**Linguagem de Programação: C#**

Departamento de Sistemas de Informação

**Universidade Federal de Sergipe (UFS) – Itabaiana, SE – Brasil**

Carlos Henrique Lima de Jesus

Charles Dayan da Conceição Costa

Jorge Matheus dos Santos

**1. Introdução**

O C# (pronuncia-se "C Sharp") é uma linguagem de programação moderna, orientada a objeto e fortemente tipada. O C# permite que os desenvolvedores criem muitos tipos de aplicativos seguros e robustos que são executados no .NET.

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

Vários recursos do C# ajudam a criar aplicativos robustos e duráveis. A [***coleta de lixo***](https://learn.microsoft.com/pt-br/dotnet/standard/garbage-collection/) recupera automaticamente a memória ocupada por objetos não utilizados inacessíveis. [***Tipos anuláveis***](https://learn.microsoft.com/pt-br/dotnet/csharp/nullable-references) são protegidos contra variáveis que não se referem a objetos alocados. O [***tratamento de exceções***](https://learn.microsoft.com/pt-br/dotnet/csharp/fundamentals/exceptions/) fornece uma abordagem estruturada e extensível para detecção e recuperação de erros. [***As expressões Lambda dão suporte a técnicas***](https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/operators/lambda-expressions) de programação funcional. [***Consulta Integrada à Linguagem (LINQ)***](https://learn.microsoft.com/pt-br/dotnet/csharp/linq/) a sintaxe cria um padrão comum para trabalhar com dados de qualquer fonte. O suporte à linguagem para [***operações assíncronas fornece sintaxe***](https://learn.microsoft.com/pt-br/dotnet/csharp/programming-guide/concepts/async/) para a criação de sistemas distribuídos. C# tem um [***sistema de tipo unificado***](https://learn.microsoft.com/pt-br/dotnet/csharp/fundamentals/types/). Todos os tipos do C#, incluindo tipos primitivos, como int e double, herdam de um único tipo de object raiz. Todos os tipos compartilham um conjunto de operações comuns. Valores de qualquer tipo podem ser armazenados, transportados e operados de maneira consistente. Além disso, o C# dá suporte a [**tipos de referência**](https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/builtin-types/reference-types) e [**tipos de valor**](https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/builtin-types/value-types) definidos pelo usuário. O C# permite a alocação dinâmica de objetos e o armazenamento em linha de estruturas leves. O C# dá suporte a métodos e tipos genéricos, que fornecem maior segurança e desempenho do tipo. O C# fornece iteradores, que permitem que os implementadores de classes de coleção definem comportamentos personalizados para o código do cliente.

O C# enfatiza o controle de versão para garantir que programas e bibliotecas possam evoluir ao longo do tempo de maneira compatível. Aspectos do design do C# que foram diretamente influenciados pelas considerações de controle de versão incluem os modificadores separados virtual e override, as regras de resolução de sobrecarga de método e suporte para declarações explícitas de membro de interface.

**2. Lexemas do C#**

Utilizamos, em C#, as seguintes regras para criação do identificador:

1. não pode ser uma palavra-reservada (palavra-chave);

2. não pode ser true nem false - literais que representam os tipos lógicos (booleanos); 3. não pode ser null - literal que representa o tipo nulo;

4. não pode conter espaços em brancos ou outros caracteres de formatação; 5. deve ser a combinação de uma ou mais letras e dígitos UNICODE-16. Por exemplo, no alfabeto latino, teríamos:

• letras de *A* a *Z* (de *\u0041* a *\u005a*);

• letras de *a* a *z* (de *\u0061* a *\u007a*);

• sublinha *\_* (*\u005f*);

• cifrão *$* (*\u0024*);

• dígitos de *0* a *9* (de *\u0030* a *\u0039*).

Observação 01: caracteres compostos (acentuados) não são interpretados igualmente aos não compostos (não acentuados). Por exemplo, *História* e *Historia* não são o mesmo identificador. Observação 02: letras maiúsculas e minúsculas diferenciam os identificadores, ou seja, *a* é um identificador diferente de *A*, *História* é diferente de *história*, etc.

