**UNIVERSIDADE FEDERAL DE SERGIPE**

**DEPARTAMENTO DE SISTEMAS DE INFORMAÇÃO - DSI**

**LINGUAGENS FORMAIS E TRADUTORES**

Alexandre Moreira Silva

Fábio Christian Barbosa Diniz

Ícaro Gabriel de Almeida Reis

Marcos Vinicius Santos Morais

Pedro Henrique Oliveira Dantas

Victor Gabriel Vieira Santos

**LINGUAGEM DE PROGRAMAÇÃO: C#**

Itabaiana - SE, janeiro de 2024

1. **Introdução**

O C# é um moderno, inovador, software livre, plataforma cruzada linguagem de programação orientada a objeto e uma das cinco principais linguagens de programação no GitHub.

C# tem suas raízes na família de linguagens C e é instantaneamente reconhecível por programadores C, C++, Java e JavaScript. Este tour fornece uma visão geral dos principais componentes da linguagem em C# 8 e versões anteriores.

Uma das principais vantagens dela é a sua interação com o framework .NET, o que significa que os desenvolvedores podem criar uma ampla gama de aplicativos para Windows, web, dispositivos móveis e serviços em nuvem. Além disso, oferece uma sintaxe avançada que permite aos desenvolvedores criar soluções robustas e escaláveis

C# é uma linguagem de programação orientada a objetos e orientada a componentes. C# fornece construções de linguagem para dar suporte direto a esses conceitos, tornando o C# uma linguagem natural para criar e usar componentes de software. Desde a sua criação, o C# adicionou recursos para dar suporte a novas cargas de trabalho e práticas emergentes de design de software. Em sua essência, C# é uma linguagem orientada a objetos. Você define os tipos e seu comportamento.

1. **Lexemas do C#**

A estrutura da linguagem de programação C# (c-sharp) possui diversas palavras-chave que atuam como identificadores predefinidos e reservados e têm um significado especial para o compilador.

Os identificadores devem seguir as seguintes regras para serem considerados válidos:

* Devem começar com letra ou sublinhado (\_).
* Os identificadores podem conter caracteres de letra Unicode, caracteres de dígitos decimais, caracteres de conexão Unicode, caracteres de combinação Unicode ou caracteres de formatação Unicode. Para mais detalhes, [Unicode Database](https://www.unicode.org/reports/tr44/).
* Não devem conter espaços em branco.
* Não podem ser palavras-chave.

Lembrando que a linguagem é case sensitive, diferenciando letras maiúsculas das minúsculas.

* 1. **Comentários**

C# oferece suporte a duas formas diferentes de comentários. Comentários de linha única começam // e terminam no final dessa linha de código. Comentários multilinhas começam com /\* e terminam com \*/. O código a seguir mostra um exemplo de cada um desses casos:

* Comentário de linha única:
  + // Isso é uma única linha comentada
* Comentário de múltiplas linhas:
  + /\* Isso é um comentário de múltiplas linhas \*/
* Comentário de documentação:
  + /\*\* Isto é um comentário de documentação \*/
  + ///Isto é um comentário de documentação
  1. **Palavras reservadas**

A primeira tabela lista palavras-chave que são identificadores reservados em qualquer lugar de um programa C#. A segunda tabela lista palavras-chave de contexto em C#. Palavras-chave contextuais têm significado especial apenas no contexto limitado do programa e podem ser usadas como identificadores fora do contexto. Em geral, à medida que novas palavras-chave são adicionadas ao C#, elas são adicionadas como palavras-chave de contexto para evitar a quebra de programas escritos em versões anteriores.

