

Programação & Algoritmos

Material de Aula

Prof. Enildo

E-mail: professor.enildo@hotmail.com

Sumário

PROGRAMAÇÃO & ALGORITMO	3
INTRODUÇÃO.....	3
FERRAMENTAS UTILIZADAS	3
INTRODUÇÃO AO CONCEITO GERAL DE LÓGICA DE PROGRAMAÇÃO	3
LÓGICA.....	3
SEQUÊNCIAS LÓGICAS	5
ALGORITMO.....	7
LÓGICA DE PROGRAMAÇÃO 'PRIMÁRIA' vs. 'NÃO PRIMÁRIA'	8
EXERCÍCIOS DE LOGICA DESCRITIVA	9
FLUXOGRAMA.....	10
OS SÍMBOLOS UTILIZADOS NO FLUXOGRAMA.....	11
CONDICIONAIS SIMPLES	14
CONDICIONAIS DE MÚLTIPLA ESCOLHA	15
ESTRUTURA DE REPETIÇÃO.....	16
UTILIZANDO O LUCIDCHART	17
EXPRESSÕES ARITMÉTICAS	21
OPERADORES ARITMÉTICOS	21
EXPRESSÕES LÓGICAS.....	22
OPERADORES LÓGICOS	22
OPERADORES RELACIONAIS.....	23
PORTUGUÊS ESTRUTURADO (PSEUDOCÓDIGO)	23
TIPOS DE DADOS	23
VARIÁVEIS	24
CONSTANTES	28
SINTAXE.....	28
LINGUAGEM DE PROGRAMAÇÃO C SHARP “C#”	29
CARACTERÍSTICAS-CHAVE:	29
VANTAGENS DE USAR C#:	29
TIPOS DE DADOS EM C#	30
MICROSOFT VISUAL STUDIO.....	34
CRIANDO UM PROGRAMA DE CONSOLE C# NO VISUAL STUDIO 2022	35
ESTRUTURA PADRÃO EM C# CONSOLE	40
NAMESPACES.....	40
CHAVES {}	40
TIPOS DE ACESSO	41
ASSEMBLY	42
CLASSE	43
CLASSE PROGRAM	44

MÉTODO	44
MÉTODO MAIN()	47
CLASSE CONSOLE.....	48
VARIÁVEIS, VETORES E MATRIZES EM C#	50
VARIÁVEIS	50
VETORES	50
MATRIZES	51
TIPOS DE OPERADORES EM C#.....	52
OPERADORES ARITMÉTICOS	52
OPERADORES DE COMPARAÇÃO	52
OPERADORES LÓGICOS	53
OPERADORES DE ATRIBUIÇÃO	54
EXEMPLOS:.....	54
OPERADORES BIT A BIT	55
EXEMPLOS:.....	55
OPERADORES CONDICIONAIS	56
OPERADOR DE COALESCÊNCIA NULO.....	57
MÉTODO CONVERT EM C#.....	58
MÉTODO PARSE	59
CONDICIONAIS	60
CONDICIONAL IF.....	60
CONDICIONAL IF-ELSE	61
CONDICIONAL SWITCH.....	61
CONDICIONAL TERNÁRIO (?).....	61
LAÇOS DE REPETIÇÃO EM C#	62
LAÇO FOR.....	62
LAÇO WHILE	64
LAÇO DO-WHILE	64
LAÇO FOREACH.....	66
.NET	68
CONCEITOS: RAD, C#, UML E IDE.....	68
PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)	68
OS PILARES DA POO	69
WINDOWS FORMS	70
CRIANDO UM APLICATIVO C# WINDOWS FORMS.	70
ADICIONANDO CONTROLES AO FORMULÁRIO.....	74

Programação & Algoritmo

Introdução

Neste material iremos ver o processo de programação estruturada e os conceitos fundamentais de lógica aplicados na elaboração de algoritmos como forma de solução de problemas. Os fundamentos das linguagens de programação, valores e variáveis, atribuição, comandos condicionais, comandos de iteração, comandos de seleção, funções e vetores. Demonstrar a resolução de problemas e o projeto de algoritmos a partir da análise do problema, desenvolvimento de estratégias de solução, representação, simulação e documentação. Ao longo do material será disponibilizados links, referencias, como material complementar e apoio aos estudos aqui abordados.

Ferramentas utilizadas

Lucidchat (“On-line” <http://lucidchart.com/>)

Visual Studio 2022

Introdução ao conceito geral de lógica de programação

Lógica de Programação é a técnica de desenvolver **algoritmos**, (sequências **lógicas**), para atingir determinados objetivos dentro de certas regras, baseadas na Lógica matemática e em outras teorias básicas da Ciência da Computação e que depois são adaptados para a Linguagem de Programação utilizada pelo programador para construir seu software.

Lógica

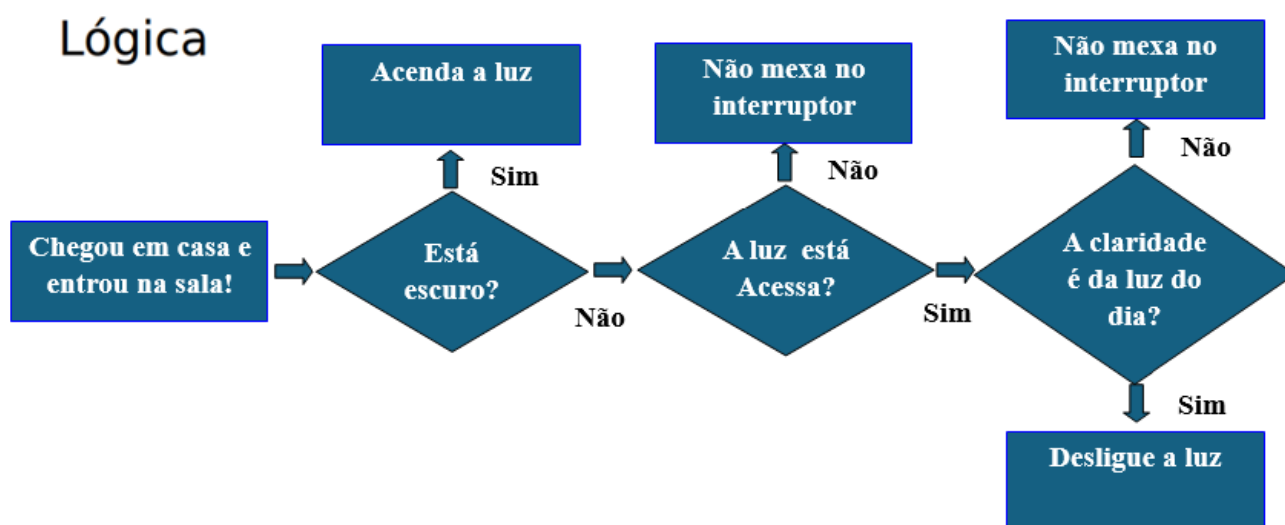
A lógica é o uso do raciocínio na busca da melhor solução para um problema ou situação.

Lógica é uma área da Filosofia dedicada à reflexão sobre formas de raciocínio (que podem ser diferentes).

A lógica define o encadeamento de ações mais coerentes para chegar a um objetivo.

Exemplos:

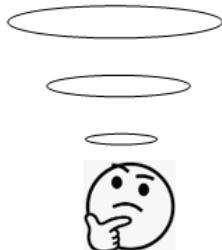
- matemática é baseada em princípios lógicos;
- o computador é uma invenção lógica — sequência de instruções/comandos lógicos;
- programas de computador — utiliza a lógica como princípio mais importante em seu desenvolvimento — todo aplicativo deve ser desenvolvido de forma racional, com sequências lógicas, coerentes e eficazes de instruções.



Vamos testar sua lógica:

Como medir exatamente 4 litros de água usando só esses 2 potes?

Os potes estão vazios! Podemos usar a torneira para enchê-los!



Solução:

1. Inicialmente os baldes de 5 e 3 litros estão vazios.
2. Encho o balde de 3 litros.
3. Verto os 3 litros no balde de 5 litros (ficam 2 litros vazios)
4. Encho o balde de 3 litros.
5. Completo o balde de 5 litros com a água do de 3 litros, onde fica 1 litro.
6. Despejo a água do balde de 5 litros.
7. Verto o 1 litro que está no balde de 3 litros no balde de 5 litros.
8. Encho o balde de 3 litros.
9. Verto os 3 litros deste balde no de 5 litros, onde já está 1 litro, ficando agora 4 litros.

Sequências Lógicas

Sequências lógicas incluem as instruções (o que deve ser feito, uma instrução equivale a um comando) e a ordem em que as instruções devem ser realizadas (quando). O objetivo da sequência lógica é encontrar um resultado ou uma solução para um problema.

- **Sequências lógicas:** são etapas a serem realizadas para atingir um objetivo, são os passos executados para conseguir solucionar um problema.
- **Instruções:** regras para situações específicas, são as ações da sequência lógica: informações, ordens ou comando que indicam o que deve ser feito. Cada ordem de uma sequência lógica é uma instrução (observe que uma ordem isolada não configura um processo completo).
- **Programa:** sequência lógica de instruções, com um objetivo final a ser atingido. Um programa é uma sequência lógica de instruções organizadas para manipular informações inseridas pelo usuário.
- **Programas de computadores** — sequências lógicas com instruções bem definidas, permitem interatividade (inserção de informações e realização de tarefas).
- **Lógica de Programação:** cumpre o papel de desenvolver a habilidade de criar sequências lógicas de instruções, padronizadas e registradas. Permite comunicar a terceiros o raciocínio lógico empregado na resolução de problemas, propicia pensamento estruturado. ¹

Exemplo:

Problema a ser resolvido

- Início
- Passo 1
- Passo 2
- Passo 3
- Passo 4
- Passo n
- Fim

¹ <https://caderno.medium.com/l%C3%B3gica-de-programa%C3%A7%C3%A3o-3d36145c0d8c>

Exemplo de uma sequência lógica para fritar um ovo.

Passo 1 – Pegar o ovo.

Passo 2 – Pegar a frigideira.

Passo 3 – Pegar óleo ou manteiga.

Passo 4 – Acender o fogo.

Passo 5 – Colocar óleo ou manteiga na frigideira.

Passo 6 – Quebrar o ovo.

Passo 7 – Por o conteúdo do ovo na frigideira.

Passo 8 – Mexer o ovo.

Exemplo de uma sequência lógica para trocar um pneu de um carro.

Passo 1 - Pare o carro em local seguro (Inicialmente, pare o seu carro em um local seguro e estável, sem riscos de deslizar. Ligue o pisca-alerta e afaste-se do tráfego).

Passo 2 - Prepare o veículo

(Puxe o freio de mão, desligue o carro e engate a marcha na primeira ou na ré. Em caso de carros de câmbio automático, posicione no modo "estacionado").

Passo 3 - Pegue o macaco e o pneu estepe (Somente depois dos dois pontos acima, desça do carro e busque o macaco, a chave de roda e o pneu estepe. Posicione o macaco sob a lataria do carro, ao lado do pneu a ser substituído. Encaixe o macaco na canaleta, uma parte metálica destinada para receber este contato. Se não identificar a canaleta, pode buscar a informação no manual do seu veículo).

Passo 4 - Apoie o carro com o macaco (depois do encaixe correto do macaco, levante-o até que o carro pareça firmemente apoiado, mas sem ainda levantá-lo do chão).

Passo 5 - Retire a calota e afrouxe os parafusos (com o pneu ainda em contato com o chão, gire a chave de roda em sentido anti-horário e afrouxe os parafusos, mas sem removê-los completamente. Ao deixar o pneu ainda no solo, será mais fácil afrouxar os parafusos e retirar a calota, pois só os parafusos irão girar, e não toda a roda).

Passo 6 - Levante um pouco mais o pneu (agora sim, com o macaco, levante mais o carro até que o pneu fique totalmente suspenso. Certifique-se de que ele esteja estável).

Passo 7 - Remova os parafusos e o pneu (continue a girar os parafusos, até que estejam soltos o suficiente para serem retirados. Em seguida, remova o pneu e coloque-o sob o carro).

Passo 8 - Posicione o pneu estepe no eixo (é hora de, enfim, encaixar o estepe no eixo. Use as mãos. Deixe as cavidades dos parafusos bem alinhadas. A válvula da câmara de gás deve estar virada para fora).

Passo 9 - Aperte bem todos os parafusos (posicione os parafusos nos lugares adequados. Com a chave de roda, dê uma volta em cada um. Daí aperte todos até o fim, até que estejam igualmente apertados).

Passo 10 - Desça o veículo e retire o macaco (com o macaco, faça o carro voltar totalmente ao chão. Aproveite para apertar uma vez mais os parafusos. Baixe totalmente o macaco e retire-o. Posicione a calota no lugar e guarde o pneu velho no porta-malas).

Algoritmo

Um **algoritmo** é uma sequência não ambígua de instruções que é executada até que determinada condição se verifique. Mais especificamente, em matemática, constitui o conjunto de processos (e símbolos que os representam) para efetuar um cálculo.

O conceito de algoritmo é frequentemente ilustrado pelo exemplo de uma receita, embora muitos algoritmos sejam mais complexos. Eles podem repetir passos (fazer iterações) ou necessitar de decisões (tais como comparações ou lógica) até que a tarefa seja completada. Um algoritmo corretamente executado não irá resolver um problema se estiver implementado incorretamente ou se não for apropriado ao problema.

Um algoritmo não representa, necessariamente, um programa de computador, e sim os passos necessários para realizar uma tarefa. Sua implementação pode ser feita por um computador, por outro tipo de autômato ou mesmo por um ser humano. Diferentes algoritmos podem realizar a mesma tarefa usando um conjunto diferenciado de instruções em mais ou menos tempo, espaço ou esforço do que outros. Tal diferença pode ser reflexo da complexidade computacional aplicada, que depende de estruturas de dados adequadas ao algoritmo.

Por exemplo, um algoritmo para se vestir pode especificar que você vista primeiro as meias e os sapatos antes de vestir a calça enquanto outro algoritmo especifica que você deve primeiro vestir a calça e depois as meias e os sapatos. Fica claro que o primeiro algoritmo é mais difícil de executar que o segundo apesar de ambos levarem ao mesmo resultado.

O conceito de um algoritmo foi formalizado em 1936 pela Máquina de Turing de Alan Turing e pelo cálculo lambda de Alonzo Church, que formaram as primeiras fundações da Ciência da Computação. 2

Lógica de Programação 'Primária' vs. 'Não Primária'

Lógica de Programação Primária: Base da programação, abrangendo conceitos básicos como:

- Estruturas de controle (sequenciais, condicionais e repetitivas)
- Variáveis e tipos de dados
- Operadores aritméticos, lógicos e relacionais
- Funções e procedimentos
- Algoritmos e resolução de problemas

Lógica de Programação Não Primária: Conceitos mais avançados, incluindo:

- Paradigmas de programação (orientação a objetos, funcional, lógica etc.)
- Estruturas de dados complexas (árvores, grafos etc.)
- Concorrência e paralelismo
- Programação para sistemas embarcados e distribuídos
- Inteligência artificial e machine learning
- Segurança e criptografia

Analogia:

- Lógica Primária: Alicerce de uma casa, fundamental para a estrutura.
- Lógica Não Primária: Acabamentos e detalhes, que personalizam e aprimoram a casa.

Exemplos:

- Lógica Primária: Calcular a média de notas de alunos.
- Lógica Não Primária: Criar um sistema de reconhecimento facial.

Níveis de Abstração:

- Lógica Primária: Mais concreta e próxima da linguagem natural.
- Lógica Não Primária: Mais abstrata e matemática.

Importância de Ambos:

- Lógica Primária: Essencial para qualquer programador.
- Lógica Não Primária: Permite a criação de programas mais complexos e eficientes.

Aprendizagem:

- Lógica Primária: Geralmente aprendida em cursos introdutórios de programação.
- Lógica Não Primária: Aprendida em cursos mais avançados, livros, tutoriais e prática.

Habilidades Essenciais:

- Lógica Primária: Raciocínio lógico, resolução de problemas, atenção aos detalhes.
- Lógica Não Primária: Criatividade, abstração, capacidade de aprender novos conceitos.

Lógica de programação 'primária' e 'não primária' são complementares e essenciais para a construção de programas eficientes e robustos. Comece com a base sólida da lógica primária e, em seguida, explore os diversos caminhos da lógica não primária para se tornar um programador completo.

Exercícios de logica descritiva

Exercício 1

Faça um algoritmo descritivo para calcular a média de 3 números, pesa para o usuário informar os números e mostre o resultado na tela.

Exercício 2

Faça um algoritmo descritivo que calcule o total a ser pago de um produto, sendo que o valor do produto e a quantidade deve ser informada pelo usuário. Mostre o total a ser pago na tela.

Exercício 3

Faça um algoritmo descritivo que leia três números, pesa para o usuário informar os números e apresente o menor deles.

Fluxograma

O fluxograma ilustra as etapas, sequências e decisões de um processo ou fluxo de trabalho. Embora haja vários outros tipos, um fluxograma básico é a forma mais simples de um mapa de processo. Trata-se de uma ferramenta robusta para planejar, visualizar, documentar e otimizar processos em diversas áreas de conhecimento.

Os engenheiros Frank e Lillian Gilbreth apresentaram o fluxograma para a American Society of Mechanical Engineers (ASME, ou Sociedade Estadunidense de Engenheiros Mecânicos) em 1921. Desde então, a ferramenta foi aprimorada e padronizada para otimizar processos em inúmeros setores. ³

Fluxograma é uma representação gráfica por meio de símbolos geométricos, da solução algorítmica de um problema, não necessariamente utilizado para algoritmos computacionais.