**2.1 Comentários**

Os comentários em C# são declarações que não são executadas pelo compilador ou interpretador. Os comentários podem ser usados para prover informações ou explicações sobre as variáveis, métodos, classes ou alguma declaração. Existem x tipos de comentários em C#:

* 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

**2.2 Palavras reservadas**

Em programação, *palavras-chave*, ou *palavras reservadas*, são as palavras que não podem ser usadas como identificadores. Em outras palavras, não podem ser usadas como nome de variáveis, nome de classes, etc. Estas palavras são assim definidas ou porque já têm uso na sintaxe da linguagem ou porque serão usadas em alguns momentos, seja para manter compatibilidade com versões anteriores ou mesmo com outras linguagens. No caso do C# temos as seguintes *palavras reservadas*:

* **abstract:** o modificador abstract indica que o item que está sendo modificado tem uma implementação ausente ou incompleta. O modificador abstrato pode ser usado com classes, métodos, propriedades, indexadores e eventos.
* **as:** o operador as converte explicitamente o resultado de uma expressão para uma determinada referência ou tipo de valor anulável. Se a conversão não for possível, o operador as retornará null. Ao contrário de uma expressão de conversão, o operador as nunca gera uma exceção.
* **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. A instrução break transfere o controle para a instrução que segue a instrução encerrada, se houver.
* **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:** a palavra-chave char é usada para declarar uma variável que pode conter caracteres Unicode de 16 bits não assinados.
* **checked/unchecked:** as instruções checked e unchecked especificam o contexto de verificação de estouro para operações e conversões aritméticas de tipo integral. Quando ocorre um estouro aritmético inteiro, o contexto de verificação de estouro define o que acontece.
* **class:** a palavra-chave class é utilizada para declarar uma classe
* **const:** use a palavra-chave const para declarar um campo constante ou um local constante. Campos e locais constantes não são variáveis e não podem ser modificados.
* **continue:** utilizada para continuar o loop. Ele continua o fluxo atual do programa e ignora o código restante na condição especificada.
* **decimal:** o tipo decimal é apropriado quando o grau de precisão necessário é determinado pelo número de dígitos à direita do ponto decimal.
* **delegate:** um delegate é um tipo de referência que pode ser usado para encapsular um método nomeado ou anônimo.
* **do:** a palavra-chave do é usada na instrução de controle para declarar um loop. Ela pode iterar uma parte do programa várias vezes.
* **double:** usada para declarar uma variável que pode conter números de ponto flutuante de 64 bits.
* **else:** utilizada para indicar as ramificações alternativas em uma instrução if.
* **enum:** utilizada para definir um conjunto fixo de constantes. Os construtores enum são sempre privados ou padrão.
* **event:** a palavra-chave event é usada para declarar um evento em uma classe publicadora.
* **explicit/implicit:** um tipo definido pelo usuário pode definir uma conversão implícita ou explícita personalizada de outro tipo ou para outro. Conversões implícitas não exigem a invocação de sintaxe especial e podem ocorrer em várias situações, por exemplo, em invocações de atribuições e métodos.
* **finally:** Usando um bloco finally, você pode limpar todos os recursos alocados em um bloco try e pode executar código mesmo se uma exceção ocorrer no bloco try. Normalmente, as instruções de um bloco finally são executadas quando o controle deixa uma instrução try. A transferência de controle pode ocorrer como resultado da execução normal, da execução de uma instrução break, continue, goto ou return, ou da propagação de uma exceção para fora da instrução try.