|  |  |
| --- | --- |
| abstract # | Indica que o item que está sendo modificado tem uma implementação ausente ou incompleta. |
| as # | O as operador converterá explicitamente uma expressão para determinado tipo se o tipo de runtime dele for compatível com esse tipo |
| base # | A palavra-chave base é usada para acessar membros da classe base de dentro de uma classe derivada. |
| bool | A palavra-chave de tipo bool é um alias para o tipo de estrutura [System.Boolean](https://learn.microsoft.com/pt-br/dotnet/api/system.boolean) do .NET que representa um valor booleano, que pode ser true ou false. |
| break | A instrução break encerra a instrução de iteração mais próxima (ou seja, loop for, foreach, while ou do) ou instrução switch |
| byte # | A palavra-chave byte é usada para declarar uma variável que pode conter valores de dados de 8 bits |
| case | A palavra-chave case é usada com as instruções switch para marcar blocos de texto. |
| catch | A palavra-chave catch é usada para capturar as exceções geradas pelas instruções try. Ele deve ser usado somente após o bloco try |
| char | Palavra-chave char é usada para declarar uma variável que pode conter caracteres Unicode de 16 bits não assinados |
| checked # | As instruções checked e unchecked especificam o contexto de verificação de estouro para operações e conversões aritméticas de tipo integral. |
| class | A palavra-chave class é utilizada para declarar uma classe |
| const | Define uma constante, um valor que não pode ser alterado. |
| continue | Utilizado para pular para a próxima iteração de um loop. |
| decimal | Tipo de dados que armazena números de ponto flutuante com precisão maior do que o tipo "double". |
| delegate # | Representa um tipo seguro para delegados. |
| do | Inicia um loop "do-while". |
| double | Tipo de dados que armazena números de ponto flutuante de precisão dupla. |
| else | Utilizado em estruturas "if" para definir um bloco de código a ser executado quando a condição é falsa. |
| enum | Declaração de um tipo enumerado, uma lista nomeada de valores constantes. |
| event # | Declara um evento, permitindo que objetos notifiquem outros objetos quando ocorrem determinadas ações. |
| explicit # | Define uma conversão de tipo explícita, especificando como um tipo pode ser convertido em outro. |
| extern # | Indica que um método é implementado externamente, geralmente em código não gerenciado. |
| finally | Bloco de código que é executado após o bloco "try" em uma estrutura "try-catch", independentemente de ocorrer uma exceção ou não. |
| fixed # | Usado para criar um bloco de código que fixa um ponteiro em um local específico na memória. |
| float | Tipo de dados que armazena números de ponto flutuante de precisão simples. |
| for | Inicia um loop "for" que executa uma sequência de instruções enquanto uma condição específica é verdadeira. |
| foreach | Itera sobre os elementos de uma coleção, como arrays ou coleções genéricas. |
| goto | Transferência incondicional de controle para um rótulo específico no código. |
| if | Estrutura de controle de fluxo condicional que executa um bloco de código se uma condição especificada for verdadeira. |
| implicit # | Define uma conversão de tipo implícita, permitindo que o compilador faça a conversão automaticamente. |
| in | Utilizado para passar argumentos por referência de forma somente leitura. |
| int | Tipo de dados que armazena valores inteiros de 32 bits. |
| interface | Declaração de uma interface, um contrato que define um conjunto de métodos que uma classe deve implementar. |
| internal # | Define que um tipo ou membro é acessível apenas dentro do mesmo assembly. |
| is | Verifica se um objeto é do tipo especificado. |
| lock # | Bloqueia o acesso a um bloco de código por meio de um objeto de sincronização, evitando condições de corrida em ambientes multithread. |
| [long](https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/integral-numeric-types) | Tipo de dados que armazena valores inteiros de 64 bits. |
| namespace | Declaração que organiza o código em um escopo lógico, evitando conflitos de nomes e permitindo a criação de hierarquias. |
| new | Utilizado para criar instâncias de tipos e ocultar membros herdados em uma classe derivada. |
| object | Tipo de dados fundamental que é a base de todos os outros tipos em C#. |
| operator # | Define operadores personalizados para tipos definidos pelo usuário. |
| out | **I**ndica que um argumento é passado como uma referência de saída. |
| override | Substitui a implementação de um método, propriedade, indexador ou evento em uma classe derivada. |
| params | Permite que um método aceite um número variável de argumentos. |
| private | Define que um tipo ou membro só é acessível dentro da própria classe. |
| protected | Define que um tipo ou membro é acessível dentro da própria classe e de suas classes derivadas. |
| public | Define que um tipo ou membro é acessível de qualquer lugar no código. |
| readonly | Indica que uma variável só pode ser atribuída durante a inicialização ou no construtor. |
| ref | Indica que um argumento é passado por referência. |
| return | Retorna um valor de um método. |
| sbyte # | Tipo de dados que armazena valores inteiros de 8 bits com sinal. |
| sealed # | Impede que uma classe seja derivada por outras classes. |
| short | Tipo de dados que armazena valores inteiros de 16 bits. |
| sizeof | Retorna o tamanho em bytes de um tipo de dados. |
| stackalloc # | Aloca memória na pilha em tempo de execução. |
| static | Define que um membro pertence à classe em vez de instâncias individuais da classe. |
| string | Tipo de dados que armazena sequências de caracteres. |
| struct | Declaração de uma estrutura, um tipo de valor. |
| switch | Estrutura de controle de fluxo que direciona a execução para um dos muitos blocos de código, dependendo do valor de uma expressão. |
| this | Referência ao objeto atual dentro de uma classe. |
| throw | Gera uma exceção. |
| try | Define um bloco de código onde exceções podem ocorrer e que pode ser seguido pelos blocos "catch" e "finally". |
| typeof | Obtém um objeto Type para um tipo de dados. |
| uint | Tipo de dados que armazena valores inteiros de 32 bits sem sinal. |
| ulong | Tipo de dados que armazena valores inteiros de 64 bits sem sinal. |
| unchecked # | Desativa a verificação de estouro de aritmética durante a execução. |
| unsafe # | Permite o uso de código não seguro. |
| ushort | Tipo de dados que armazena valores inteiros de 16 bits sem sinal. |
| using # | Declaração que indica o início e o término do escopo de um objeto que implementa a interface IDisposable, garantindo a liberação de recursos. |
| virtual # | Permite que um método seja substituído em classes derivadas. |
| void | Tipo de dados que representa a ausência de tipo. |
| volatile # | Indica que um campo pode ser modificado por vários threads simultaneamente. |
| while | Inicia um loop "while" que executa uma sequência de instruções enquanto uma condição específica é verdadeira. |