“A principal função do fluxograma é mapear e medir processos de maneira segmentada, compreensível e simples. Nesse sentido, a partir dos símbolos e desenhos, o fluxograma objetiva representar cada um dos estágios do processo. Por isso, o fluxograma é uma ferramenta de extrema importância.” ⁴

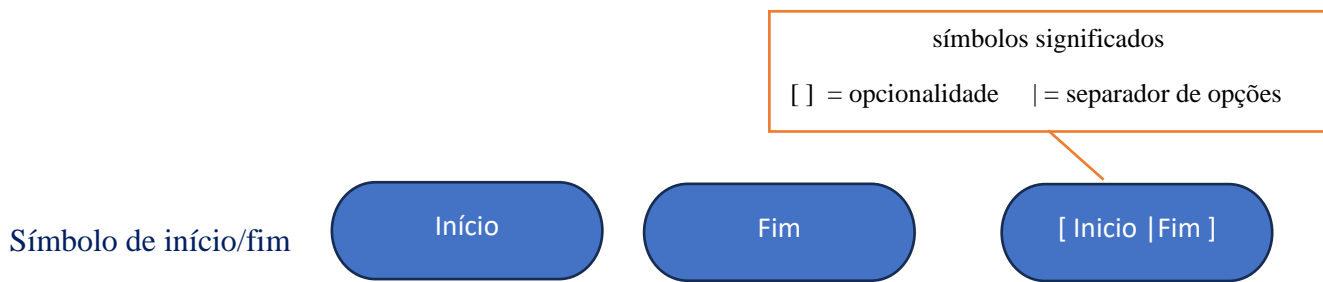
Fluxogramas em diferentes áreas

- **TI (programação):** o fluxograma está presente na programação para visualizar de forma mais simples a execução de um código, para demonstrar a organização do código e de várias outras formas.
- **Vendas:** o fluxograma é utilizado para mapear processo de venda, fazer levantamentos, planejar estratégias, entre outras formas.
- **Engenharias:** o fluxograma é utilizado para representar fluxos de processos, avaliar o ciclo de vida de uma estrutura, apresentar o protótipo de um produto.
- **Negócios:** o fluxograma é utilizado para documentar processos, entender o comportamento de clientes, descrever tarefas e diversas outras formas.
- **Educação:** o fluxograma é utilizado para desenvolver projetos, planejar cursos, criar plano de aulas e outras formas.

³ <https://asana.com/pt/resources/what-is-a-flowchart>

⁴ <https://www.voitto.com.br/blog/artigo/fluxograma>

Os símbolos utilizados no fluxograma.



Também conhecido como “Símbolo de terminação”, este símbolo representa os pontos iniciais, finais e resultados potenciais de um caminho. Muitas vezes contém “Início” ou “Fim” dentro da forma.⁵



Também conhecido como “Símbolo de dados”, esta forma representa dados disponíveis para entradas ou saídas, bem como representa recursos utilizados ou gerados. O símbolo da fita de papel também representa entrada/saída, no entanto, está desatualizado e não é mais de uso comum em fluxogramas.



Representa a entrada de dados manual. Exemplos de entrada de dados manual é o teclado.



Representa a entrada ou a saída de um documento, especificamente. Exemplos de entrada são o recebimento de um relatório, um e-mail ou um pedido. Exemplos de saída usando um símbolo de documento são geralmente uma apresentação, um memorando ou uma carta.



⁵ <https://medium.com/@valkcastellani/fluxograma-representa%C3%A7%C3%A3o-gr%C3%A1fica-de-um-algoritmo-88347c2c8fad>

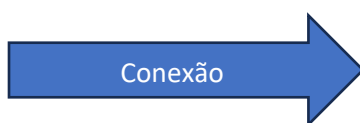
Também conhecido como “Símbolo de ação”, esta forma representa um processo, ação ou função. É o símbolo mais amplamente usado em fluxogramas.

Símbolo de decisão



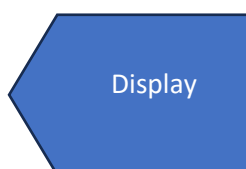
Indica uma questão a ser respondida, geralmente com sim/não ou verdadeiro/falso. O caminho do fluxograma pode se dividir em diferentes ramificações dependendo da resposta ou das consequências em seguida.

Símbolo de conexão



A “seta” é um símbolo de conexão, e serve para indicar uma interligação entre dois outros símbolos, e indica a direção do fluxo.

Símbolo de Display / Exibição



Uma etapa que exibe informações a uma pessoa “na tela”.⁶

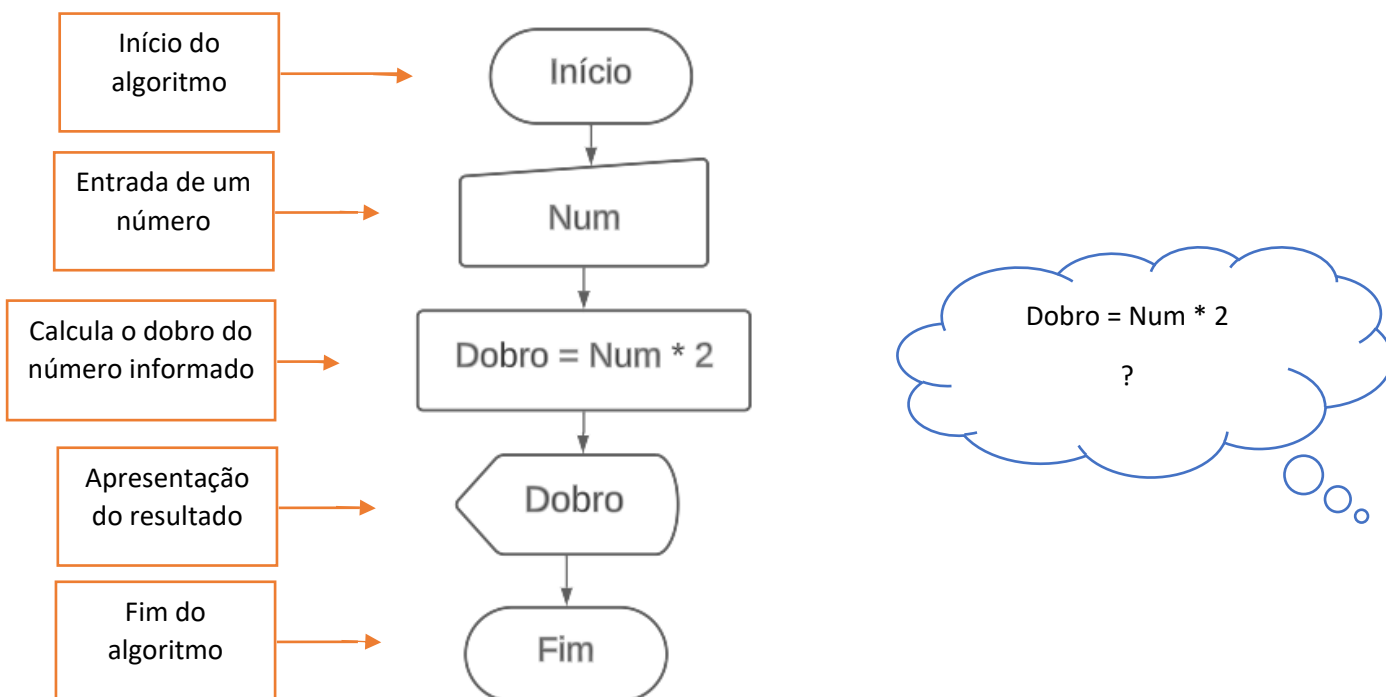
A figura a seguir mostra outros símbolos.

Operação	Decisão	Input Output	conexão de páginas
Inspeção	Preparação	Cartão perfurado	Preparação
Demora	Terminal	Memória principal	Decisão
Transporte	Junção	Sub-rotina	Display
Armazenamento	“Ou”	Tambor magnético	Extrair
Ações combinadas	Disco magnético	Conector	Vários documentos
Processo	Fita magnética	Classificar	Agrupar
Operação Manual	Documento	Fita papel perfurada	Entrada manual

Fonte - 2 - https://www.researchgate.net/figure/Figura-4-Simbologia-do-Fluxograma_fig1_355208109

⁶ <https://www.lucidchart.com/pages/pt/fluxograma-simbologia>, <https://www.venki.com.br/blog/significados-simbolos-fluxograma-de-processos/>, aconselho a leitura destes dois sites para conhecimento de outros símbolos, assim como para aumentar o conhecimento.

Exemplo: Fluxograma para calcular o dobro de determinado número informado:



que significa o processo do exemplo acima?

Pensando que o número em questão já é conhecido! O número é 3!

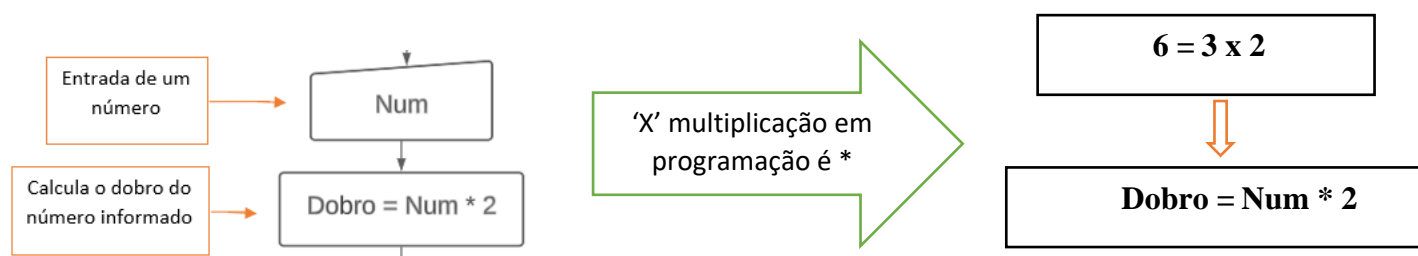
Então teremos a seguinte conta:

$$3 \times 2 = 6$$

Em programação o símbolo = “igual” pode significar atribuição, o mesmo que :

$$6 = 3 \times 2$$

Equação “Na matemática, uma equação é uma igualdade envolvendo uma ou mais incógnitas (valores desconhecidos)”.



Voltando para o fluxograma exemplo, vemos que temos duas incógnitas, o “**Num**” que será o número informado e o “**Dobro**” que é o resultado o qual depende do número informado, que também não sabemos!



Condicionais simples

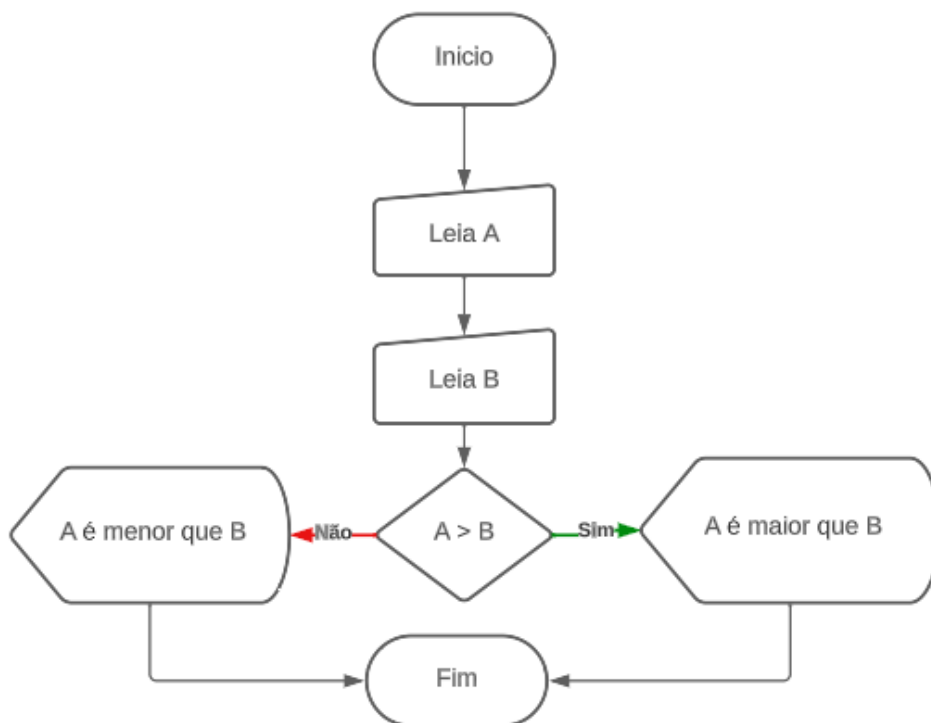
Em lógica de programação, as estruturas condicionais são recursos oferecidos pelas linguagens para que seja possível verificar uma condição e alterar o fluxo de execução do algoritmo. Assim, é possível definir uma ação específica para diferentes cenários e obter exatamente o resultado esperado.

As estruturas condicionais permitem que um programa execute diferentes comandos de acordo com as condições estabelecidas.

O uso das estruturas condicionais é praticamente indispensável na maioria dos projetos, já que elas são capazes de realizar diferentes funções de forma prática.

Uma estrutura condicional é baseada em uma condição que se for atendida o algoritmo toma uma decisão., contém pontos de decisão. Indica a sequência de funcionamento em processos simples, que depende de uma condição para executar um tipo de tarefa. ⁷

Exemplo: Fluxograma com uma condicional simples, que verifica se o conteúdo de A é maior ou menor que B:

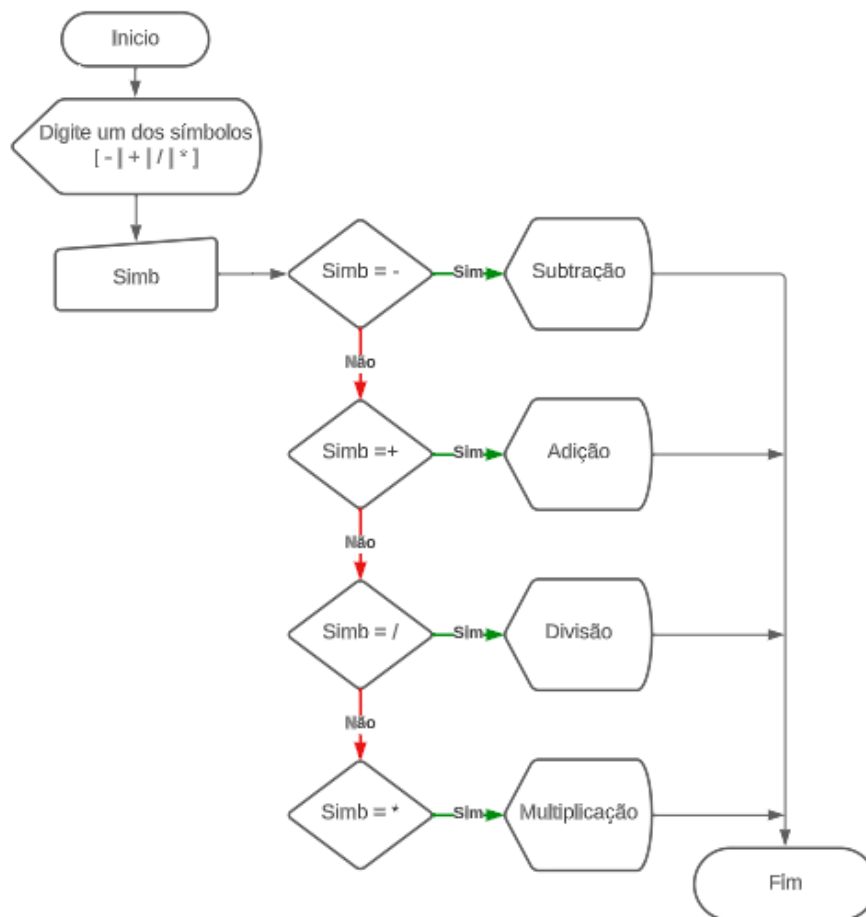


⁷ <https://rockcontent.com/br/talent-blog/estruturas-condicionais-2/>

Condicionais de múltipla escolha

A estrutura de seleção múltipla escolha/caso permite escolher entre várias opções de ação de acordo com um valor ou condição específica. É similar a uma estrutura condicional se/senão, mas possui a vantagem de poder avaliar mais de duas opções diferentes e apresentar um código mais enxuto. Em lógica de programs estruturas condicionais são recursos oferecidos pelas linguagens para que seja possível verificar uma condição e alterar o fluxo de execução do algoritmo. Assim, é possível definir uma ação específica para diferentes cenários e obter exatamente o resultado esperado.

Exemplo: Fluxograma com uma condicional de multipla escolha, que mostra o significado do simbolo matemático digitado:



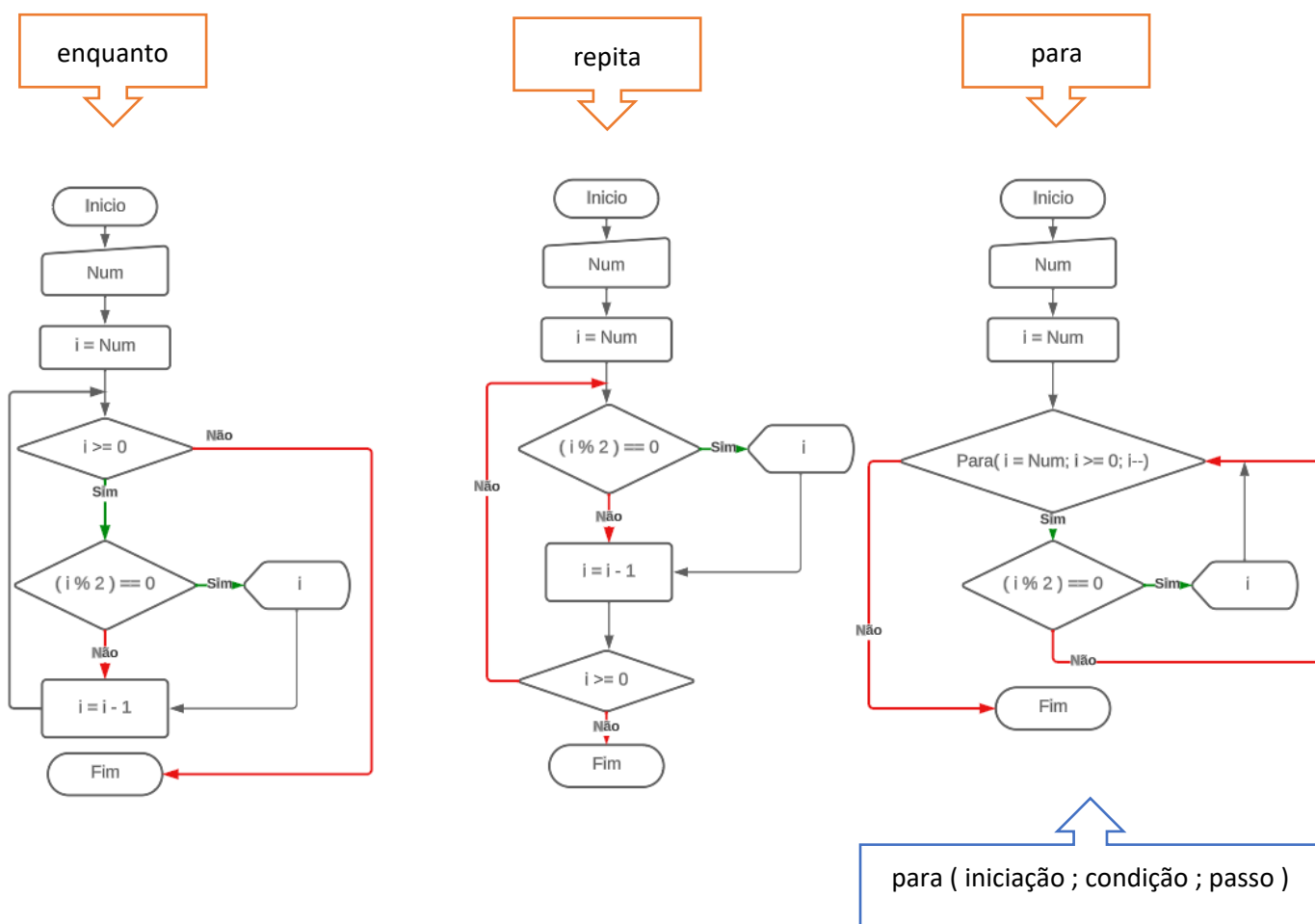
Estrutura de repetição

Estrutura utilizada quando é necessário efetuar a repetição de um trecho do algoritmo, cria um “**looping**” para efetuar o processamento tantas vezes quantas forem necessárias. Loopings são também conhecidos como **laços de repetição**.