* **fixed:** a instrução fixed impede que o coletor de lixo realoque uma variável que pode ser movida e declara um ponteiro para essa variável.
* **float:** usada para declarar uma variável que pode conter um número de ponto flutuante de 32 bits.
* **for:** usada para iniciar um loop for. Tem o objetivo de executar um conjunto de instruções / funções repetidamente quando algumas condições se tornam verdadeiras. Se o número de iterações for fixo, é recomendável usar o loop.
* **foreach:** a instrução foreach executa uma instrução ou um bloco de instruções para cada elemento em uma instância do tipo que implementa a interface System.Collections.IEnumerable ou System.Collections.Generic.IEnumerable<T>.
* **goto:** A instrução goto transfere o controle para uma instrução marcada por um rótulo, você pode usar a instrução goto para sair de um loop aninhado. Você também pode usar a instrução goto na instrução switch a fim de transferir o controle para uma seção switch com um rótulo case constante.
* **if:** testa a condição. Executa o bloco if se a condição for verdadeira.
* **in:** a palavra-chave in é usada nos seguintes contextos: parâmetros de tipo genérico em interfaces e delegados genéricos. Como um modificador de parâmetro, que permite passar um argumento para um método por referência, não por valor. Instruções foreach. Cláusulas from em expressões de consulta LINQ. Cláusulas de junção em expressões de consulta LINQ.
* **int:** usada para declarar uma variável que pode conter um número inteiro assinado de 32 bits.
* **interface:** a palavra-chave interface é usada para declarar uma interface. Pode ter apenas métodos abstratos.
* **internal:** a palavra-chave internal é um modificador de acesso para tipos e membros de tipo.
* **is:** O operador is verifica se o resultado de expressão é compatível com um determinado tipo. Você também pode usar o is operador para corresponder uma expressão com um padrão.
* **lock: a** instrução lock obtém o bloqueio de exclusão mútua para um determinado objeto, executa um bloco de instruções e, em seguida, libera o bloqueio.
* **long:** a palavra-chave long é usada para declarar uma variável que pode conter um número inteiro de 64 bits.
* **namespace:** a palavra-chave namespace é usada para declarar um escopo que contém um conjunto de objetos relacionados.
* **new:** o operador new cria uma nova instância de um tipo.
* **object:** o tipo object é um alias de System.Object no .NET. No sistema de tipos unificado do C#, todos os tipos, predefinidos e definidos pelo usuário, tipos de referência e tipos de valor, herdam direta ou indiretamente de System.Object. Você pode atribuir valores de qualquer tipo a variáveis do tipo object.
* **operator:** um tipo definido pelo usuário pode sobrecarregar um operador C# predefinido. Ou seja, um tipo pode fornecer a implementação personalizada de uma operação caso um ou ambos os operandos sejam do mesmo tipo.
* **out:** você pode usar a palavra-chave out em dois contextos: Como um modificador de parâmetro, que permite passar um argumento para um método por referência, não por valor.
* **override:** o modificador override é necessário para estender ou modificar a implementação abstrata ou virtual de um método, propriedade, indexador ou evento herdado.
* **params:** usando a palavra-chave params, você pode especificar um parâmetro do método que aceita um número variável de argumentos. O tipo de parâmetro deve ser uma matriz unidimensional.
* **private:** modificador de acesso. É usado para indicar que um método ou variável pode ser acessado apenas na classe em que é declarado.
