.

ConsoleModifiers

ConsoleModifiers é uma enumeração em C# que representa as teclas modificadoras do teclado, como Alt, Shift e Control. A enumeração ConsoleModifiers possui os seguintes valores:

* Alt: Representa a tecla Alt.
* Shift: Representa a tecla Shift.
* Control: Representa a tecla Control.

Esses valores são usados em conjunto com o operador bitwise & para verificar se as teclas modificadoras estão pressionadas juntamente com a tecla principal lida através do método Console.ReadKey().

Ao usar a enumeração ConsoleModifiers, você pode verificar as combinações de teclas modificadoras pressionadas pelos usuários para executar ações diferentes em sua aplicação.

Veja um exemplo:

{

  Console.WriteLine("Pressione uma tecla");

  var tecla = Console.ReadKey(true);

*bool* pressinadoAlt = (tecla.Modifiers & *ConsoleModifiers*.Alt) != 0;

*bool* pressionadoShift = (tecla.Modifiers & *ConsoleModifiers*.Shift) != 0;

*bool* pressionadoConstrol = (tecla.Modifiers & *ConsoleModifiers*.Control) != 0;

  if (pressinadoAlt) Console.Write("Alt ");

  if (pressionadoShift) Console.Write("Shift ");

  if (pressionadoConstrol) Console.Write("Control ");

  Console.WriteLine(tecla.Key);

}

No exemplo fornecido, são utilizadas as propriedades Modifiers e as enumerações ConsoleModifiers para verificar se teclas modificadoras específicas, como Alt, Shift e Control, estão pressionadas juntamente com a tecla principal lida através do Console.ReadKey().

Atribuem o valor true às variáveis pressionadoAlt, pressionadoShift e pressionadoControl se as teclas modificadoras Alt, Shift e Control estiverem pressionadas, respectivamente, juntamente com a tecla principal lida.

Essas linhas de código utilizam a operação de bitwise AND (&) entre o valor de tecla.Modifiers e as enumerações ConsoleModifiers.Alt, ConsoleModifiers.Shift e ConsoleModifiers.Control. Se o resultado dessa operação for diferente de zero, significa que a tecla modificadora correspondente está pressionada.

Essa abordagem é útil quando você precisa verificar quais teclas modificadoras estão sendo usadas em combinação com a tecla principal. Por exemplo, você pode executar ações diferentes com base nas combinações de teclas pressionadas.

Console.ReadLine();

O método Console.ReadLine() é usado para ler uma linha de texto digitada pelo usuário no console em uma aplicação C#. Ele aguarda até que o usuário pressione a tecla Enter após digitar o texto e, em seguida, retorna uma string contendo o texto digitado.

Aqui está um exemplo de como usar o Console.ReadLine():

{

  Console.WriteLine("Digite seu nome");

*string* nome = Console.ReadLine();

  Console.WriteLine("Boa Noite seu " + nome);

}

Nesse exemplo, o programa exibe a mensagem "Digite seu nome: " no console, aguardando que o usuário digite seu nome e pressione Enter. O texto digitado pelo usuário é lido usando Console.ReadLine() e armazenado na variável nome. Em seguida, uma mensagem de boas-vindas é exibida no console, usando o valor digitado pelo usuário.

Console.ReadLine() retorna uma string que representa a linha de texto digitada pelo usuário, incluindo todos os caracteres até que o usuário pressione Enter. Se o usuário não digitar nada e apenas pressionar Enter, a string retornada será vazia.

Você pode usar o Console.ReadLine() para interagir com o usuário, solicitar entrada de informações, criar menus interativos e muito mais.

Limpar o buffer de teclado

while (Console.KeyAvailable) Console.ReadKey(true);

A instrução while (Console.KeyAvailable) Console.ReadKey(true); é usada para limpar o buffer do teclado, descartando todas as teclas pressionadas que estão atualmente no buffer.

Quando o usuário pressiona uma tecla, essa tecla é armazenada em um buffer interno antes de ser lida pela chamada ao método Console.ReadKey(). No entanto, se houver teclas pressionadas no buffer que ainda não foram lidas, o método Console.ReadKey() irá lê-las a partir do buffer em vez de esperar pela entrada do usuário.

A instrução while (Console.KeyAvailable) verifica se há teclas disponíveis no buffer do teclado. Se houver, o loop continuará a chamar Console.ReadKey(true) para ler e descartar essas teclas pressionadas, utilizando o parâmetro true para que elas não sejam exibidas no console.