Podemos utilizar tres tipos: **enquanto**, **repita** e **para**.

A principal vantagem é que o algoritmo passa a ter um tamanho menor, podendo ampliar o processamento, sem alterar o tamanho do código.

Exemplo: Fluxograma com laço de repetição que mostra os numeros pares anteriores a partir de um número informado:



A diferença entre um diagrama de fluxo de dados (DFD) e um fluxograma (FC) é que um diagrama de fluxo de dados normalmente descreve o fluxo de dados dentro de um sistema e o fluxograma geralmente descreve a lógica detalhada de um processo de negócios.

Utilizando o Lucidchart

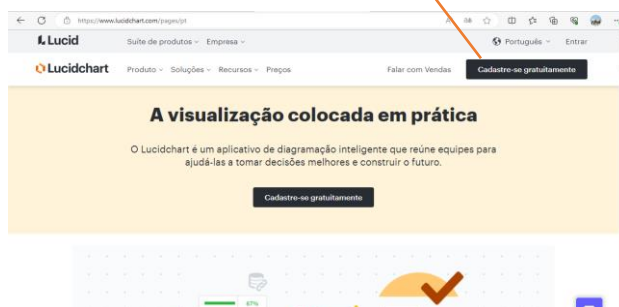
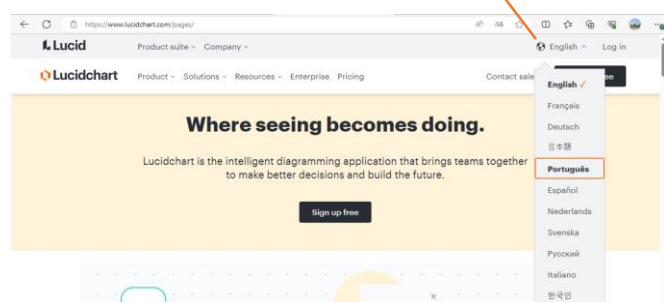
O Lucidchart é uma plataforma disponível para PC (Windows, macOS ou Linux), Google Chrome, além de ser possível fazer download gratuito pelo celular Android ou iPhone (iOS). Ao fazer login e entrar no Lucidchart, o usuário pode criar e mostrar diagramas, fluxogramas, além de mapa mental.

Acesso

Link: <https://www.lucidchart.com>

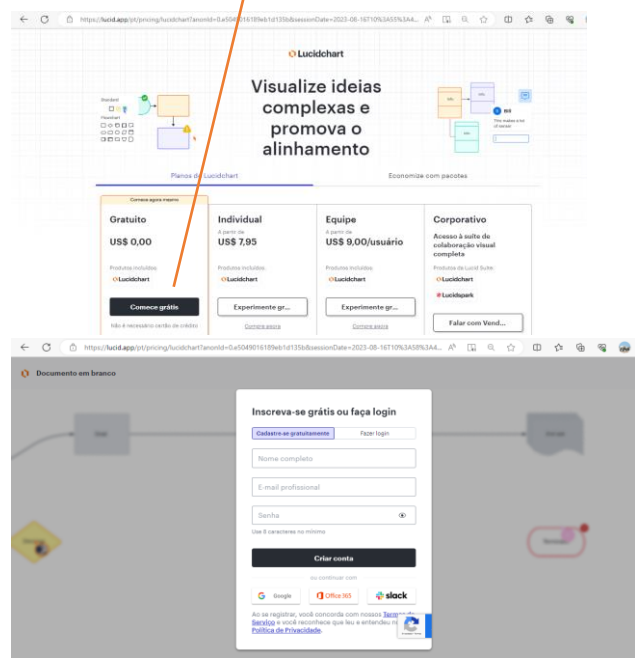
Você pode escolher o idioma de sua preferência

Clique em cadastre-se gratuitamente



Para os nossos exercícios a versão gratuita irá atender

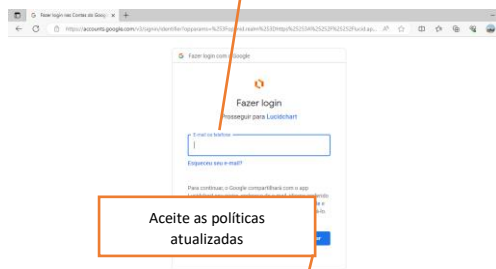
É necessário realizar o registro, mas podemos utilizar a conta google se quiser



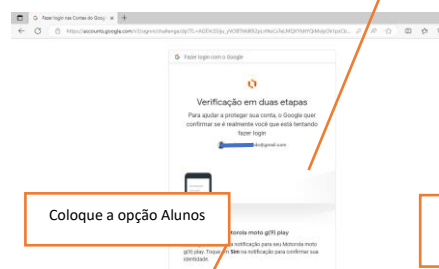
Se optar em utilizar a conta google, entre com o seu gmail

Você vai precisar confirmar a validação pelo celular, em caso de outra conta será por e_mail

Se for pelo celular basta clicar em sim! Se não, deve entrar no e-mail informado.



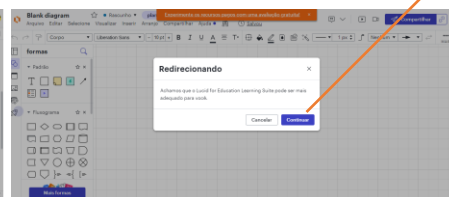
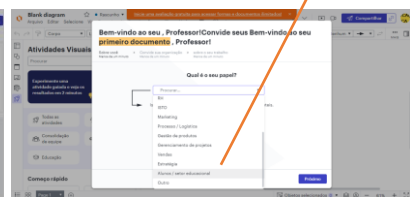
Aceite as políticas atualizadas



Coloque a opção Alunos

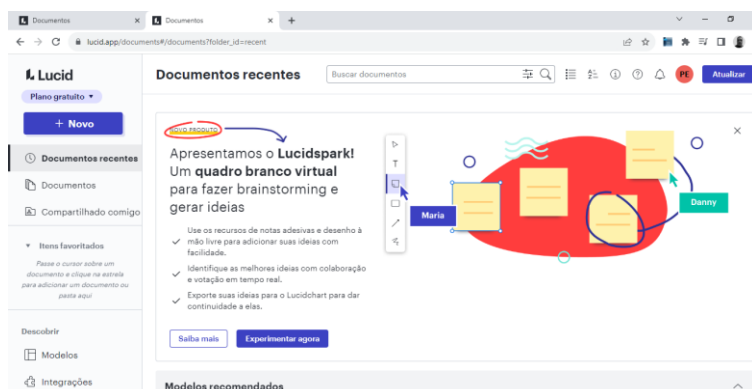
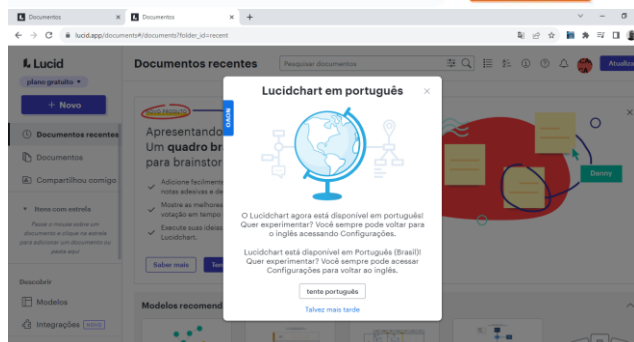
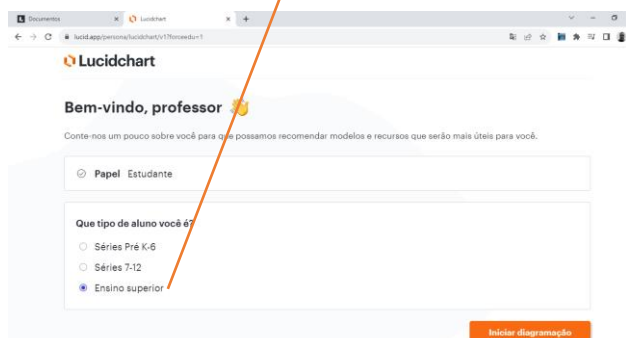


Confirme o redirecionamento para versão estudante



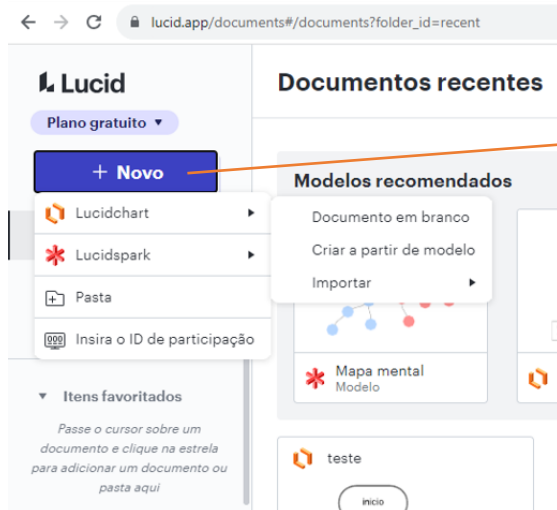
Indique o perfil

Você pode optar por utilizar a versão em português



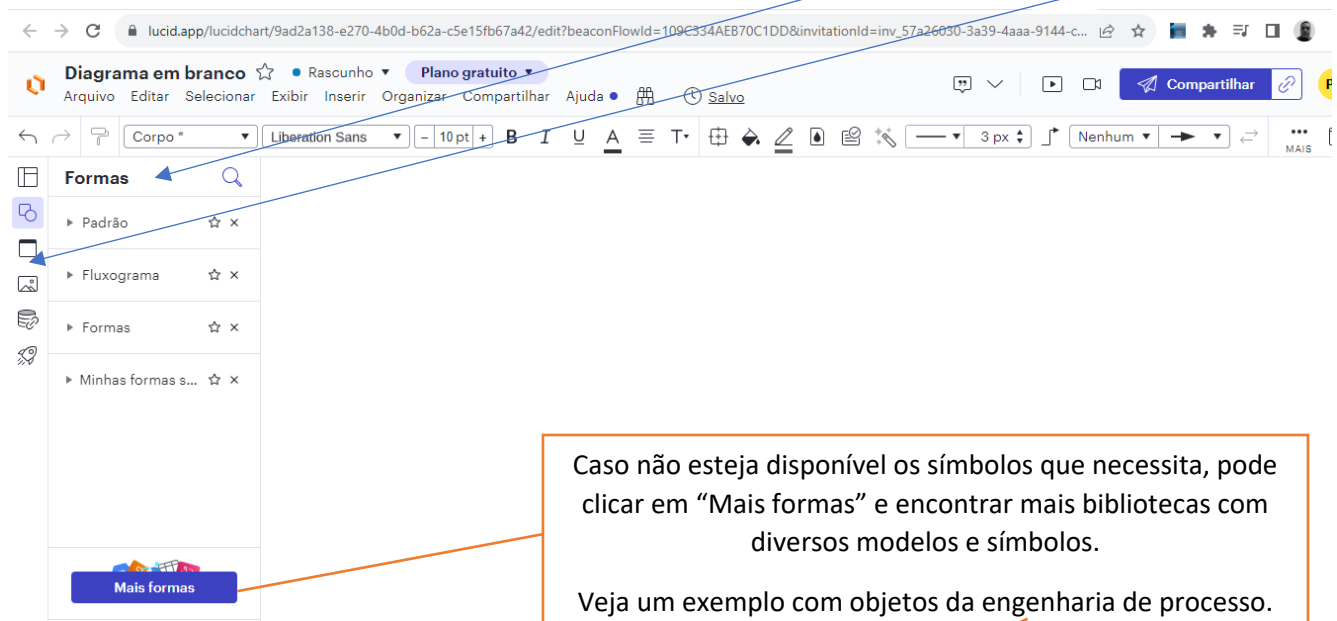
Ok, estamos prontos para começa!





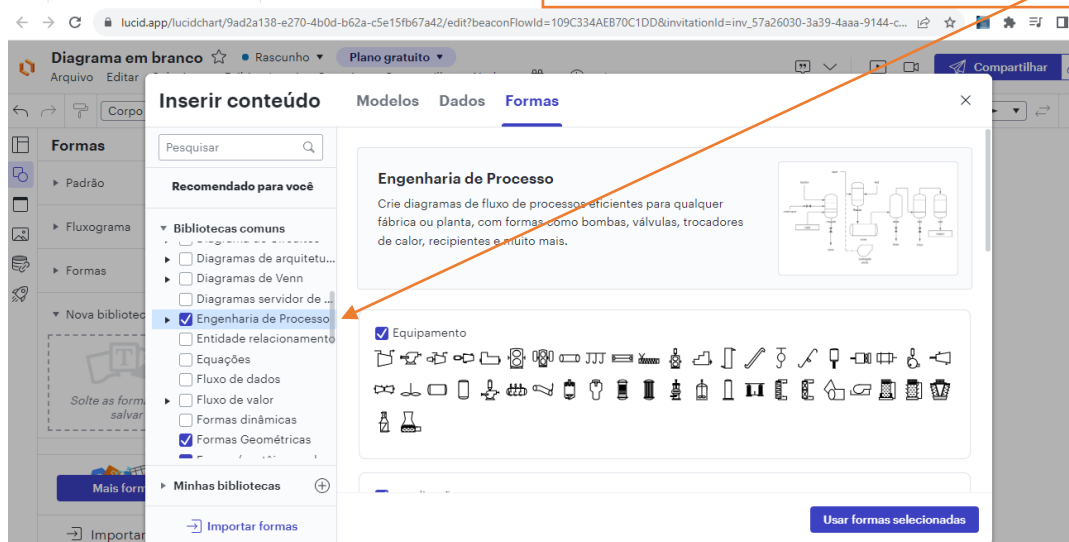
Para iniciar o nosso exemplo, clique em “+Novo”/ Lucidchart/ Documento em branco.

Observe do lado esquerdo as formas que podemos utilizar para criar diversos diagramas, também uma palheta com modelos, desenhos e outros.

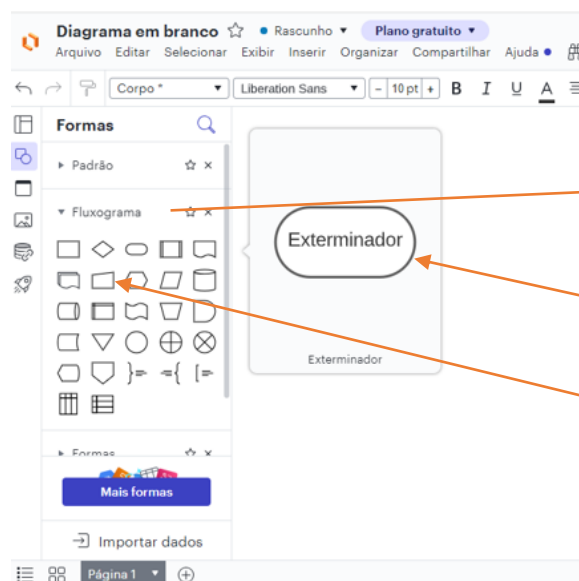


Caso não esteja disponível os símbolos que necessita, pode clicar em “Mais formas” e encontrar mais bibliotecas com diversos modelos e símbolos.

Veja um exemplo com objetos da engenharia de processo.



Exemplo



Em fluxograma encontraremos os objetos que utilizaremos para um fluxograma de programação.

Aqui o símbolo início é chamado de "Exterminador".

Símbolo de início. Para incluir basta clicar no símbolo.

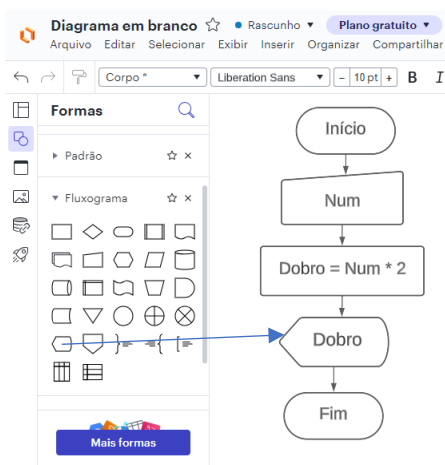
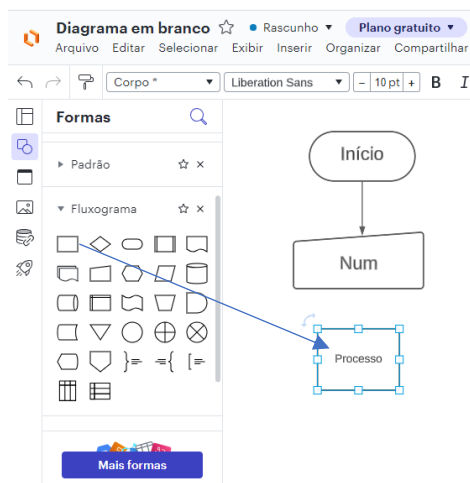


Para incluir o texto, basta estar com o objeto selecionado. Também pode formatar o texto na barra de opções acima (tamanho do texto, tipo, negrito).

Para alterar o tamanho do objeto, com ele selecionado, clique nos quadrados em torno do objeto e arraste até atingir o tamanho desejado.



Para conectar os objetos, você pode selecionar o objeto e clicar na bolinha vermelha, depois basta arrastar até o objeto que será conectado.