* **protected:** modificador de acesso. Pode ser acessível dentro e fora do pacote, mas apenas por herança. Não pode ser aplicado na classe.
* **public:** a palavra-chave public é um modificador de acesso para tipos e membros de tipo. Acesso público é o nível de acesso mais permissivo.
* **readonly:** A palavra-chave readonly é um modificador que pode ser usado em quatro contextos:Em uma declaração de campo, readonly indica que a atribuição ao campo só pode ocorrer como parte da declaração ou em um construtor na mesma classe. Um campo readonly pode ser atribuído e reatribuído várias vezes na declaração de campo e no construtor.
* **ref:** a palavra-chave ref indica que uma variável é uma referência ou um alias para outro objeto.
* **return:** usada para retornar um método quando sua execução estiver concluída.
* **sbyte:** o SByte tipo de valor representa inteiros com valores que variam de 128 negativos a 127 positivos.
* **sealed:** quando aplicado a uma classe, o modificador sealed impede que outras classes herdem dela.
* **short:** a palavra-chave short representa um inteiro com sinal de 16 bits.
* **sizeof:** o operador sizeof retorna o número de bytes ocupados por uma variável de um determinado tipo.
* **stackalloc:** a expressão stackalloc aloca um bloco de memória na pilha. Um bloco de memória alocado na pilha criado durante a execução do método é descartado automaticamente quando esse método é retornado.
* **static:** o modificador static serve para declarar um membro estático que pertença ao próprio tipo, em vez de um objeto específico.
* **string: o** tipo string representa uma sequência de zero ou mais caracteres Unicode.
* **struct:** um tipo de struct é um tipo de valor que pode encapsular dados e funcionalidades relacionadas.
* **switch:** a palavra-chave switch contém uma instrução switch que executa o código com base no valor do teste. A instrução switch testa a igualdade de uma variável em relação a vários valores.
* **this:** a palavra-chave this refere-se à instância atual da classe e também é usada como um modificador do primeiro parâmetro de um método de extensão.
* **throw:** a palavra-chave throw é usada para lançar explicitamente uma exceção. A palavra-chave throw é usada principalmente para lançar exceções personalizadas. É seguido por uma instância.
* **try:** utilizada para iniciar um bloco de código que será testado para exceções. O bloco try deve ser seguido pelo bloco catch ou finally.
* **typeof:** o operador typeof obtém a instância System.Type para um tipo. O argumento do operador typeof deve ser o nome de um tipo ou um parâmetro de tipo.
* **uint:** representa um inteiro sem sinal de 32 bits.
* **ulong:** o tipo de valor UInt64 representa inteiros não assinados com valores variando de 0 a 18446744073709551615.
* **unsafe:** a palavra-chave unsafe denota um contexto inseguro, que é necessário para qualquer operação que envolva ponteiros.
* **ushort:** representa um inteiro sem sinal de 16 bits.
* **using:** a palavra-chave using tem dois usos principais: A instrução using define um escopo no final do qual um objeto será descartado. A diretiva using cria um alias para um namespace ou importa tipos definidos em outros namespaces.
* **virtual:** a palavra-chave virtual é usada para modificar uma declaração de método, propriedade, indexador ou evento e permitir que ela seja substituída em uma classe derivada.
* **void:** a palavra-chave void é usada para especificar que um método não possui um valor de retorno.
* **volatile:** a palavra-chave volatile indica que um campo pode ser modificado por vários threads que estão em execução ao mesmo tempo.
* **while:** a instrução while executa uma instrução ou um bloco de instruções enquanto uma expressão booleana especificada é avaliada como true.