Essa construção é útil quando você deseja descartar quaisquer teclas pressionadas que não tenham sido processadas e evitar que elas afetem a lógica do seu programa.

É importante observar que Console.KeyAvailable e Console.ReadKey(true) são métodos síncronos e bloqueantes. O loop while (Console.KeyAvailable) irá continuar até que todas as teclas pressionadas no buffer tenham sido lidas e descartadas.

Lembre-se de utilizar essa construção com cautela, pois ela pode interferir na interatividade da sua aplicação. Em alguns casos, pode ser necessário utilizar técnicas mais avançadas, como manipulação de threads, para tratar a entrada do usuário de forma mais complexa.

Trabalhando com strings

O básico

Uma string é uma cadeia (sequência, arranjo) de caracteres individuais. Por exemplo, a string Etec é formada pelos caracteres E, t, e e c. No .NET, por padrão, é utilizada a codificação UTF-16.

Para definir uma string em C# utilizamos as aspas duplas (").

Você já conheceu as strings quando estudou o comando Console.WriteLine:

 Console.WriteLine("*Etec* *Adolpho* Berezin");

Nesse exemplo, a string é definida em "Etec Adolpho Berezin" e passada como argumento para o método Console.WriteLine.

Podemos armazenar uma string em memória utilizando variáveis do tipo string:

string nome = "Ermogenes";

*string* sobrenome = "Palacio";

Concatenação

A concatenação de strings é o processo de combinar duas ou mais strings em uma única string. Em C#, você pode realizar a concatenação de strings usando o operador de adição (+), o método Concat() da classe string ou a interpolação de strings.

Digamos que agora queremos uma variável contendo o nome completo. Precisamos instruir o computador a juntar os conteúdos das variáveis. Esse processo pode ser feito de várias maneiras.

Vejamos:

1. Usando o operador de adição (+):

{

*string* nome = "João";

*string* sobrenome = "Silva";

*string* nomeCompleto = nome + " " + sobrenome;

   Console.WriteLine(nomeCompleto); // Saída: João Silva

}

Nesse exemplo, as variáveis **nome** e **sobrenome** são concatenadas usando o operador de adição (**+**). O resultado da concatenação é atribuído à variável **nomeCompleto**, que contém o nome completo.

1. Usando o método Concat():

{

*string* saudacao = "Ola, ";

*string* nome = "Silva";

*string* mensagem = string.Concat(saudacao, nome);

    Console.Write(mensagem);

}

Nesse exemplo, o método Concat() da classe string é usado para concatenar as strings saudacao e nome.

1. Usando o método Format():

{

*string* saudacao = "Ola,";

*string* nome = "Silva";

*string* mensagem = string.Format("{0} {1}", saudacao, nome);

  Console.Write(mensagem);

}

Neste exemplo, o método string.Format() para combinar as strings saudacao e nome com um espaço entre elas.

A chamada string.Format("{0} {1}", saudacao, nome) insere o conteúdo da variável saudacao no marcador de posição {0} e o conteúdo da variável nome no marcador de posição {1}. O resultado é a string final "Olá, Silva", onde o espaço em branco separa a saudação do nome.

1. Usando interpolação de strings:

{

*string* nome = "Pedro";

*int* idade = 25;

*string* mensagem = $"Meu nome é {nome} e tenho {idade} anos.";

  Console.WriteLine(mensagem);

}

Nesse exemplo, a interpolação de strings é usada para combinar variáveis e texto em uma única string. As variáveis nome e idade são inseridas na string usando a sintaxe ${}.

A concatenação de strings é uma operação comum em muitos cenários de programação, como a construção de mensagens, a formatação de saídas e a geração dinâmica de conteúdo. O C# oferece várias formas flexíveis de realizar a concatenação de strings para atender às suas necessidades.

1. Objeto auxiliar (Buffer)

*StringBuilder* nomeCompletoBuffer = new *StringBuilder*();

nomeCompletoBuffer.Append(nome);

nomeCompletoBuffer.AppendFormat(" {0}", sobrenome);

*string* nomeCompleto = nomeCompletoBuffer.ToString();

Console.WriteLine(nomeCompleto);

A classe StringBuilder serve para construir o nome completo a partir das variáveis nome e sobrenome.

Nesse código, você cria um objeto StringBuilder chamado nomeCompletoBuffer. Em seguida, usa os métodos Append() e AppendFormat() para adicionar o conteúdo das variáveis nome e sobrenome ao StringBuilder. O método Append() é usado para adicionar o conteúdo da variável nome, e o método AppendFormat() é usado para adicionar o conteúdo da variável sobrenome com formatação.