Expressões aritméticas

Usamos operadores aritméticos para construir expressões matemáticas que permitam a realização de cálculos e execução de fórmulas em um algoritmo. Os operadores aritméticos seguem as mesmas regras da matemática básica, incluindo a ordem de precedência dos operadores e sua simbologia, com algumas exceções.⁸

Operadores aritméticos

Os operadores aritméticos executam operações matemáticas. O "módulo" é o resto da divisão inteira. Considere que **a=10** e **b=20**.⁹

Operador	Descrição	Exemplo	Resultado
+	Soma	$a + b$	30
-	Subtração	$a - b$	-10
*	Multiplicação	$a * b$	200
/	Divisão inteira	a / b	2
%	Módulo / resto da divisão	$a \% b$	0
++	Incremento	$a++$	11
--	Decremento	$a--$	9

⁸ <http://www.bosontreinamentos.com.br/logica-de-programacao/06-logica-de-programacao-operadores-e-expressoes-aritmeticas/>

⁹ https://www.inf.ufpr.br/roberto/ci067/02_operad.html

Expressões lógicas

Expressão lógica é uma expressão algébrica cujos operadores são os operadores lógicos e cujos operandos são relações (resultados de operações relacionais) e/ou variáveis do tipo lógico.

O resultado de uma expressão lógica é sempre um valor lógico: VERDADEIRO ou FALSO.

Operadores lógicos

Os operadores lógicos são utilizados quando é necessário usar duas ou mais condições dentro da mesma instrução “Condicional (Se)” para que seja tomada uma única decisão cujo resultado será verdadeiro ou falso.¹⁰

Os operadores lógicos combinam condições simples em expressões lógicas. O valor de retorno se uma expressão lógica é verdadeira ou falsa.

Os operadores lógicos são:

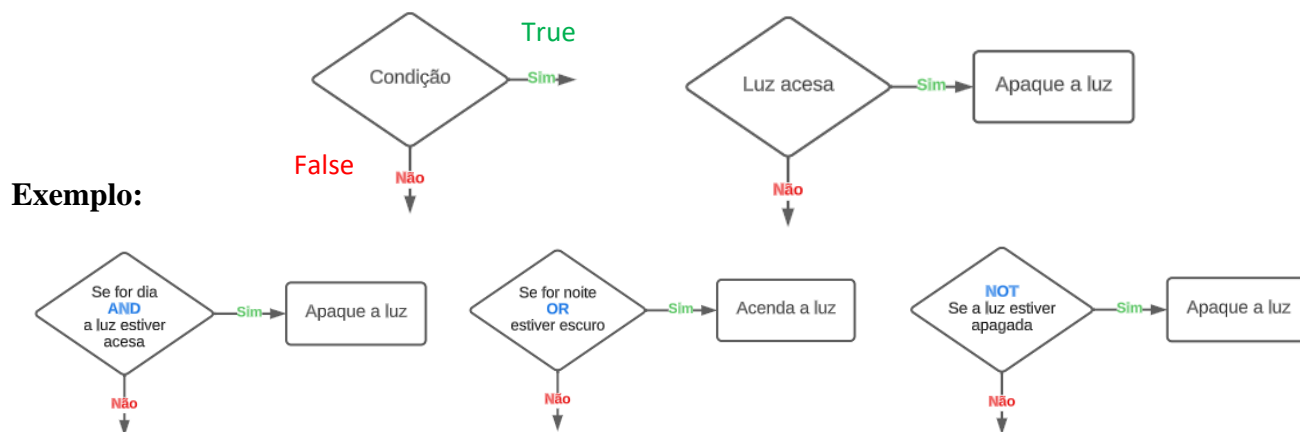
- **&&** (E/AND)
- **||** (OU/OR)
- **!** (NÃO/NOT)

Para o operador && (E) o resultado será verdadeiro caso todas as comparações sejam verdadeiras.

Usando o operador || (OU) o resultado será verdadeiro bastando apenas uma das comparações ser verdadeira.

O operador lógico de negação ! (NÃO) é utilizado para inverter o resultado de uma determinada condição. Ou seja, se a condição for verdadeira esta torna-se falsa, e se a condição for “falsa” ela torna-se verdadeira.

A	B	A And B	A Or B	Not A
A	B	A && B	A B	!A
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE



¹⁰ <https://linguagemc.com.br/operadores-logicos-em-c/>

Operadores relacionais

Operadores relacionais são utilizados para comparar valores, o resultado de uma expressão relacional é um valor booleano (VERDADEIRO ou FALSO). Os operadores relacionais são: igual, diferente, maior, menor, maior ou igual, menor ou igual.

Símbolo	Nome do Operador	Exemplo	Significado
>	Maior que	$x > y$	x é maior que y
>=	Maior ou igual	$x >= y$	x é maior ou igual a y
<	Menor que	$x < y$	x é menor que y
<=	Menor ou igual	$x <= y$	x é menor ou igual a y
==	Igualdade	$x == y$	x é igual a y
!=	Diferente de	$x != y$	x é diferente de y

Português Estruturado (Pseudocódigo)

Pseudocódigo é um método de descrever um processo ou escrever código de programação e algoritmos usando uma linguagem natural.

Ou seja, o pseudocódigo não é o código em si, mas sim uma descrição do que o código deve fazer. Ele é usado como um plano ou passo a passo detalhado, mas compreensível, a partir do qual um programa pode ser escrito, o pseudocódigo nada mais é do que um rascunho de um programa ou algoritmo antes de ser implementado em uma linguagem de programação.

Tipos de Dados

Numérico

Específicas para armazenamento de números, que posteriormente poderão ser utilizados para cálculos.

- **Inteiro** - Números inteiros, que não possuam casas decimais, como 0, 1, 100, 2000, 3054.
- **Real** - Números que possuem casas decimais, números negativos, fracionários, como 0.25, 1.44, 3.22, 20.10, -30.54.

Caractere

Essas variáveis são utilizadas para armazenamento de conjunto de caracteres que não contenham números (literais). Ex: nomes, cargos etc.

Lógico

Armazenam somente dados lógicos que podem ser Verdadeiro ou Falso ou 0 e 1

Variáveis

Um algoritmo manipula dados, que podem ser dados variáveis ou constantes. Dados variáveis são apresentados por variáveis, enquanto dados constantes são representados por constantes!

Uma variável pode ser imaginada como um “caixa” para armazenar valores de dados. Esta caixa só pode armazenar um único valor por vez. No entanto, o valor armazenado na caixa pode mudar inúmeras vezes durante a execução do algoritmo. Em um ambiente computacional de verdade, a caixa correspondente a uma variável é uma posição da memória do computador.

Uma variável possui nome, tipo e conteúdo.

O nome de uma variável deve ser único, isto é, identificar, de forma única, a variável no algoritmo. O tipo de uma variável define os valores que podem ser armazenados na variável. O conteúdo de uma variável é o valor que ela armazena. É importante lembrar que uma variável só pode armazenar um valor de cada vez. No entanto, ela pode assumir vários valores distintos do mesmo tipo durante a execução do algoritmo.

O ato de se criar uma variável é conhecido como declaração de variável. Na linguagem Portugol, declaramos uma variável usando uma sentença da seguinte forma:

var nome : tipo

- **var** indica que uma variável será criada
- **nome** é o nome da variável
- **tipo** é o tipo da variável.

Por exemplo:

var lado : real

declara uma variável de nome lado do tipo real.

Podemos declarar mais de uma variável do mesmo tipo em uma mesma linha.

Por exemplo:

var lado, area : real

Note que nenhum conteúdo (“valor”) foi associado à variável durante a sua declaração. Esta associação é denominada definição e deve ser realizada após a declaração da variável usando uma instrução de leitura ou um comando de atribuição.

A instrução de leitura tem a forma, exemplos:

leia (nome) “onde nome é o nome de uma variável.”

leia (lado) “é uma instrução de leitura que atribui um valor à variável lado.

O valor atribuído pela instrução deve ser fornecido como entrada para o algoritmo durante a sua execução.

Exemplos de leitura do comprimento dos lados de um quadrado.

A instrução de atribuição possui a forma:

nome <- valor

- Onde nome é o nome de uma variável.
- Valor é um valor do mesmo tipo de dados da variável.

Por exemplo:

lado <- 2.5

Atribui o valor 2.5 à variável de nome lado. Para que uma instrução de atribuição faça sentido, a variável lado deve ser do tipo real e deve ter sido declarada antes da instrução de atribuição ser executada.

- O símbolo <- é conhecido como operador de atribuição.

Muitas vezes, o valor atribuído a uma variável através de uma instrução de atribuição é resultante de uma expressão aritmética ou outro tipo de expressão.

Por exemplo:

area <- lado * lado

É uma instrução de atribuição que atribui o valor da variável lado ao quadrado à variável area.

O que vemos no lado direito do operador de atribuição, **lado * lado**, é um exemplo de expressão aritmética.

Um valor atribuído a uma variável permanece associado a ela até que uma instrução de atribuição, que o substitua por outro, seja executada. Em qualquer dado instante de tempo durante a execução de um algoritmo, o valor armazenado em uma qualquer variável (se algum) é denominado valor atual (ou valor corrente) da variável. Enquanto não atribuirmos um valor a uma variável, a variável permanecerá com valor desconhecido.

Finalmente, é importante lembrar que uma variável só poderá receber um valor através de uma instrução de leitura ou atribuição.

Exemplos

Seguem abaixo alguns exemplos de declaração de variáveis:

var fruta : caractere

var letra : caractere

var resultado : logico

var altura : real

var idade : inteiro

A seguir, temos exemplos de instruções de atribuição que atribuem valores a essas variáveis:

fruta <- "maçã"

letra <- "c"

resultado <- falso

altura <- 1.83

idade <- 5

As mesmas variáveis podem ter valores atribuídos a elas através de instruções de leitura como segue:

leia (fruta)

leia (letra)

leia (altura)

leia (idade)

Note que não escrevemos uma instrução de leitura para a variável resultado. Isto se deve ao fato de instruções de leitura não poderem ser aplicadas a variáveis do tipo logico.

Nomes de variáveis

Na linguagem Portugol, usamos as seguintes regras para criar um nome de variável:

- Nomes de variáveis devem possuir como primeiro caractere uma letra ou o símbolo ' _ ' (sublinhado). Os demais caracteres, se algum, devem ser letras, números ou sublinhado.
- Nomes de variáveis não podem ser iguais a palavras reservadas.
- Nomes de variáveis podem ter, no máximo, 127 caracteres.
- Não há diferença entre letras maiúsculas e minúsculas em nomes de variáveis.
- nomes de variáveis não podem conter espaços em branco.
- nomes de variáveis não podem ser palavras reservadas da linguagem Portugol.

Uma palavra reservada é uma palavra que possui um significado especial para a linguagem Portugol. Em geral, uma palavra reservada identifica uma instrução.

O conjunto de palavras reservadas do Portugol é mostrado na tabela a seguir.

aleatorio	e	grauprad	passo
abs	eco	inicio	pausa
algoritmo	enquanto	int	pi
arcos	entao	interrompa	pos
arcsen	escolha	leia	procedimento
arctan	escreva	literal	quad
arquivo	exp	log	radpgrau
asc	faca	logico	raizq
ate	falso	logn	rand
caractere	fimalgoritmo	maiusc	randi
caso	fimenquanto	mensagem	repita
compr	fimescolha	minusc	se
copia	fimfuncao	nao	sen
cos	fimpara	numerico	senao
cotan	fimprocedimento	numpcarac	timer
cronometro	fimrepita	ou	tan
debug	fimse	outrocaso	var
declare	funcao	para	verdadeiro
			xou

Por exemplos de nomes de variáveis

São nomes válidos para variáveis:

- _12234
- Fruta
- x123

Não são nomes válidos para variáveis:

- maria bonita
- pi
- fru?ta
- 1xed

O nome “maria bonita” contém um espaço em branco.

O nome “pi” é uma palavra reservada.

O nome “fru?ta” contém um caractere que não é letra, número nem sublinhado.

O nome “1xed” inicia com um número.

Constantes

Uma constante é uma variável! No sentido de que uma constante também reserva um espaço de memória para o tipo de dado que manipulará. Entretanto, uma constante armazenará um valor ÚNICO, um valor que NÃO mudará com o tempo de execução do programa.

Define um símbolo cujo valor permanece inalterável durante o seu ciclo de vida. Segue as mesmas regras que a definição de variáveis exceto que não é possível omitir o valor de inicialização.

Sintaxe

constante [tipo] [nome] <- [valor]

constante [tipo] [nome] <- [expressão]

constante [tipo] [nome] <- [valor] , [nome] <- [expressão]

Exemplos

constante inteiro meses <- 12

constante real pi <- 3.14

Sintaxe

Principais instruções utilizadas nos pseudocódigos no “VisualG”

COMANDO	UTILIZADO PARA
escreva (“ ”)	comando usado para imprimir uma mensagem na tela.
leia ()	comando usado para ler valores digitados no teclado.
início	comando para iniciar o programa principal.
fimalgoritmo	comando para finalizar o algoritmo.
var	comando para declarar variáveis.
<-	comando de atribuição.
+	somar dois valores.
–	subtrair dois valores.
algoritmo	comando para indicar o início do programa.
var	declaração de variável
se / então / Senão / fimse	condicional (Se)

LINGUAGEM DE PROGRAMAÇÃO C SHARP “C#”

C# é uma linguagem de programação versátil e robusta.

C# (pronunciado "C Sharp") é uma linguagem de programação moderna, orientada a objetos e de tipagem forte, desenvolvida pela Microsoft como parte da plataforma .NET. Ela oferece uma sintaxe clara e concisa, inspirada em linguagens como C++ e Java, tornando-a amigável para iniciantes e experientes.

Características-chave:

- **Orientada a objetos:** Permite criar classes, objetos e interfaces, promovendo reutilização de código, modularidade e flexibilidade.
- **Tipagem forte:** Garante segurança e confiabilidade no código, evitando erros de tipo em tempo de execução.
- **Multiplataforma:** Suporta desenvolvimento para Windows, macOS, Linux, web e dispositivos móveis, com .NET Core e Xamarin.
- **Versátil:** Cria desde aplicações web complexas até jogos, software desktop e aplicativos móveis multiplataforma.
- **Código aberto:** A linguagem C# é open source, permitindo que a comunidade contribua para seu desenvolvimento e aprimoramento.

Vantagens de usar C#:

- **Facilidade de aprendizado:** Sintaxe intuitiva e similar a outras linguagens populares.
- **Grande comunidade:** Ampla base de desenvolvedores, com vasta documentação, tutoriais e ferramentas disponíveis.
- **Alto desempenho:** Código eficiente e otimizado para .NET.
- **Suporte da Microsoft:** Integração com o Visual Studio e outras ferramentas da Microsoft.
- **Ampla gama de aplicações:** Versatilidade para diversos tipos de projetos.

Recursos úteis:

Site oficial C#: <https://dotnet.microsoft.com/en-us/languages/csharp>

Documentação oficial: <https://learn.microsoft.com/en-us/dotnet/>

Tutoriais Microsoft Learn: <https://learn.microsoft.com/en-us/dotnet/csharp/>

Exemplos de aplicações C#:

- Aplicações web: ASP.NET Core, ASP.NET MVC
- Aplicações desktop: Windows Forms, WPF
- Jogos: Unity
- Aplicações móveis: Xamarin
- Software empresarial: .NET Framework

C# é uma linguagem de programação poderosa, versátil e com grande potencial. Se você busca uma linguagem para iniciar sua carreira em desenvolvimento ou deseja ampliar seus conhecimentos, C# é uma excelente escolha.

Tipos de dados em c#

Em C#, existem vários tipos de dados que podem ser usados para armazenar diferentes tipos de valores.

Tipos de Dados Numéricos:

- **short:** Representa números inteiros curtos.
- **int:** Representa números inteiros.
- **long:** Representa números inteiros longos.
- **float:** Representa números de ponto flutuante de precisão simples.
- **double:** Representa números de ponto flutuante de precisão dupla.
- **decimal:** Representa números decimais de alta precisão.

Tipos de Dados de Texto:

- **string:** Representa uma sequência de caracteres.
- **char:** Representa um único caractere Unicode.

Tipos de Dados Booleanos:

- **bool:** Representa um valor booleano verdadeiro ou falso.

Tipos de Dados de Data e Hora:

- **DateTime:** Representa uma data e hora.
- **TimeSpan:** Representa um intervalo de tempo.

Tipos de Dados Especiais:

- **object:** Representa qualquer tipo de dado.
- **dynamic:** Permite o atraso da resolução de tipo para tempo de execução.
- **var:** Permite a inferência de tipo em tempo de compilação.

Tipos de Dados de Coleções:

- **Array:** Representa uma coleção de elementos do mesmo tipo.
- **List<T>:** Representa uma lista dinâmica de elementos do tipo T.
- **Dictionary<TKey, TValue>:** Representa uma coleção de pares chave-valor.

Tipos de Dados Enumeração:

- **enum:** Define um tipo de valor nomeado composto de constantes nomeadas.

Esses são apenas alguns dos tipos de dados disponíveis em C#. Cada um desses tipos de dados tem seu próprio propósito e é usado para armazenar diferentes tipos de informações em um programa C#.

Tipos de valor: Armazenam o valor do dado diretamente na variável. São copiados quando atribuídos a outra variável ou passados como argumento para um método. Exemplos: tipos numéricos, char, bool.

Tipos de referência: Armazenam um ponteiro para um local na memória onde o valor real do dado está armazenado. Quando atribuídos a outra variável ou passados como argumento para um método, a referência é copiada, não o valor em si. Exemplos: string, class, interface, array.

Tabela que lista os tipos de dados em C# juntamente com suas capacidades e faixas de valores:

Tipos de valor				
Tipo de Dado	Descrição	(bits)	Capacidade	Detalhes Adicionais
bool	falso).	1	1 bit	-
byte	Representa números inteiros de 8 bits sem sinal.	8	0 a 255	-
sbyte	Representa números inteiros de 8 bits sem sinal.	8	-128 a 127	-
char	Representa caracteres Unicode de 16 bits.	16	0 a 65535	Suporta caracteres Unicode, incluindo emojis.
decimal	Representa números inteiros de 96 bits com sinal com 28-29 dígitos significativos.	96	-292.967.295.242.883.279.024 a 292.967.295.242.883.279.023	Ideal para cálculos de alta precisão e representação de valores monetários.
double	Representa números de ponto flutuante de 64 bits.	64	-1.7976931348623157e+308 a 1.7976931348623157e+308	Ampla faixa de valores, mas menor precisão que decimal.
float	Representa números de ponto flutuante de 32 bits.	32	-3.402823e+38 a 3.402823e+38	Menor faixa de valores e precisão que double, mas mais eficiente em termos de memória.
int	Representa números inteiros de 32 bits.	32	-2.147.483.648 a 2.147.483.647	Tipo de inteiro mais comum, adequado para a maioria das aplicações.
uint	Representa números inteiros de 32 bits.	32	0 a 4.294.967.295	
long	Representa números inteiros de 64 bits.	64	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Grande faixa de valores para cálculos complexos.
ulong	Representa números inteiros de 64 bits.	64	0 a 18,446,744,073,709,551,615	Grande faixa de valores para cálculos complexos.
short	Representa números inteiros de 16 bits.	16	-32,768 a 32,767	Menor faixa de valores que int, mas mais eficiente em termos de memória.
ushort	Representa números inteiros de 16 bits sem sinal.	16	0 a 65535	Menor faixa de valores que int, mas mais eficiente em termos de memória.
struct	Representa estruturas de dados compostas que contêm campos de vários tipos.	Variável	Depende dos campos da estrutura	Estruturas são úteis para agrupar dados relacionados e podem ter métodos próprios.
string	Representa cadeias de caracteres.	Variável	Até 2 GB (limite do .NET)	Armazena sequências de caracteres Unicode.
array	Representa coleções de elementos do mesmo tipo.	Variável	Depende do tamanho do array	Eficiente para armazenar e acessar grandes quantidades de dados do mesmo tipo.