**2.2.1 Palavras-chave contextuais**

* **add:** a palavra-chave contextual add é usada para definir um acessador de evento personalizado que é invocado quando o código cliente assina seu evento.
* **and:** o operador lógico and (e) é um Operador Lógico que corresponde a uma das três operações lógicas envolvidas na função de pesquisa.
* **alias:** a palavra-chave alias é um apelido que podemos dar a um namespace.
* **ascending:** a palavra-chave contextual ascending é usada na cláusula orderby em expressões de consulta para especificar que a ordem de classificação é do menor para o maior.
* **args:** instruções de nível superior podem fazer referência à variável args para acessar quaisquer argumentos de linha de comando que foram inseridos.
* **async:** o modificador async serve para especificar que um método, uma expressão lambda ou um método anônimo é assíncrono.
* **await:** o operador await suspende a avaliação do método async delimitador enquanto a operação assíncrona representada por seu operando não é concluída.
* **by:** a palavra-chave contextual by é usada na cláusula group de uma expressão de consulta para especificar como os itens retornados devem ser agrupados.
* **descending:** a palavra-chave contextual descending é usada na cláusula orderby em expressões de consulta para especificar que a ordem de classificação é do maior para o menor.
* **dynamic:** o tipo dynamic indica que o uso de variável e as referências aos seus membros ignoram a verificação de tipo de tempo de compilação.
* **equals:** a palavra-chave contextual equals é usada uma cláusula join em uma expressão de consulta a fim de comparar os elementos de duas sequências.
* **from:** uma expressão de consulta deve começar com uma cláusula from.
* **get:** a palavra-chave get define um método do acessador em uma propriedade ou um indexador que retorna o valor da propriedade ou o elemento do indexador.
* **global: o** alias global, que é o alias do namespace global. O namespace global é o namespace que contém namespaces e tipos que não são declarados dentro de um namespace nomeado.
* **group:** a cláusula group retorna uma sequência de objetos IGrouping<TKey,TElement> que contêm zero ou mais itens que correspondem ao valor de chave do grupo.
* **init:** a palavra-chave init define um método de acessador em uma propriedade ou um indexador.
* **into: a** palavra-chave contextual into pode ser usada para criar um identificador temporário para armazenar os resultados de uma cláusula group, join ou select em um novo identificador**.**
* **join: a** cláusula join é útil para associar elementos de sequências de origem diferentes que não têm nenhuma relação direta no modelo de objeto.
* **let:** em uma expressão de consulta, às vezes é útil armazenar o resultado de uma subexpressão para usá-lo em cláusulas subsequentes. É possível fazer isso com a palavra-chave let, que cria uma nova variável de intervalo e a inicializa com o resultado da expressão fornecida.
* **managed/unmanaged (convenção de chamada de ponteiro em função):** você pode especificar a convenção de chamada para delegate\* usando as palavras-chave managed e unmanaged.
* **nameof: u**ma expressão nameof produz o nome de uma variável, tipo ou membro como a constante de cadeia de caracteres. A expressão nameof é avaliada em tempo de compilação e não tem efeitos em tempo de execução. Quando o operando é um tipo ou um namespace, o nome produzido não é totalmente qualificado.
* **nint:** a palavra-chave nint é utilizada para definir inteiros de tamanho nativo.
* **not:** o operador lógico not corresponde a uma expressão quando o padrão negado não corresponde à expressão.
* **notnull:** a palavra-chave notnull é utilizada para especificar que o argumento de tipo deve ser um tipo de valor não anulável ou um tipo de referência não anulável.
* **nuint:** representa um inteiro sem sinal em que a largura do bit é igual a um ponteiro.
* **on:** a palavra-chave contextual on é usada na cláusula join de uma expressão de consulta a fim de especificar a condição de união.
* **or:** a palavra-chave or corresponde a uma expressão quando um dos padrões corresponde à expressão,
* **orderby:** a cláusula orderby faz com que a sequência ou subsequência (grupo) retornada seja classificada em ordem crescente ou decrescente.
* **partial (tipo):** as definições de tipo partial permitem que a definição de uma classe, estrutura, interface ou registro seja dividida em vários arquivos.
* **partial (método): u**m método partial tem sua assinatura definida em uma parte de um tipo partial e sua implementação definida em outra parte do tipo.
* **record:** você usa a palavra-chave record para definir um tipo de referência que fornece funcionalidade interna para encapsular dados.
* **remove: a** palavra-chave contextual remove é usada para definir um acessador de eventos personalizado invocado quando o código cliente cancela a assinatura do seu evento.
* **select: e**m uma expressão de consulta, a cláusula select especifica o tipo de valores que serão produzidos quando a consulta é executada.
* **set: a** palavra-chave set define um método acessador em uma propriedade ou indexador que atribui um valor ao elemento da propriedade ou do elemento.
* **unmanaged (restrição de tipo genérico):** você pode usar a unmanaged restrição para especificar que o parâmetro de tipo deve ser um tipo não anulável não gerenciado. A restrição unmanaged permite que você escreva rotinas reutilizáveis para trabalhar com tipos que podem ser manipulados como blocos de memória.
* **value:** apalavra-chave contextual value é usada no acessador set em declarações de property e indexer. É semelhante a um parâmetro de entrada de um método.
* **var:** a palavra-chave var permite que o resultado seja uma coleção de tipos anônimos e o nome desse tipo não é acessível, exceto para o próprio compilador.
* **when:** a palavra-chave contextual when funciona para especificar uma condição de filtro nos seguintes contextos.
* **where:** a cláusula where em uma definição genérica especifica restrições sobre os tipos que são usados como argumentos para parâmetros de tipo em um tipo genérico, método, delegado ou função local. Restrições podem especificar interfaces, classes base ou exigir que um tipo genérico seja uma referência, valor ou tipo não gerenciado.
* **with: u**ma expressão with permite uma "mutação não destrutiva", projetada para produzir uma cópia da expressão do destinatário com modificações em atribuições no member\_initializer\_list.
* **yield:** a instrução yield é utilizada em um iterador para fornecer o próximo valor de uma sequência ao iterar a sequência.