Finalmente, você chama o método ToString() do StringBuilder para obter a string final resultante da concatenação dos valores. Essa string é atribuída à variável nomeCompleto.

A utilização do StringBuilder é recomendada quando você precisa realizar várias operações de concatenação ou formatação de strings, pois ele evita a criação repetida de novas strings a cada operação, melhorando o desempenho e a eficiência de memória.

Interpolação e Formatação de Composição

A interpolação e a formatação de composição são recursos no C# que permitem inserir valores em strings de maneira mais conveniente. Vamos explorar cada um deles:

1. Interpolação de strings: A interpolação de strings é uma forma mais legível e concisa de concatenar valores em strings. Ela permite inserir expressões ou variáveis diretamente dentro de uma string precedida pelo símbolo de cifrão ($) e envolvida por chaves ({}). O compilador C# substitui essas expressões pelo valor real durante a execução.

Aqui está um exemplo de como usar a interpolação de strings:

{

*string* nome = "Marcelo";

*int* idade = 53;

*string* mensagem = $"O aluno da Etec {nome} tem {idade} anos”;

  Console.WriteLine(mensagem);

}

Saída = O aluno da Etec Marcelo tem 53 anos

1. Formatação de composição: A formatação de composição é uma técnica que permite controlar como os valores são apresentados em uma string, definindo um formato específico para eles. Ela é feita usando espaços reservados dentro da string, que são substituídos pelos valores reais em tempo de execução.

Aqui está um exemplo de como usar a formatação de composição:

 {

*string* nome = "Marcelo";

*int* idade = 53;

*string* mensagem = string.Format("O aluno {0} tem {1} anos", nome, idade);

  Console.WriteLine(mensagem);

}

Saída = O aluno Marcelo tem 53 anos

Além disso, o C# oferece diversos formatos predefinidos que podem ser usados para controlar a apresentação de valores numéricos, datas, moedas, entre outros.

{

*string* nome = "Marcelo";

*double* valorConta = 5365.65;

*string* mensagem = string.Format(" {0} tem {1:C} na conta", nome, valorConta);

    Console.WriteLine(mensagem);

}

Saída = Marcelo tem R$ 5.365,65 na conta

Esses recursos tornam a manipulação de strings mais simples e legível, além de permitir um controle mais preciso sobre a formatação dos valores. Vale ressaltar que a interpolação de strings foi introduzida no C# 6.0, enquanto a formatação de composição está disponível desde versões anteriores do C#.

Alinhamentos:

|  |  |  |  |
| --- | --- | --- | --- |
| Entrada | alinhamento | Saída | Explicação |
| abc | 10 | •••••••abc | A saída terá 10 caracteres, com a entrada alinhada à direita |
| abc | -10 | abc••••••• | A saída terá 10 caracteres, com a entrada alinhada à esquerda |

Caso o valor seja maior do que o tamanho especificado, o alinhamento é ignorado.

Formatações:

|  |  |  |  |
| --- | --- | --- | --- |
| Entrada | formatação | Saída | Explicação |
| 2124.5 | C | R$ 2.124,50 | Formatação de moeda, na cultura atual do sistema operacional |
| 2124.5 | N4 | 2.124,5000 | Saída numérica formatada como número da cultura atual, com 4 casas decimais |
| 0,3982 | P1 | 39,8 % | Saída numérica multiplicada por 100, com uma casa decimal, seguida do caractere % |

Sempre que for usar a formatação e for dar um print , colocar em mente que precisa estar entre strings.

Conversão para string

Todos os objetos de C# possuem um método para conversão do seu valor em string, chamado .ToString(). Caso não haja uma representação humanamente legível adequada, será retornado o tipo do objeto.

Cada tipo pode disponibilizar parametrizações para personalizar a saída. Muitos deles (mas nem todos) aceitam a formatação de composição.