Tipos de referência		
Tipo de Dado	Descrição	Capacidade
classe	Objetos definidos pelo usuário com propriedades, métodos e eventos	Armazena estado através de propriedades. * Define comportamentos através de métodos. * Encapsula lógica através de eventos. * Suporta herança para reuso de código. * Permite polimorfismo para implementar diferentes comportamentos.
Interface	Define um conjunto de métodos e propriedades que as classes devem implementar	Especifica um contrato para classes implementarem. * Promove a modularidade e o design orientado a objetos. * Permite desacoplamento entre classes. * Suporta programação baseada em interfaces.
Delegado	Uma referência a um método que pode ser chamado por meio de um ponteiro de função	Armazena uma referência a um método como um objeto. * Permite encapsular métodos em variáveis ou coleções. * Facilita a programação assíncrona e eventos. * Suporta programação funcional em C#.
Array	Coleção de elementos do mesmo tipo	Armazena um conjunto de valores do mesmo tipo. * Permite acesso ordenado aos elementos por meio de índices. * Suporta operações como pesquisa, inserção, remoção e classificação. * Oferece eficiência para processamento de coleções de dados.
Enum	É um dicionário onde a chave é o valor enum e o valor é a descrição correspondente.	abreviação de enumeração, é um tipo de dados definido pelo usuário que representa uma coleção de constantes nomeadas.

Lembre-se de que a capacidade de armazenamento e a faixa de valores podem variar dependendo do tipo de dado e do sistema em que o código está sendo executado.

Variáveis e Constantes em C#

Variáveis:

Em C#, variáveis são usadas para armazenar dados na memória do computador. Elas possuem um nome, um tipo de dado e um valor. O tipo de dado determina qual tipo de informação a variável pode armazenar, como números, strings ou objetos.

Declaração de variáveis:

```
// Declarando uma variável do tipo int
```

```
int numero;
```

```
// Declarando uma variável do tipo int, e atribuindo um valor já na declaração.
```

```
int numero2 = 10;
```

```
// Declarando uma variável do tipo string
```

```
string nome;
```

```
// Declarando uma variável do tipo string, e atribuindo um valor já na declaração.
```

```
string nome1 = "João";
```

```
// Declarando uma variável do tipo bool
```

```
bool isAtivo;
```

```
// Declarando uma variável do tipo bool, e atribuindo um valor já na declaração.
```

```
bool isAtivo = true;
```

Tipos de dados:

Tipos primitivos: int, float, double, bool, char, string

Tipos de referência: classes, interfaces, structs, arrays

Modificadores de acesso:

public: acessível em qualquer lugar do programa

private: acessível apenas na classe em que foi declarada

internal: acessível apenas no assembly em que foi declarada

protected: acessível na classe em que foi declarada e em classes derivadas

Palavras-chave:

var: inferência de tipo

const: declarar uma constante

Constante:

Uma constante é um valor que não pode ser alterado após a inicialização.

Declaração de constantes em C#

```
// Declarando uma constante do tipo int
```

```
const int PI = 3.14;
```

```
// Declarando uma constante do tipo string
```

```
const string GREETING = "Olá";
```

Use o código com cuidado.

Diferenças entre variáveis e constantes:

Enquanto as variáveis são espaços na memória do computador que armazenam valores que podem mudar ao longo do tempo, as constantes representam valores fixos que não são alterados durante a execução do programa. Ambas são importantes para a construção de programas robustos e legíveis.

Exemplos:**// Variável**

```
int numero = 10;
```

```
numero = 20;
```

// Constante

```
const int PI = 3.14;
```

Recursos:

C# - Variáveis e Constantes: <https://www.devmedia.com.br/introducao-a-variaveis-e-constantas-no-csharp/29629>

Constantes (Guia de Programação em C#): <https://learn.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/constants>

Instruções de declaração – variáveis locais e constantes, variáveis, variáveis de referência local (ref local): <https://learn.microsoft.com/pt-br/dotnet/csharp/language-reference/statements/declarations>

Microsoft Visual Studio

O Microsoft Visual Studio é um ambiente de desenvolvimento integrado (IDE) desenvolvido pela Microsoft. Ele é usado para desenvolver programas de computador, sites, aplicativos da web, serviços da web e aplicativos móveis. O Visual Studio suporta várias linguagens de programação, incluindo C/C++, C#, Visual Basic .NET, F#, Python, JavaScript, TypeScript e outras.

Recursos principais do Visual Studio incluem:

- **Editor de Código:** O Visual Studio oferece um editor de código poderoso com recursos como destaque de sintaxe, conclusão de código e IntelliSense, que fornece sugestões de código contextuais.
- **Depurador:** O Visual Studio inclui um depurador que permite aos desenvolvedores depurar seu código definindo pontos de interrupção, inspecionando variáveis e passando pela execução do código.
- **Gerenciamento de Projetos e Soluções:** O Visual Studio organiza o código em projetos e soluções, facilitando o gerenciamento de grandes bases de código com vários arquivos e dependências.
- **Integração com Controle de Versão:** O Visual Studio se integra a sistemas de controle de versão como Git, permitindo que os desenvolvedores gerenciem mudanças no código-fonte e colaborem com membros da equipe.
- **Extensões e Complementos:** O Visual Studio suporta extensões e complementos que ampliam sua funcionalidade. Essas extensões podem adicionar novos recursos, ferramentas e suporte a idiomas ao IDE.
- **Desenvolvimento Multiplataforma:** O Visual Studio oferece suporte ao desenvolvimento para várias plataformas, incluindo Windows, macOS, Android, iOS e Linux.
- **Desenvolvimento na Nuvem:** O Visual Studio fornece integração com serviços de nuvem como o Azure, permitindo que os desenvolvedores criem e implantem aplicativos nativos da nuvem diretamente do IDE.

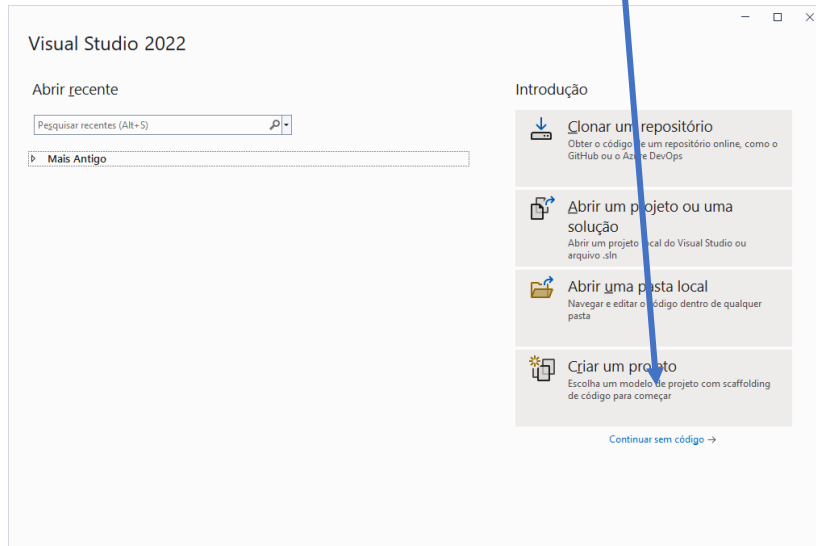
O Visual Studio está disponível em diferentes edições, incluindo Community (gratuita), Professional e Enterprise, cada uma oferecendo recursos e capacidades diferentes adaptados às necessidades de desenvolvedores individuais ou equipes.

Veja edital de instalação disponibilizado pelo seu professor.



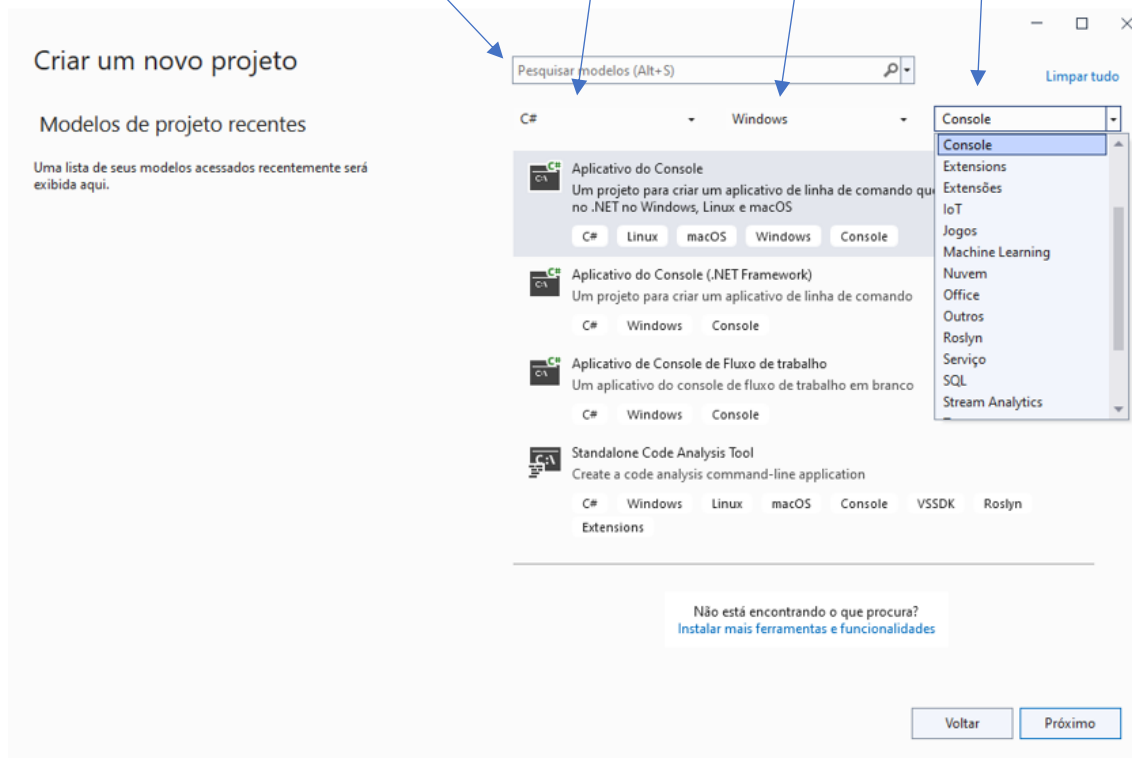
Criando um programa de console C# no Visual Studio 2022

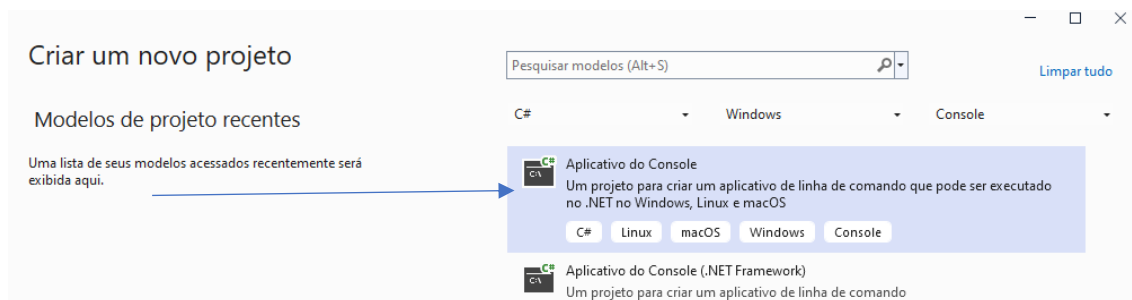
Abra o Visual Studio 2022. Na tela inicial, clique em "Criar um novo projeto".



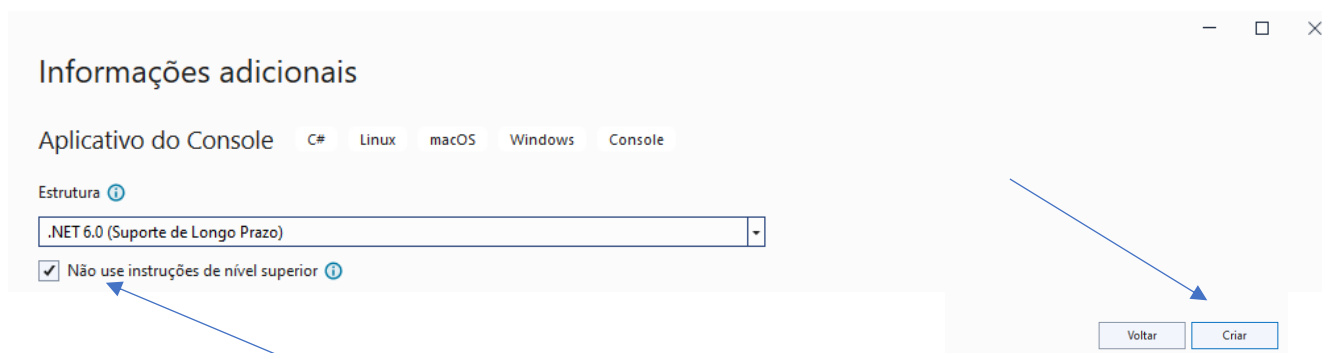
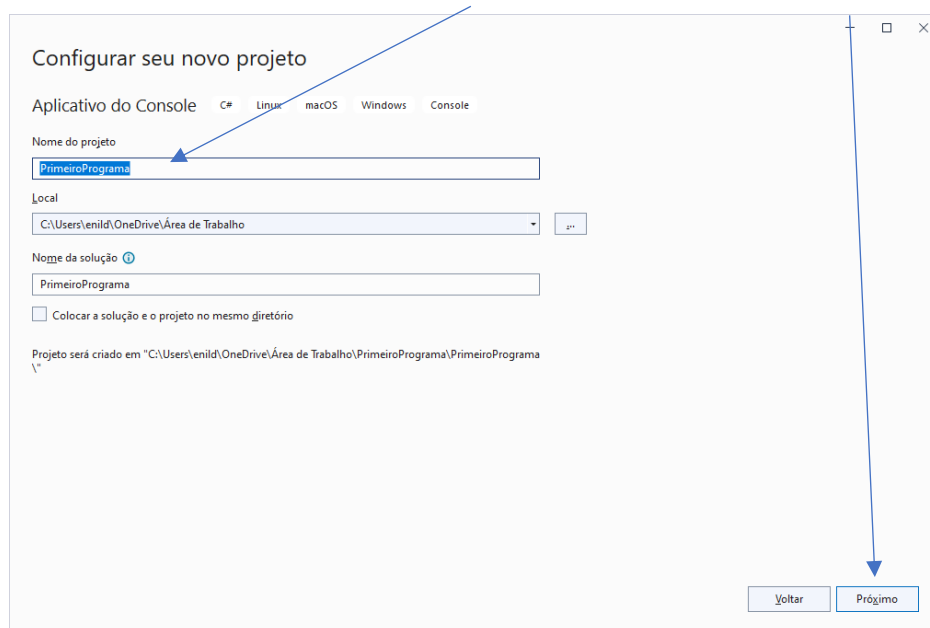
Nas caixas de pesquisas, você pode filtrar a linguagem, a plataforma, e tipo de projeto.

Também podemos usar a caixa de pesquisa, digite "console" e selecione o template "Aplicativo de console C#".



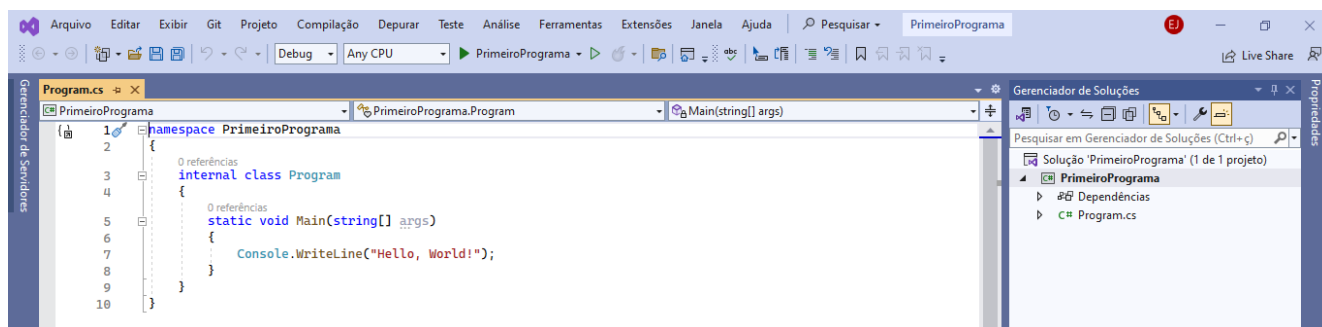


Dê um nome ao seu projeto, por exemplo, "PrimeiroPrograma", e clique em "Próximo".