* 1. **Literais reservadas**

**null:** Representa uma referência nula, isto é, que não se refere a nenhum objeto. Apenas alguns tipos podem receber null como valor.

**true/false:** Representa um valor booleano.

**default:** Utilizado para especificar o bloco de código padrão em uma instrução switch; Retornar o valor padrão de um tipo especificado.

* 1. **Operadores e Delimitadores**
     1. **Operadores aritméticos**

|  |  |  |
| --- | --- | --- |
| **Categoria** | **Definição** | **Operadores** |
| Unário | Incremento/decremento  pré-fixado | ++op | --op |
| Incremento/decremento  pós-fixado | op++ | op-- |
| Mais/menos | +op | -op |
| Binário | Multiplicação | \* |
| Divisão | / |
| Resto | % |
| Adição | + |
| Subtração | - |

* + 1. **Operadores lógicos**

|  |  |  |
| --- | --- | --- |
| **Categoria** | **Definição** | **Operadores** |
| Unário | Negação | !op |
| Binário | AND lógico | & |
| OR exclusivo lógico | ^ |
| OR lógico | | |
| AND condicional (curto-circuito) | && |
| OR condicional (curto-circuito) | || |

* + 1. **Operadores bit a bit e shift**

|  |  |  |
| --- | --- | --- |
| **Categoria** | **Definição** | **Operadores** |
| Unário | Complemento | ~op |
| Binário | Deslocamento à esquerda | << |
| Deslocamento à direita | >> |
| AND lógico | & |
| OR exclusivo lógico | ^ |
| OR lógico | | |

* + 1. **Operadores de comparação**

|  |  |  |
| --- | --- | --- |
| **Categoria** | **Definição** | **Operadores** |
| Igualdade | Igualdade | == |
| Desigualdade | != |
| Comparação | Menor que | < |
| Maior que | > |
| Menor ou igual que | <= |
| Maior ou igual que | >= |

* + 1. **Operadores de atribuição**

|  |  |  |
| --- | --- | --- |
| **Categoria** | **Definição** | **Operadores** |
| Associativo à direita | Atribuição simples | = |
| Atribuição composta (aritmético, booleano, bit a bit e união nula) | x op= y |

* + 1. **Operador condicional ternário**

É usado o conjunto de operadores (?:) para simplificar o resultado de uma expressão booleana. Esse operador é associativo à direita.