192.ToString("N4") // Retorna 192,0000 – N informa quantas casas decimais terá

Também é possível a conversão de inteiros para string usando a classe Convert, onde pode ser definida a base numérica a ser utilizada:

Convert.ToString(192, 2) // "11000000", que é a representação de 192 em binário

Convert.ToString(192, 8) // "300", que é a representação de 192 em octal

Convert.ToString(192, 16) // "c0", que é a representação de 192 em hexadecimal

Em C#, é possível converter valores de diferentes tipos para uma representação em string. Existem várias maneiras de realizar a conversão para string. Aqui estão algumas das principais abordagens:

1. Método ToString(): A maioria dos tipos em C# possui um método chamado ToString(), que retorna uma representação em string do valor. Esse método pode ser chamado diretamente no valor que você deseja converter. Aqui está um exemplo:

{

*int* idade = 25;

*string* idadeString = idade.ToString();

  Console.WriteLine(idadeString);

}

Saída = 25

1. Método Convert.ToString(): A classe Convert em C# possui um método estático chamado ToString(), que converte um valor para uma representação em string. Esse método pode ser usado para converter valores de diferentes tipos. Aqui está um exemplo:

{

*double* preco = 25.00;

*string* precoString = Convert.ToString(preco);

    Console.WriteLine(precoString);

}

Saída = 25

1. Concatenação com uma string vazia: Uma maneira simples de converter um valor para string é concatená-lo com uma string vazia (""). O compilador irá converter automaticamente o valor para string. Aqui está um exemplo:

{

*bool* isTrue = true;

*string* StringIsTrue = "" + isTrue;

    Console.WriteLine(StringIsTrue);

}

1. Método string.Format(): O método string.Format() permite formatar e compor strings usando espaços reservados. Ele também pode ser usado para converter valores para string. Aqui está um exemplo:

{

*int* numero = 25;

*string* numeroString = string.Format("{0}", numero);

   Console.WriteLine(numeroString);

}

Essas são algumas das maneiras mais comuns de realizar a conversão para string em C#. A escolha do método adequado depende do contexto e dos requisitos específicos do seu código.

Opções úteis em System.String

Resultam em uma string:

* Insert insere uma cadeia de caracteres na string atual.

{

*string* nome = "Marcelo Bertochi";

*string* nomecompleto = nome.Insert(8, "Camara ");

  Console.WriteLine(nomecompleto);

}

Saída = Marcelo Camara Bertochi

Com o Insert() podemos adicionar uma string no meio de outras strings sem afetar o valor de uma variável.

* PadLeft insere uma ou mais ocorrências de um caractere especificado no início de uma string.

O método PadLeft() é um método da classe System.String em C# que permite preencher uma string à esquerda com caracteres específicos até que ela atinja um determinado comprimento.

Aqui está um exemplo de uso do método PadLeft():

{

*string* texto = "123";

*int* comprimentoDesejado = 5;

*char* textopreenchido = '0'; //Tipo Char usa-se aspas simples

*string* textoCompleto = texto.PadLeft(comprimentoDesejado,

textopreenchido);

   Console.WriteLine(textoCompleto);

}

Neste exemplo, a string original "123" possui comprimento 3. Usando o método PadLeft(), especificamos o comprimento desejado como 5 e o caractere de preenchimento como '0'. O método PadLeft() adiciona caracteres '0' à esquerda da string original até que ela atinja o comprimento desejado. O resultado é a string "00123", onde foram adicionados dois caracteres '0' à esquerda para atingir o comprimento desejado de 5.

O método PadLeft() é útil quando você precisa formatar strings para exibição em formatos específicos, como números com dígitos significativos ou representação de datas e horas com zeros à esquerda. Ele também pode ser combinado com outros métodos para formatação de strings em C#.

PadRight insere uma ou mais ocorrências de um caractere especificado no final de uma string.

A classe System.String em C# realmente possui o método PadRight(), que é usado para preencher uma string à direita com caracteres específicos até que ela atinja um determinado comprimento. Peço desculpas pela confusão anterior.

Aqui está um exemplo de uso correto do método PadRight():

{

*string* nome = "Marcelo";

*int* ComprimentoTexto = 10;

*char* preenchimentoTexto = '\*';

   Console.WriteLine($"{nome.PadRight(ComprimentoTexto, preenchimentoTexto

)}");

}

Remove exclui uma subcadeia de caracteres da string.

Replace substitui uma subcadeia de caracteres por outra subcadeia de caracteres na string.

Substring obtém partes de uma string de acordo com sua posição

Substring(x, y) retorna os y próximos caracteres na string iniciando na posição x

Substring(z) retorna todos os caracteres da string a partir da posição z até o final

ToLower converte todos os caracteres em uma string em minúsculas.

ToUpper converte todos os caracteres em uma string em letras maiúsculas.

Trim remove todas as ocorrências de um caractere do início e do fim de uma string.

TrimEnd remove todas as ocorrências de um caractere do final de uma string.

TrimStart remove todas as ocorrências de um caractere do início de uma string.