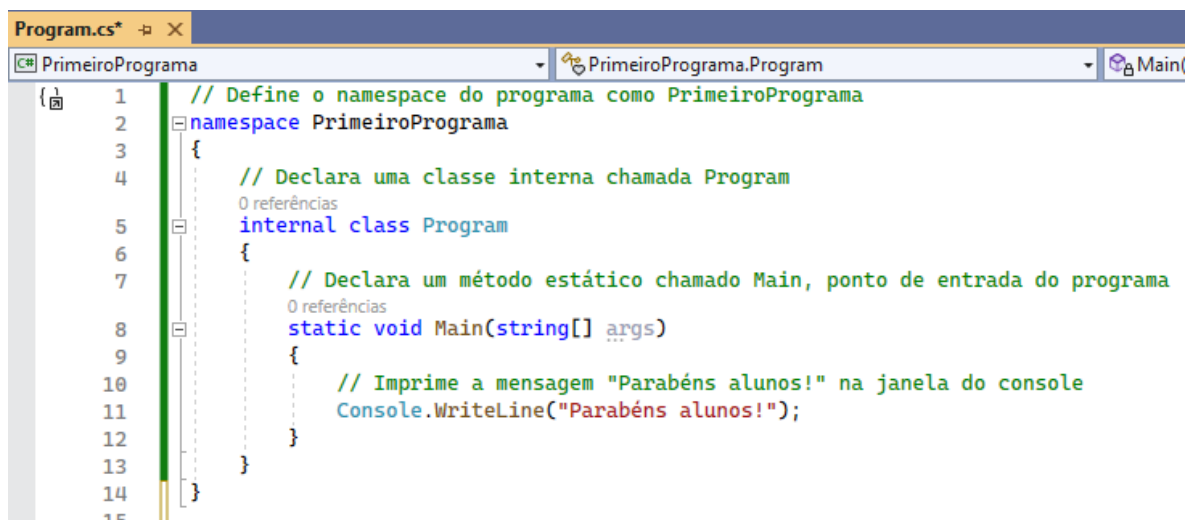


A opção "Não usar instruções de nível superior" no Visual Studio 2022 afeta a maneira como o código C# é escrito e organizado. Desmarcá-la permite que você utilize o recurso de "instruções de nível superior", introduzido no C# 9.

As instruções de nível superior permitem escrever código C# executável diretamente na raiz de um arquivo, sem a necessidade de encapsulá-lo em uma classe ou método. Isso torna a escrita de programas simples mais concisa e legível, especialmente para scripts e pequenos utilitários.

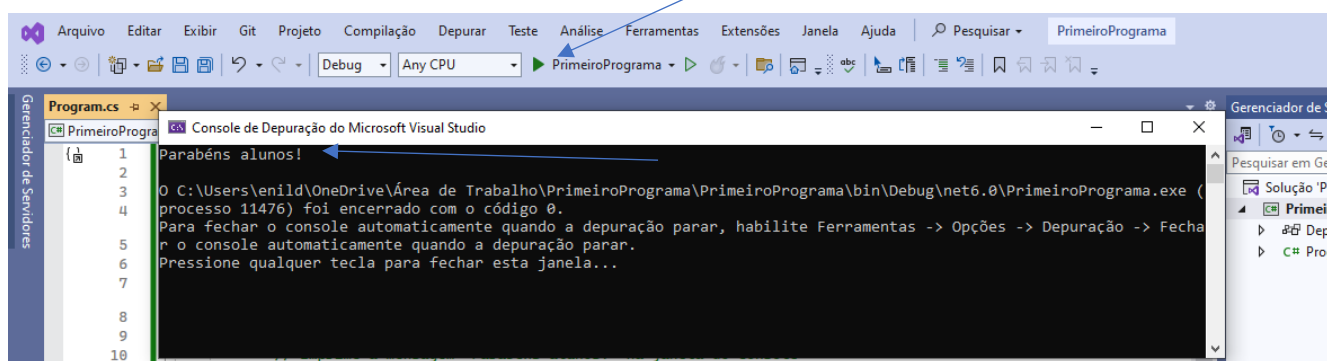


Alterando o texto e comentando o código:

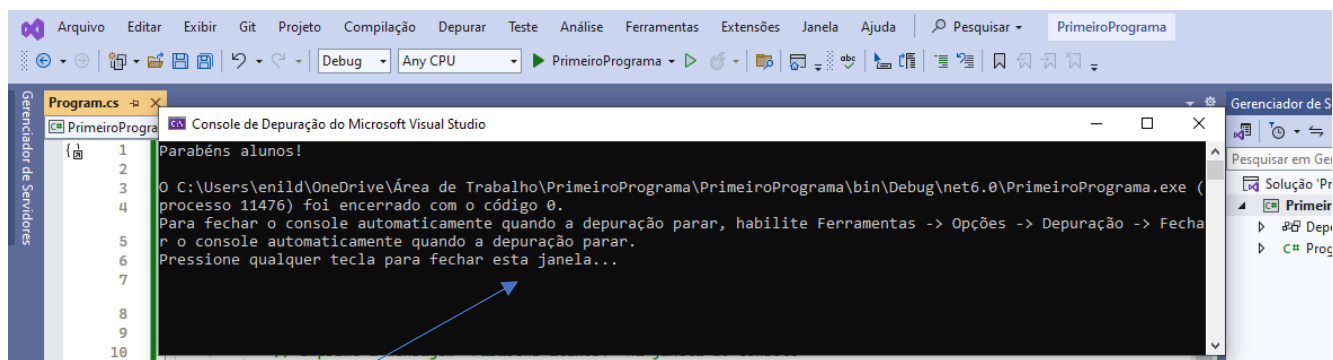


Para executar o programa, pressione a tecla F5 ou clique no botão "Executar" (um triângulo verde) na barra de ferramentas.

Uma janela do console será aberta e exibirá a mensagem "Parabéns alunos!".

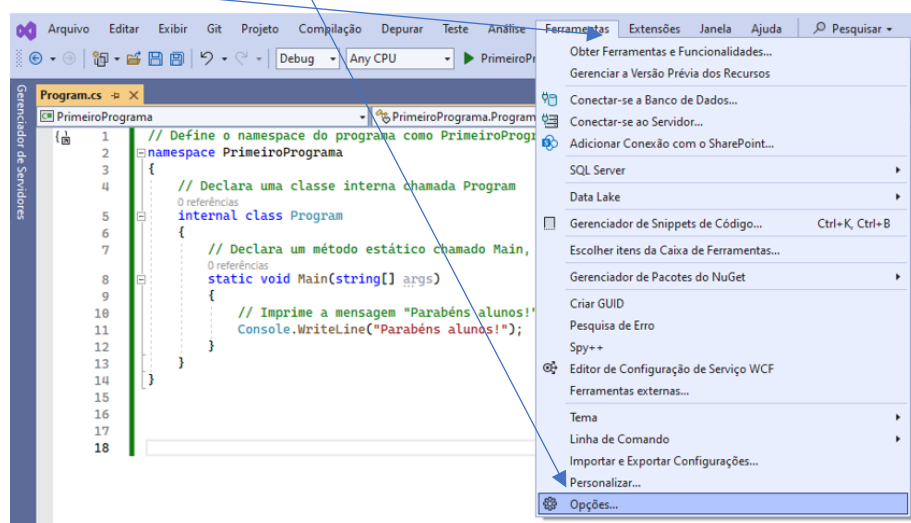


Parabéns! Você acabou de criar seu primeiro programa de console C# no Visual Studio 2022.

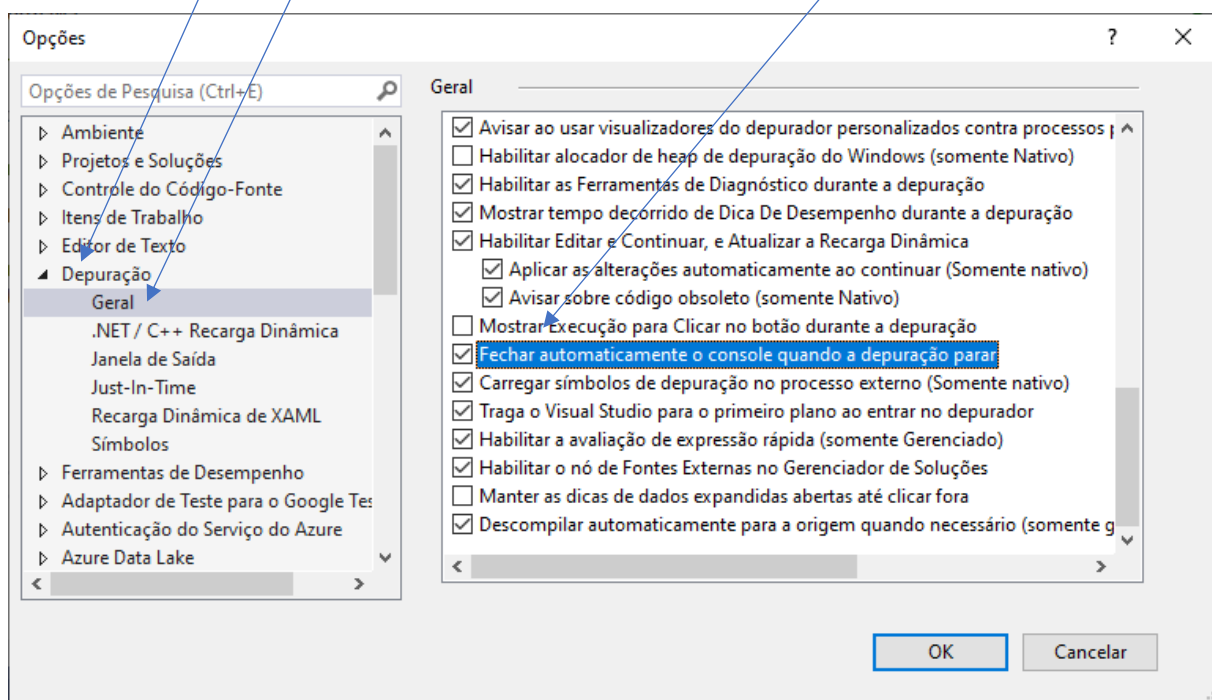


Vamos remover estas informações na janela console.

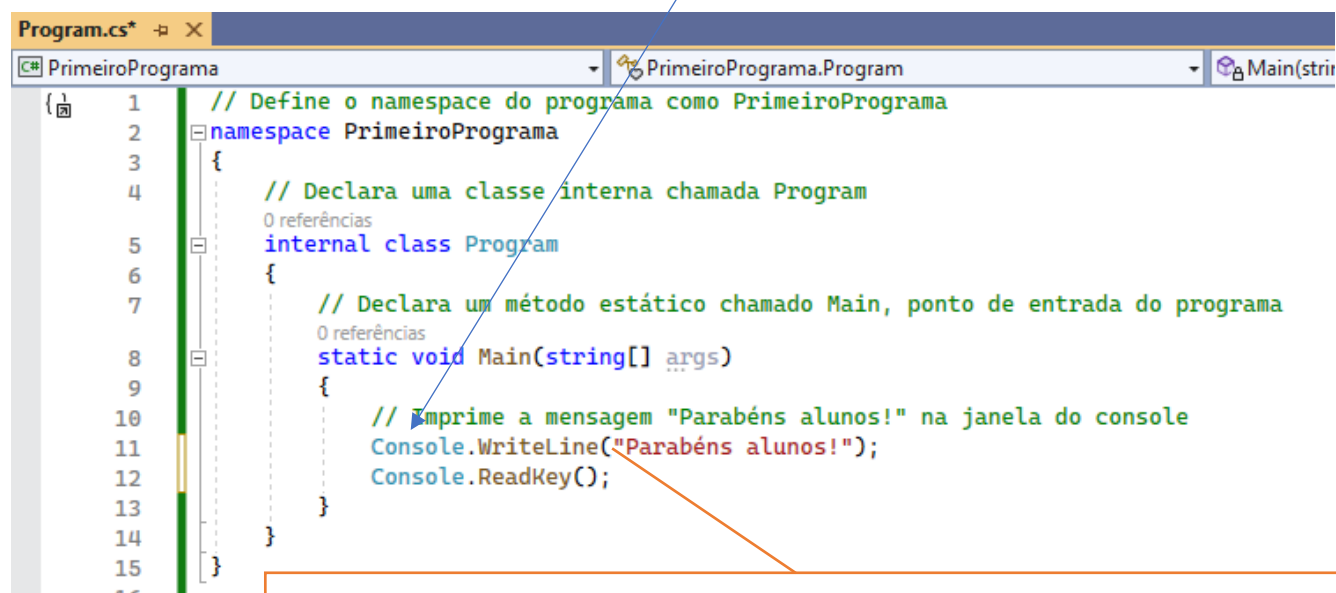
Clique na opção Ferramentas/ Opções...



Agora em Depuração/Geral, marque a opção “Fechar automaticamente o console quando a depuração parar”.



Agora o programa irá executar e fechar automaticamente, mas assim não conseguiríamos ver nada! Para que a tela não feche precisamos acrescentar um código para que o programa fique aguardando uma ação.



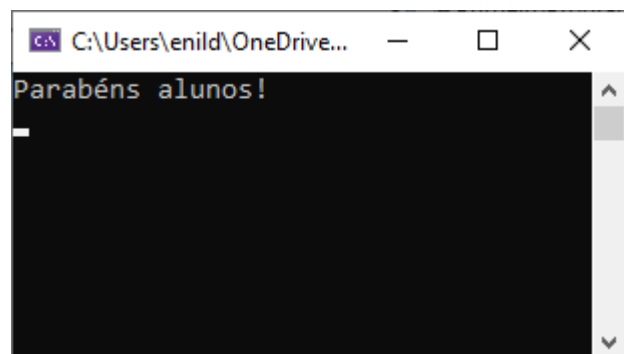
```
1 // Define o namespace do programa como PrimeiroPrograma
2 namespace PrimeiroPrograma
3 {
4     // Declara uma classe interna chamada Program
5     internal class Program
6     {
7         // Declara um método estático chamado Main, ponto de entrada do programa
8         static void Main(string[] args)
9         {
10             // Imprime a mensagem "Parabéns alunos!" na janela do console
11             Console.WriteLine("Parabéns alunos!");
12             Console.ReadKey();
13         }
14     }
15 }
```

Console.ReadKey(); é um método usado em linguagens de programação como C# para obter a entrada do usuário no console.

Aguarda a entrada do usuário: Quando este método é chamado, o programa pausa sua execução e espera que o usuário pressione uma tecla do teclado.

Lê a tecla: depois que uma tecla é pressionada, Console.ReadKey() lê as informações sobre o pressionamento a tecla.

Teste



Dicas:

Você pode explorar e modificar o código para experimentar diferentes funcionalidades do C#.

Utilize a documentação oficial da Microsoft para C# para aprender mais sobre a linguagem:

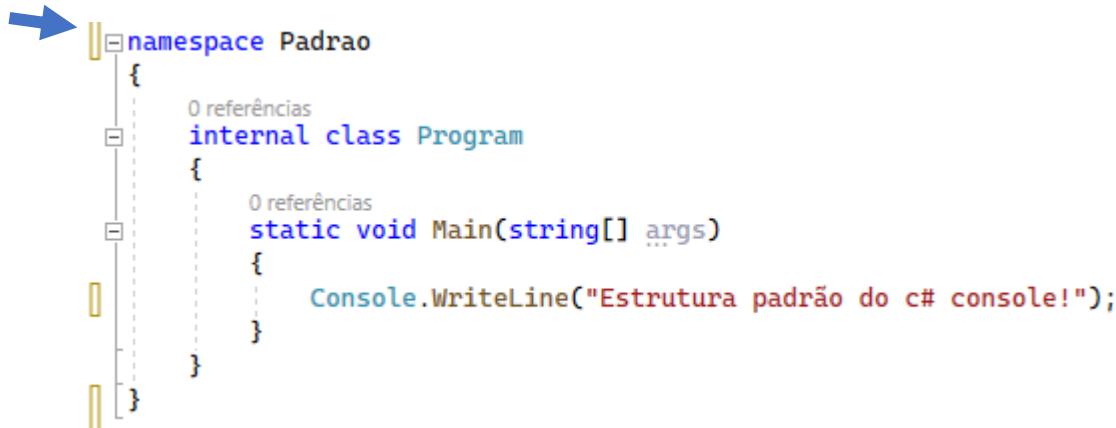
<https://learn.microsoft.com/en-us/dotnet/csharp/>

Explore tutoriais online e cursos para aprofundar seus conhecimentos em programação C#.

Estrutura padrão em c# console

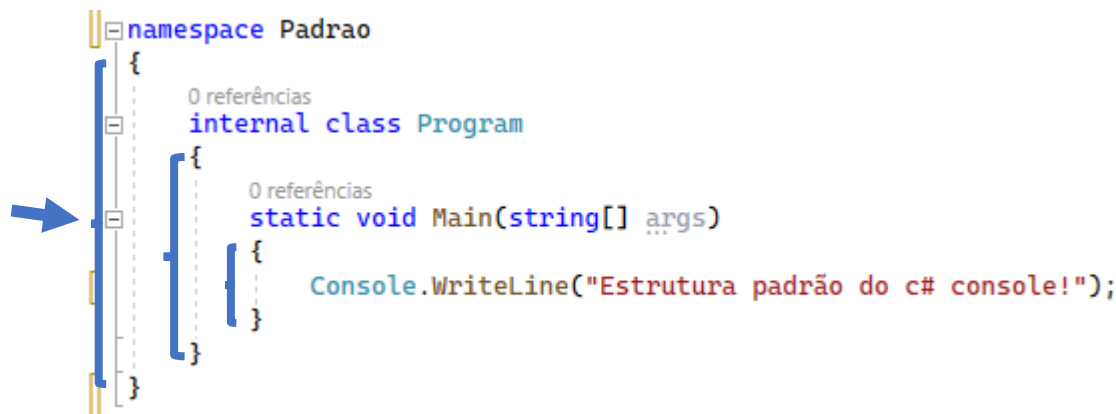
Namespaces

Namespaces são como pastas em um sistema operacional, mas para código. Eles agrupam elementos de código relacionados, como classes, interfaces e structs, sob um nome comum. Isso promove a organização do código em projetos grandes e complexos.



Chaves {}

Em C#, as chaves `{ }` têm vários usos, dependendo do contexto.



Chaves são mais comumente usadas para **definir blocos de código**. Esses blocos agrupam várias instruções que devem ser executadas como uma unidade.

- **Corpos do método:** o código dentro de um método é encapsulado entre chaves, definindo as instruções que o método executa.
- **Instruções condicionais (if, else if, else):** O código a ser executado com base na condição é colocado entre chaves após a instrução `if`, ou `. else ifelse`
- **Instruções de loop (for, while):** O código que é executado repetidamente dentro do loop é definido entre chaves.

Tipos de acesso



```
namespace Padrao
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Estrutura padrão do c# console!");
        }
    }
}
```

O diagrama ilustra a estrutura de um projeto em C#. No topo, há um namespace chamado 'Padrao'. Dentro dele, há uma classe chamada 'Program' declarada com a palavra-chave 'internal'. Dentro da classe 'Program', há um método estático chamado 'Main' que recebe um array de strings como parâmetro. O corpo do método 'Main' contém uma chamada para 'Console.WriteLine' com a mensagem 'Estrutura padrão do c# console!'. Uma seta azul aponta para a palavra-chave 'internal' na declaração da classe 'Program'.

Em C#, `internal` é uma palavra-chave usada para controlar a acessibilidade de tipos (classes, structs, interfaces etc.) e membros de tipos (métodos, propriedades etc.).

Escopo de `internal`

Elementos declarados como `internal` são acessíveis apenas dentro do mesmo assembly.