Ex.:

string classify;

int input = 5;

// Forma clássica

if (input >= 0)

{

classify = "nonnegative";

}

else

{

classify = "negative";

}

// Forma ternária

classify = (input >= 0) ? "nonnegative" : "negative";

* + 1. **Operador de expressão lambda #**

É usado o operador (=>) tanto para expressões lambda como também para expression body definition. Na primeira situação é usado para separar os parâmetros de entrada à esquerda da expressão lambda do corpo que fica à direita. Na segunda situação a sintaxe é *membro => expressão* e pode ser aplicado para definir métodos, operadores, propriedades read-only, construtores, finalizadores e acessadores de propriedades e indexadores.

Ex.:

// Lambda com dois parâmetros

int[] numbers = { 4, 7, 10 };

int product = numbers.Aggregate(1, (interim, next) => interim \* next);

Console.WriteLine(product); // output: 280

// Lambda sem parâmetros de entrada

Func<string> greet = () => "Hello, World!";

// Expression body para definir um método

public override string ToString() => $"{fname} {lname}".Trim();

* 1. **Literais String**

Literais strings são representações de código que representam um valor de um objeto string. Existem várias maneiras de se representar um objeto String, em C#.

**EX:** string unicode = "Café é delicioso. \u2605"; // Representação Unicode

Os literais string em C# podem incluir caracteres de escape (como \n para nova linha ou \t para tabulação) e podem ser concatenados usando o operador +. Além disso, a classe System.String em C# oferece muitos métodos e propriedades úteis para manipulação de strings.

* 1. **Literais numéricos**

Em C#, os literais numéricos são representações diretas de valores numéricos. Eles podem ser usados para inicializar variáveis numéricas nos diferentes tipos disponíveis na linguagem.

* + 1. **Literais integrais**
       1. **Literal decimal**

O literal decimal é representado diretamente como um número inteiro no sistema decimal.

**Ex:** int decimalLiteral = 42; // Literal decimal

* + - 1. **Literal hexadecimal**

O literal hexadecimal é representado com um prefixo "0x" indicando que o número está em hexadecimal.

**Ex:** int hexadecimalLiteral = 0x2A; // Literal hexadecimal (prefixo 0x)

* + - 1. **Literal binário**

O literal binário é representado com um prefixo "0b" indicando que o número está em binário.

**Ex:** int binarioLiteral = 0b101; // Literal binario (prefixo 0b)

* + 1. **Literais de ponto flutuante**

Em C# são utilizados para representar números reais (números com casas decimais). As literais de ponto flutuante são, por padrão, do tipo primitivo double.

**Ex:** float flutuanteLiteral = 3.14f;// Literal de ponto flutuante do tipo float

double duploLiteral = 3.14;// Literal de ponto flutuante do tipo double

* + 1. **Literais booleanas**

Em C# representam os dois valores possíveis de uma variável booleana: **true** (verdadeiro) ou **false** (falso).

**Ex:** boolean a = true; // Ok!

a = false; // Ok!

a = 0; // Erro de compilação

a = 1; // Erro de compilação

* + 1. **Literais Char**

Em C# representam um único caractere e são escritos entre aspas simples (**'**).

**Ex:** char caractereA = 'A'; // Caractere literal 'A'

char caractereNumerico = '7'; // Caractere literal '7'

char caractereEspecial = '$'; // Caractere literal '$'

1. **Delimitadores**

Utiliza delimitadores semelhantes à linguagem C. O (;) indica o fim de uma instrução. A (,) é usada para separar parâmetros de funções. Ainda, os parênteses são usados para agrupar expressões e as chaves dividem os blocos de comandos da linguagem.

1. **Referências**

C# documentation, 2024. Disponível em: <https://learn.microsoft.com/en-us/dotnet/csharp/>