**2.3 Literais reservados**

* **null**: define que uma variável não está apontando para nada.
* **false**: valor lógico que indica falso.
* **true**: valor lógico que indica verdadeiro.
* **default:** utilizada para especificar o bloco de código padrão em uma instrução switch.

**2.4 Operadores e delimitadores**

**Operador** em C# é um símbolo que é usado para executar operações. Por exemplo: +, -, \*, / etc.

Existem muitos tipos de operadores em C# que são fornecidos abaixo:

• Operador Unário,

• Operador aritmético,

• Operador shift (deslocamento),

• Operador relacional,

• Operador bit a bit (bitwise),

• Operador lógico,

• Operador ternário e

• Operador de atribuição.

Precedência de operadores em C#

| **Tipo de**  **Operador** | **Categoria** | **Precedência** |
| --- | --- | --- |
| Unário | sufixos | *expr*++ *expr*-- |
| prefixo | ++*expr* --*expr* |
| Aritmética | multiplicativo | \* |
| divisivo | / |
| sobra | % |
| aditivo | + |
| subtrativo | - |
| Shift | mudança | << >> >>> |
| Relacional | comparação | < > <= >= |
| igualdade | == != |
| Bit a bit | bit a bit AND | & |
| bit a bit OR  exclusivo | ^ |
| bit a bit  inclusive OR | | |
| complemento bit a bit | ~ |
| Lógico | AND lógico | && |
| OR lógico | || |
| Ternário | ternário | ? : |
| Atribuição | atribuição | = += -= \*= /= %= &= ^= |=  <<= >>= >>>= |

**2.5 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* nome = “José”; // Atribui "José" a variável nome *Console.WriteLine*(nome); // Imprime “José”

**2.6 Literais Numéricos**

**2.6.1 Literais Integer**

É possível representar números do tipo integer de três formas em C#: decimal (base 10), octal (base 8) e hexadecimal (base 16).

**2.6.1.1 Literal Decimal**

Os decimais (base 10) em C#, são representados apenas com números, sem prefixo ou sufixo. Exemplo: *int* idade = 62; // 62 é uma literal decimal

**2.6.1.2 Literal Octal**

Octal, em C#, utiliza números entre 0 e 7. Para representar os números em octais, deve colocar um “0” na frente de qualquer inteiro. Ex.: *int* x = 06; // 06 é uma literal octal

**2.6.1.3 Literal Hexadecimal**

Hexadecimais, ou hex, são números construídos com 16 símbolos distintos**.** sendo que do 10 ao 15 são representados pelas letras do alfabeto: a, b, c, d, e, f. Respectivamente, ficando desta forma: 0 1 2 3 4 5 6 7 8 9 a b c d e f.

Um detalhe importante a ser acrescentado, é que C# não é caseSensitive quanto ao reconhecimento das letras em hex. tanto faz você utilizar *123Cafe* quando *123CAFE*. Todos representarão 12312101514.

**2.6.4 Literais de ponto flutuante**

Literais de ponto flutuante são representados por números, sendo eles decimais, e outros números representando a fração. As literais de ponto flutuante são, por padrão, do tipo primitivo double. Ex.: *float* f = 3.14f;  *double* d = 1,50d; *decimal* de = 2.80m;

**2.6.5 Literais Booleanas**

Literais booleanas são representações de código para os valores booleanos. Sendo assim, seus valores só poderão ser true ou false.

**Ex.:**

boolean a = true; // Ok!

a = false; // Ok!

a = 0; // Erro de compilação

a = 1; // Erro de compilação

**2.6.6 Literais Char:**

São representadas por um único caractere. Por exemplo: *char* x = ‘s’. Perceba que não é utilizado duplas aspas neste caso. Mas sim, aspas simples.

**Referências:**

C# Documentation, 2023. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/csharp/>