Um assembly é a unidade básica de implantação e gerenciamento de código em C#. Ele tipicamente contém o código compilado (DLL) e outros recursos necessários para a execução.

Imagine seu código organizado em diferentes arquivos (.cs). Se você declara algo como `internal`, ele pode ser acessado por qualquer outro código dentro do mesmo assembly, independente do arquivo (.cs) onde esteja localizado.

Vantagens de usar `internal`

- **Encapsulamento:** `internal` ajuda a encapsular a implementação de detalhes internos de um assembly, ocultando-os do código externo. Isso melhora a modularidade e facilita a manutenção do código.
- **Evita acoplamento excessivo:** Limita a dependência de outras partes do programa na implementação específica. Isso torna o código mais flexível e evita problemas quando partes do código mudam.

Cenários de uso de `internal`

- **Classes utilitárias internas:** Classes que fornecem funcionalidades usadas apenas por outras partes do mesmo assembly podem ser declaradas como `internal`.
- **Métodos auxiliares internos:** Métodos que auxiliam na implementação de funcionalidades públicas, mas não precisam ser expostos externamente, podem ser `internal`.
- **Componentes privados:** Em desenvolvimento baseado em componentes, classes que fazem parte de um componente específico e não precisam ser acessíveis de fora desse componente podem ser marcadas como `internal`.

Quando não usar internal

Classes e métodos que devem ser acessíveis de outros assemblies: Se a funcionalidade precisa ser compartilhada por diferentes partes do programa, use public em vez de internal.

Resumo

Internal é uma ferramenta útil para promover a modularidade e ocultar detalhes de implementação dentro de um assembly em C#. Use-o para classes e membros que não precisam ser acessíveis de fora do assembly específico.

ASSEMBLY

No contexto da programação, um assembly tem significados diferentes dependendo do idioma ou ambiente específico.

1. Montagem em .NET (C#, VB.NET etc.)

No mundo de (.NET Framework e .NET Core), um assembly é a unidade fundamental de implantação e controle de versão do código. É essencialmente um contêiner que empacota um conjunto de recursos relacionados necessários para o funcionamento de um aplicativo ou biblioteca. Esses recursos normalmente incluem:

- Código compilado (geralmente em um formato como CIL – Common Intermediate Language).
- Metadados: Estas informações descrevem os tipos, métodos, e recursos dentro da assembleia. É crucial para o CLR (Common Language Runtime) compreender e gerenciar o código.
- Manifesto: Este arquivo especifica detalhes sobre a montagem, como seu nome, versão, dependências de outros assemblies, e requisitos de segurança.
- Recursos opcionais: As montagens também podem conter recursos adicionais, como imagens, arquivos de texto, ou dados de configuração.

Pontos principais sobre montagens em .NET

- Implantação: Os assemblies são as unidades principais que você implanta e compartilha aplicações NET. Eles podem ser distribuídos como DLLs (Dynamic-Link Libraries) ou EXEs (Executables).
- Versionamento: Os assemblies têm versões associadas a eles, permitindo a execução lado a lado de diferentes versões da mesma montagem. Isso ajuda a gerenciar problemas de compatibilidade.
- Dependências: As montagens podem depender de outras montagens, formando uma hierarquia. O CLR localiza e carrega os assemblies necessários para a execução de um aplicativo.

2. Assembly em linguagens de baixo nível (linguagem Assembly):

Na programação em linguagem assembly, assembly refere-se ao processo de tradução de instruções escritas em código assembly (instruções mnemônicas que representam código de máquina) em código de máquina que o processador pode entender diretamente. Essa tradução normalmente é feita por um programa chamado assembler.

Pontos principais sobre Assembly em linguagem Assembly

- **Programação de baixo nível:** A linguagem assembly fornece uma representação do código de máquina mais legível por humanos em comparação com instruções binárias brutas.
- **Específico de hardware:** As instruções do código assembly são específicas para a arquitetura do processador alvo (por exemplo, ex., x86, BRAÇO).
- **Otimização de performance:** A linguagem assembly às vezes pode oferecer um controle mais preciso sobre o hardware, permitindo possíveis otimizações de desempenho em cenários específicos.


Resumo

Um assembly é uma unidade de código empacotada, metadados, recursos, e informações de manifesto para implantação e controle de versão.

Em linguagem assembly, assembly refere-se ao processo de tradução do código assembly (instruções mnemônicas) para o código de máquina do processador.

O significado específico de “montagem” depende do contexto em que você está trabalhando. Na maioria dos casos relacionados ao desenvolvimento de aplicativos modernos. O significado .NET é mais relevante.

Classe



```
namespace Padrao
{
    0 referências
    internal class Program
    {
        0 referências
        static void Main(string[] args)
        {
            Console.WriteLine("Estrutura padrão do c# console!");
        }
    }
}
```

Em C#, uma classe atua como um modelo ou modelo para a criação de objetos. Define as propriedades (dados) e funcionalidades (métodos) que os objetos daquela classe possuirão.

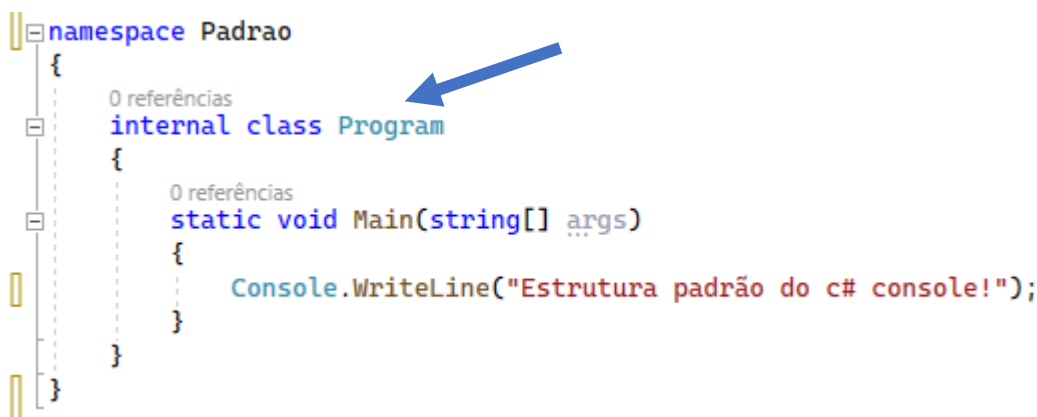
Pense em uma classe como um protótipo que especifica as características e os comportamentos de um determinado tipo de coisa. Por exemplo, você poderia ter uma Carclasse que defina os atributos (cor, marca, modelo) e ações (acelerar, frear, virar) que todos os objetos carro teriam em comum.

Componentes de uma classe

Estes são os blocos de construção que definem uma classe. Eles podem ser de dois tipos principais:

- **Campos (ou atributos):** representam as propriedades ou dados que um objeto contém. Eles podem armazenar valores como a cor (string cor) ou a velocidade () de um carro int Velocidade.
- **Métodos:** definem as ações que os objetos podem executar. Eles normalmente operam nos dados (campos) do objeto e podem retornar um valor ou executar uma operação. Uma Carclasse pode ter métodos como Acelerador() ou VirarEsquerda().

Classe Program

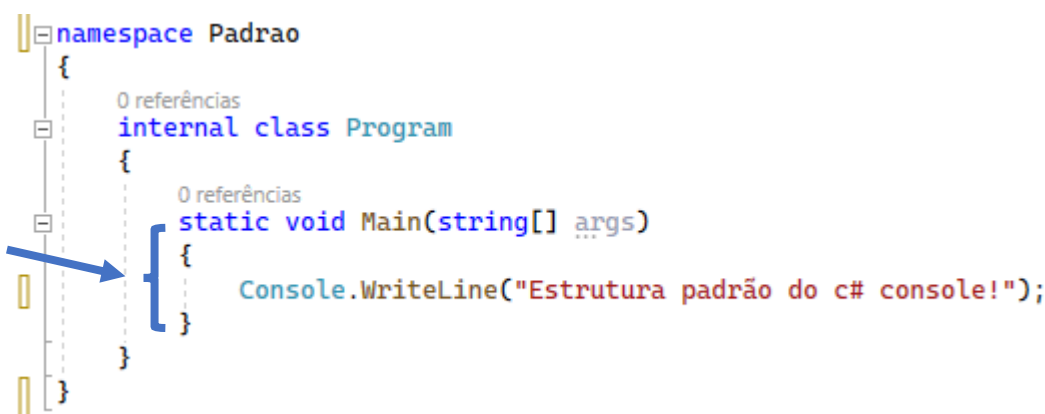


```
namespace Padrao
{
    0 referências
    internal class Program
    {
        0 referências
        static void Main(string[] args)
        {
            Console.WriteLine("Estrutura padrão do c# console!");
        }
    }
}
```

Embora o terceiro parâmetro da declaração seja o nome, ou seja, 'Program' é o nome! Poderia ser qualquer outro nome!

A Program classe em C# desempenha uma função variável dependendo do tipo de aplicativo que você está construindo. Ponto de entrada para aplicativos de console (abordagem tradicional). Em aplicativos de console tradicionais, a Program classe abriga o Main método, que serve como ponto de entrada do programa. O CLR (Common Language Runtime) executa o código Main quando o aplicativo é iniciado.

MÉTODO



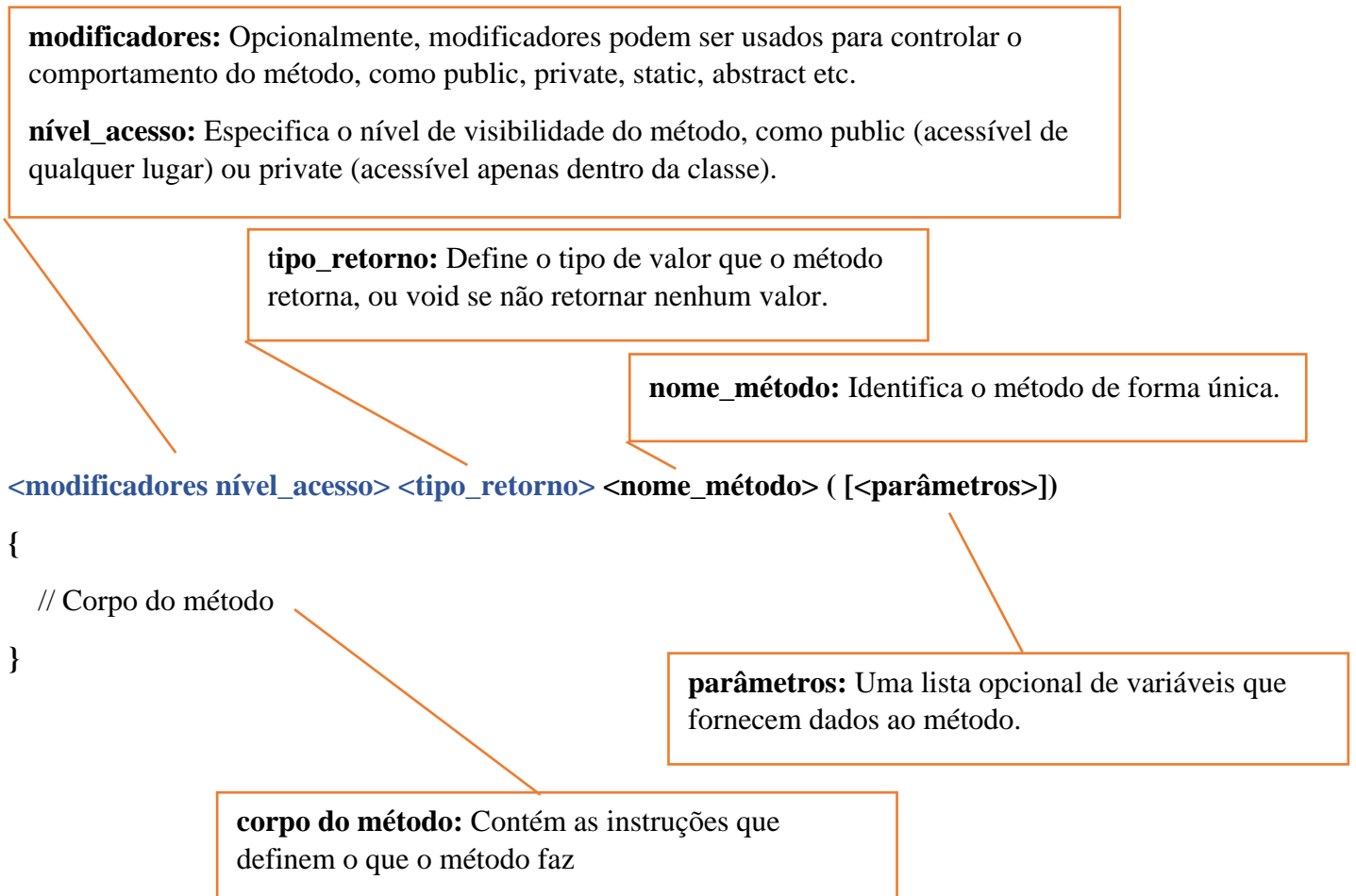
```
namespace Padrao
{
    0 referências
    internal class Program
    {
        0 referências
        static void Main(string[] args)
        {
            Console.WriteLine("Estrutura padrão do c# console!");
        }
    }
}
```

Em C#, um método é um bloco de código reutilizável que executa uma tarefa específica. Ele é semelhante a uma função em outras linguagens de programação, mas com algumas diferenças importantes.

Características principais dos métodos em C#

- **Modularidade:** Os métodos promovem a modularidade do código, dividindo-o em unidades menores e mais gerenciáveis.
- **Reutilização:** Os métodos podem ser reutilizados em diferentes partes do programa, evitando duplicação de código.
- **Organização:** Os métodos ajudam a organizar o código de forma clara e concisa, tornando-o mais fácil de ler, entender e manter.
- **Encapsulamento:** Os métodos podem encapsular dados e lógica, protegendo-os de acesso externo indesejado.

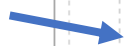
Estrutura básica de um método em C#



Exemplo de método em C#:

Neste exemplo, o método `CalcularSoma` recebe dois números inteiros como parâmetros (a e b), soma os valores e retorna o resultado como um número inteiro.


```
namespace Padrao
{
    0 referências
    internal class Program
    {
        0 referências
        static void Main(string[] args)
        {
            Console.WriteLine("Estrutura padrão do c# console!");
            int resultado = CalcularSoma(5, 3);
        }
        1 referência
        static int CalcularSoma(int a, int b)
        {
            int soma = a + b;
            return soma;
        }
    }
}
```



Chamando um método

Para chamar um método, basta usar seu nome seguido dos parâmetros entre parênteses.

```
namespace Padrao
{
    0 referências
    internal class Program
    {
        0 referências
        static void Main(string[] args)
        {
            Console.WriteLine("Estrutura padrão do c# console!");
            int resultado = CalcularSoma(5, 3);
        }
        1 referência
        static int CalcularSoma(int a, int b)
        {
            int soma = a + b;
            return soma;
        }
    }
}
```



Tipos de métodos

Em C#, existem diferentes tipos de métodos, cada um com características e propósitos específicos:

- **Métodos de instância:** Associados a objetos específicos e podem acessar e modificar seus dados.
- **Métodos estáticos:** Não estão associados a objetos e pertencem à classe em si.
- **Métodos abstratos:** Definem uma funcionalidade que deve ser implementada por classes derivadas.
- **Métodos virtuais:** Permitem que subclasses reimplementem sua lógica.

Benefícios do uso de métodos

- **Código mais limpo e organizado:** Divide o código em unidades menores e fáceis de gerenciar.
- **Melhor reutilização de código:** Evita duplicação de código e promove a modularidade.
- **Maior legibilidade e manutenibilidade:** Torna o código mais fácil de entender e modificar.
- **Encapsulamento aprimorado:** Protege dados e lógica de acesso externo indesejado.
- **Maior flexibilidade:** Permite a criação de código mais flexível e reutilizável.

Recursos para aprender mais

Documentação oficial da Microsoft sobre métodos em C#: <https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/methods>

Método Main()

Em C#, o Main() método tem um significado especial como ponto de entrada para aplicativos de console. Serve como ponto de partida para a execução do programa, onde são iniciadas as principais lógicas e funcionalidades.

```
namespace Padrao
{
    0 referências
    internal class Program
    {
        0 referências
        static void Main(string[] args)
        {
            Console.WriteLine("Estrutura padrão do c# console!");
        }
    }
}
```

Principais características de Main()

- **Ponto de entrada:** Main() é o método designado onde começa a execução do programa. É chamado pelo Common Language Runtime (CLR) quando o programa é iniciado.
- **Método Estático:** Main() é sempre declarado como um método estático, o que significa que está associado à própria classe e não a uma instância específica do objeto.
- **Tipo de retorno void:** Main() não retorna nenhum valor. Em vez disso, ele controla o fluxo e a execução do programa.
- **Parâmetros padrão:** Main() normalmente possui um único parâmetro do tipo string[] denominado args. Esta matriz contém argumentos de linha de comando passados quando o programa é executado.

Estrutura de Main()

Como Main() é o ponto de entrada, você não o chama explicitamente. O CLR o invoca automaticamente quando o programa é iniciado.

Papel de Main()

- **Inicializar aplicativo:** Main() geralmente é usado para inicializar o estado do aplicativo, como configurar variáveis, conectar-se a recursos ou carregar dados de configuração.
- **Executar lógica central:** é onde reside a lógica principal do programa, normalmente envolvendo interações do usuário, cálculos, processamento de dados ou quaisquer outras funcionalidades principais.
- **Controlar o fluxo do programa:** Main() controla o fluxo de execução do programa usando instruções condicionais, loops e chamadas de função.
- **Lidar com argumentos de linha de comando:** se o programa aceitar argumentos de linha de comando, Main() poderá acessar e processar esses argumentos usando o args parâmetro.

Significado de Main()

- **Ponto de partida:** Main() marca o início da execução do programa, preparando o cenário para o comportamento da aplicação.
- **Controle Central:** Atua como o hub central para controlar o fluxo, a lógica e as interações do programa.
- **Ponto de personalização:** os desenvolvedores podem personalizar Main() para adaptar a inicialização, o comportamento e a resposta do programa à entrada do usuário ou aos argumentos da linha de comando.

Em resumo, o Main() método desempenha um papel crucial em aplicações de console C#, servindo como ponto de entrada, centro de controle e ponto de customização para a execução do programa.

Classe Console

Em C#, a Console classe, encontrada no System namespace, fornece uma maneira para seus aplicativos de console interagirem com a linha de comando (terminal baseado em texto).

A Console classe é estática, o que significa que você não precisa criar uma instância dela para usar seus métodos.

É essencial para entrada e saída em aplicativos de console.

Métodos como "Console.BackgroundColor", "Console.ForegroundColor" podem ser usados para alterar a aparência da janela do console (a disponibilidade pode variar dependendo da plataforma).

Métodos da classe Console em C#

A classe Console em C# oferece diversos métodos para interação com a linha de comando.

Texto saída

- **Console.Write(string text):** Escreve o texto especificado no console sem avançar para a próxima linha.

- **Console.WriteLine(string text):** Escreve o texto especificado no console e avança para a próxima linha.
- **Console.WriteLine(string format, object arg0):** Escreve o texto formatado com o argumento especificado. Sobrecargas disponíveis para múltiplos argumentos.
- **Console.WriteLine(string format, params object[] args):** Escreve o texto formatado com os argumentos fornecidos como um array.

Leitura entrada

- **Console.ReadLine():** Lê uma linha de texto digitada pelo usuário e retorna como string.
- **Console.ReadKey(bool intercept):** Lê uma única tecla pressionada pelo usuário (com ou sem interceptação de teclas de atalho do sistema). Retorna um objeto ConsoleKeyInfo.
- **Console.Read():** Lê um único caractere (código ASCII) pressionado pelo usuário e retorna um inteiro.

Formatação de Texto

- **Console.SetOut(TextWriter newOut):** Define um novo fluxo de saída para o console.
- **Console.GetOut():** Retorna o fluxo de saída atual do console.
- **Console.SetError(TextWriter newError):** Define um novo fluxo de erro para o console.
- **Console.GetError():** Retorna o fluxo de erro atual do console.

Controle do Console

- **Console.Clear():** Limpa o texto exibido no console.
- **Console.Beep(int frequency, int duration):** Emite um bipe com a frequência e a duração especificadas.
- **Console.Title = string title:** Define o título da janela do console.
- **Console.CursorLeft = int left:** Define a posição do cursor na coluna especificada.
- **Console.CursorTop = int top:** Define a posição do cursor na linha especificada.
- **Console.CursorVisible = bool visible:** Controla a visibilidade do cursor do console.

Cores e Formatação

Console.ForegroundColor = ConsoleColor color: Define a cor da frente do texto do console.

Console.BackgroundColor = ConsoleColor color: Define a cor do fundo do texto do console.

Console.ResetColor(): Restaura as cores padrão do console.

Outras Propriedades e Métodos

Console.In: Um objeto TextReader que representa a entrada padrão do console.

Console.Error: Um objeto TextWriter que representa a saída de erro do console.

Console.OpenStandardInput(): Abre o fluxo de entrada padrão do console.

Console.OpenStandardOutput(): Abre o fluxo de saída padrão do console.

Console.OpenStandardError(): Abre o fluxo de erro padrão do console.

Observações

A disponibilidade de alguns métodos de formatação e cores pode variar dependendo do sistema operacional em que o programa é executado.

Recursos Adicionais

Para mais detalhes e exemplos de uso, consulte a documentação oficial da Microsoft:

<https://learn.microsoft.com/en-us/dotnet/api/system.console?view=net-8.0>

Guia de Programação em C#: Classes e Estruturas: <https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/methods>

Para interfaces gráficas de usuário (GUIs), explore bibliotecas como Windows Forms ou WPF em C#.

Para operações avançadas de entrada/saída, considere usar fluxos e arquivos.

Variáveis, Vetores e Matrizes em C#

Em C#, as variáveis, vetores e matrizes são elementos fundamentais para armazenar e organizar dados durante a execução de um programa. Cada um possui características e usos específicos, sendo essenciais para construir aplicações eficientes e robustas.

Variáveis

As variáveis, “armazenando dados individuais”, servem como contêineres para armazenar valores singulares de diferentes tipos de dados, como números, textos, caracteres booleanos e objetos. Elas são declaradas utilizando a palavra-chave `var` ou o tipo de dados específico, seguido do nome da variável e de um valor inicial (opcional).

Declaração de variáveis

```
int numero = 10; // Variável do tipo inteiro com valor inicial 10
string nome = "Fulano"; // Variável do tipo string com valor inicial "Fulano"
bool estaAtivo = true; // Variável do tipo booleano com valor inicial true
```

As variáveis permitem que os programadores manipulem dados durante a execução do programa, utilizando operações matemáticas, atribuições, comparações e outros comandos.

Vetores

Vetores, “coleções ordenadas de elementos”, também conhecidos como arrays unidimensionais, servem para armazenar coleções ordenadas de elementos do mesmo tipo. Eles são declarados utilizando a palavra-chave `int[]`, `string[]`, `bool[]`, ou o tipo de dados desejado seguido de colchetes `[]` e o tamanho do vetor entre colchetes.

Declaração de vetores

```
int[] notas = new int[5]; // Vetor de 5 elementos do tipo inteiro
string[] nomes = new string[] { "Ana", "João", "Maria" }; // Vetor inicializado com valores
bool[] estaPresente = { true, false, true }; // Vetor de booleanos inicializado
```

Acessando elementos do vetor

```
Console.WriteLine($"Primeira nota: {notas[0]}"); // Acessando o primeiro elemento
Console.WriteLine($"Nome do aluno 2: {nomes[1]}"); // Acessando o segundo elemento
```

Os elementos de um vetor são acessados por meio de um índice entre colchetes, iniciando em 0. Vetores facilitam a manipulação de grupos de dados relacionados, como notas de alunos, nomes de produtos ou valores de temperatura.

Matrizes

Matrizes, “armazenando dados bidimensionais”, também conhecidas como arrays multidimensionais, são estruturas que armazenam dados em uma grade bidimensional, com linhas e colunas. Elas são declaradas utilizando a palavra-chave `int[,]`, `string[,]`, `bool[,]`, ou o tipo de dados desejado seguido de colchetes `[]` com o número de linhas e colunas entre colchetes separados por vírgulas.

Declaração de matrizes

```
int[,] tabelaNotas = new int[3, 4]; // Matriz de 3 linhas e 4 colunas (12 elementos)
string[,] nomesAlunos = { { "Ana", "João" }, { "Maria", "Pedro" } }; // Matriz inicializada com valores
bool[,] estaPresenteAula = { { true, false }, { false, true } }; // Matriz de booleanos inicializada
```

Acessando elementos da matriz

```
Console.WriteLine($"Nota do aluno 1 na disciplina 2: {tabelaNotas[0, 1]}");
Console.WriteLine($"Nome do aluno na linha 1, coluna 2: {nomesAlunos[1, 1]}");
Console.WriteLine($"Presença do aluno 2 na aula 1: {estaPresenteAula[1, 0]}");
```

Os elementos de uma matriz são acessados por meio de índices entre colchetes, com o primeiro índice indicando a linha e o segundo índice indicando a coluna. Matrizes são úteis para representar dados organizados em tabelas, como notas de alunos em diferentes disciplinas, horários de aulas ou mapas de jogo.

Tipos de operadores em C#

Em C#, operadores são símbolos que representam operações a serem executadas em um ou mais operandos. Eles são os blocos de construção para criar expressões e realizar cálculos. C# oferece uma ampla variedade de operadores categorizados em diferentes tipos com base em sua funcionalidade.

Operadores aritméticos

`+`, `-`, `*`, `/`, `%`, `++`, `--` : Execute operações aritméticas básicas como adição, subtração, multiplicação, divisão, módulo, incremento e decremento.

Exemplos:

```
int soma = 10 + 5; // soma = 15 Soma(+)  
  
int diferenca = 20 - 12; // diferenca = 8 Subtração(-)  
  
int produto = 3 * 6; // produto = 18 Multiplicação(*)  
  
int quociente = 18 / 4; // quociente = 4 Divisão(/)  
  
int resto = 17 % 3; // resto = 2 Resto da divisão(%)  
  
int contador = 0;  
contador++; // contador = 1 Incremento(++)  
  
int numero = 10;  
numero--; // numero = 9 Decremento(--)
```

Operadores de comparação

`==`, `!=`, `<`, `>`, `<=`, `>=`: compare valores de igualdade, desigualdade, menor que, maior que, menor ou igual a e maior ou igual a.

Exemplos:

```

int x = 10;
int y = 15;
bool igualdade = x == y; // igualdade = false Igual(==)

string nome1 = "Pedro";
string nome2 = "Pedro";
bool nomesIguais = nome1 == nome2; // nomesIguais = true

int idade = 25;
bool maiorDeIdade = idade != 18; // maiorDeIdade = true Desigual(!=)

char letra = 'a';
bool diferenteDeB = letra != 'b'; // diferenteDeB = true

int nota1 = 90;
int nota2 = 85;
bool nota1Melhor = nota1 < nota2; // nota1Melhor = false Menor que(<)

decimal preco1 = 12.50m;
decimal preco2 = 10.99m;
bool preco1MaisCaro = preco1 < preco2; // preco1MaisCaro = false

int distancia1 = 200;
int distancia2 = 150;
bool caminhoMaisCurto = distancia1 > distancia2; // caminhoMaisCurto = true Maior que(>)

DateTime data1 = new DateTime(2023, 12, 31);
DateTime data2 = new DateTime(2024, 1, 1);
bool data2MaisRecente = data1 > data2; // data2MaisRecente = false

int pontosTime1 = 70;
int pontosTime2 = 70;
bool empate = pontosTime1 <= pontosTime2; // empate = true Menor que ou igual(<=)

int temperatura = 22;
bool temperaturaAmena = temperatura <= 25; // temperaturaAmena = true

int nivelAcesso = 3;
int nivelMinimo = 2;
bool acessoPermitido = nivelAcesso >= nivelMinimo; // acessoPermitido = true Maior que ou igual(>=)

int paginaAtual = 5;
int ultimaPagina = 5;
bool chegouNaUltimaPagina = paginaAtual >= ultimaPagina; // chegouNaUltimaPagina = true

```

Operadores lógicos

&&(AND), ||(OR), !(NOT): Combine expressões booleanas usando operadores lógicos para representar operações AND, OR e NOT.

Exemplos:

```
int idade = 18;
bool carteiraDeMotorista = true;
bool podeDirigir = idade >= 18 && carteiraDeMotorista; // podeDirigir = true E(&&)

string nome = "Ana";
int nota = 90;
bool boaAluna = nome == "Ana" && nota >= 90; // boaAluna = true

bool estaChovendo = true;
bool temGuardaChuva = false;
bool precisaDeGuardaChuva = estaChovendo || temGuardaChuva; // precisaDeGuardaChuva = true Ou(||)

int diaSemana = 1; // Segunda-feira
bool finalDeSemana = diaSemana == 6 || diaSemana == 7; // finalDeSemana = false

bool estaLigado = false;
bool estaDesligado = !estaLigado; // estaDesligado = true Negação(!)

int temperatura = 30;
bool estaFrio = !(temperatura >= 25); // estaFrio = false
```

Operadores de Atribuição

=, +=, -=, *=, /=, %=: Atribua valores a variáveis e execute operações de atribuição combinadas, como adicionar, subtrair, multiplicar, dividir e calcular o módulo durante a atribuição.

Exemplos:

```
int numero = 10; // Atribuição simples(=)
string nome = "João";

int contador = 0;
contador += 5; // Atribuição com adição(+=)

decimal preco = 12.90m;
preco += 2.50m; // preco = 15.40m

int pontuacao = 100;
pontuacao -= 20; // pontuacao = 80 Atribuição com subtração(-=)

double distancia = 150.50;
distancia -= 35.25; // distancia = 115.25
```



```

int quantidade = 3;
quantidade *= 2; // quantidade = 6  Atribuição com multiplicação(=)

float area = 5.2f;
area *= 3.5f; // area = 18.2f

int total = 48;
total /= 4; // total = 12  Atribuição com divisão(/=)

int horasTrabalhadas = 20;
horasTrabalhadas /= 8; // horasTrabalhadas = 2.5

int dividendo = 17;
dividendo %= 3; // dividendo = 2  Atribuição com resto da divisão(%)

int numeroDePessoas = 15;
numeroDePessoas %= 5; // numeroDePessoas = 0

```

Operadores bit a bit

&, |, ^, ~, <<, >>: Execute operações bit a bit em representações binárias de valores, incluindo AND, OR, XOR, NOT, deslocamento para a esquerda e deslocamento para a direita.

Exemplos:

```

int num1 = 10101100; // 0b10101100
int num2 = 10100111; // 0b10100111
int resultadoE = num1 & num2; // 0b10100100 bitwise(&): Realiza a operação AND bitwise entre
// dois valores binários.Retorna 1 somente se ambos os bits correspondentes
Console.WriteLine($"num1 & num2: {resultadoE:b}"); // Output: 10100100

int num3 = 01010100; // 0b01010100
int num4 = 11110011; // 0b11110011
int resultadoOu = num3 | num4; // 0b11110111 bitwise(|): Realiza a operação OR bitwise entre dois
// valores binários.Retorna 1 se pelo menos um dos bits correspondentes
Console.WriteLine($"num3 | num4: {resultadoOu:b}"); // Output: 11110111

int num5 = 10101100; // 0b10101100
int num6 = 10100111; // 0b10100111
int resultadoXor = num5 ^ num6; // 0b00001011 bitwise(^): Realiza a operação XOR bitwise entre dois valores
// binários.Retorna 1 se os bits correspondentes forem diferentes
Console.WriteLine($"num5 ^ num6: {resultadoXor:b}"); // Output: 00001011

int num = 01010100; // 0b01010100
int resultadoNegacao = ~num; // 10101011  Negação bitwise(~): Inverte todos os bits de um valor binário.

int num7 = 00110101; // 0b00110101
int deslocamento = 2;
int resultadoEsquerda = num << deslocamento; // 11010100  Deslocamento à esquerda(<<)
Console.WriteLine($"num << {deslocamento}: {resultadoEsquerda:b}"); // Output: 11010100

int num8 = 10110101; // 0b10110101
int deslocamento1 = 2;
int resultadoDireita = num >> deslocamento1; // 00110101  Deslocamento à direita(>>)
Console.WriteLine($"num >> {deslocamento1}: {resultadoDireita:b}"); // Output: 00110101

```

Operadores Condicionais

?: : o operador ternário fornece uma maneira concisa de expressar instruções condicionais dentro de uma expressão.

Sintaxe:

condição: Uma expressão booleana que determina qual valor será retornado.

condição ? valorVerdadeiro : valorFalso

valorFalso: O valor a ser retornado se a condição for false.

valorVerdadeiro: O valor a ser retornado se a condição for true.

Exemplo 1: Verificando a maioridade:

```
int idade = 18;
string mensagem = idade >= 18 ? "Maior de idade" : "Menor de idade";

Console.WriteLine(mensagem); // Output: Maior de idade
```

Neste exemplo, a variável idade é comparada com 18. Se idade for maior ou igual a 18, a string "Maior de idade" é atribuída à variável mensagem. Caso contrário, a string "Menor de idade" é atribuída.

Exemplo 2: Atribuindo notas com base em pontuação:

```
int pontuacao = 75;
string notaLetra;

switch (pontuacao)
{
    case >= 90:
        notaLetra = "A";
        break;
    case >= 80:
        notaLetra = "B";
        break;
    case >= 70:
        notaLetra = "C";
        break;
    case >= 60:
        notaLetra = "D";
        break;
    default:
        notaLetra = "F";
        break;
}
```

Exemplo com Switch Case

O mesmo exemplo com ? :

```
int pontuacao = 75;
string notaLetra;

notaLetra = pontuacao >= 90 ? "A" : pontuacao >= 80 ? "B" : pontuacao >= 70 ? "C" : pontuacao >= 60 ? "D" : "F";
```

Operador de coalescência nulo

?? : recupera com segurança um valor de um operando se não for nulo; caso contrário, ele retornará um valor padrão.

O operador null-coalescing (??) em C# é utilizado para verificar se um operando à esquerda é null. Se for, o operando à direita é retornado. Caso contrário, o operando à esquerda é retornado.

Sintaxe:

operandoEsquerdo ?? operandoDireito

- operandoEsquerdo: O valor que será verificado se é null.
- operandoDireito: O valor a ser retornado se o operandoEsquerdo for null.

Exemplo: Atribuindo valor padrão a uma variável

```
int? numero = null;  
int valorPadrao = numero ?? 0;  
Console.WriteLine($"Valor padrão: {valorPadrao}"); // Output: 0
```

Neste exemplo, a variável numero é null. O operador ?? verifica se numero é null. Como é, o valor 0 é atribuído à variável valorPadrao.

Precedência e associatividade do operador

Os operadores C# têm uma ordem de precedência definida, que determina a ordem na qual as operações são avaliadas em uma expressão. Os operadores também possuem associatividade, que determina a direção na qual as operações são agrupadas quando vários operadores com a mesma precedência estão presentes.

Sobrecarga do Operador

C# permite sobrecarregar operadores, dando-lhe o poder de definir um comportamento personalizado para operadores quando usados com tipos definidos pelo usuário. Isso permite criar operações significativas para seus tipos de dados personalizados.

Os operadores são ferramentas essenciais para manipular dados e realizar cálculos em C#. Compreender os diferentes tipos de operadores, sua precedência e associatividade é crucial para escrever código C# correto e eficiente. A sobrecarga de operadores fornece um mecanismo poderoso para estender a funcionalidade dos operadores para tipos de dados personalizados.

Método Convert em c#

A classe Convert em C# fornece métodos estáticos para converter tipos de dados de um formato para outro. Esses métodos são úteis para converter valores entre tipos numéricos, strings, booleanos, data e hora, e muito mais.

Métodos comumente usados na classe Convert

Para conversão de tipos numéricos:

- **ToInt32(object value):** Converte um valor para um inteiro de 32 bits.
- **ToDouble(object value):** Converte um valor para um número de ponto flutuante de precisão dupla.
- **ToString(object value):** Converte um valor para uma string. Você pode especificar a cultura para formatação (por exemplo, Convert.ToString(valor, CultureInfo.GetCultureInfo("pt-BR"))) para formatação brasileira).

Para conversão de strings:

- **ToBoolean(string value):** Converte uma string para um valor booleano (true ou false).
- **ToInt32(string value):** Converte uma string para um inteiro de 32 bits.
- **TryParse(string value, out T result):** Tenta converter uma string para o tipo especificado (T) e retorna true se a conversão for bem-sucedida. O valor convertido é armazenado na variável result.

Para conversão de data e hora:

- **DateTime(object value):** Converte um valor para um objeto DateTime.
- **ToString(DateTime value):** Converte um objeto DateTime para uma string. Você pode especificar o formato de data e hora desejado.

Observações.: É importante certificar-se de que o valor que você está tentando converter é compatível com o tipo de destino. Se a conversão não for possível, uma exceção `InvalidCastException` pode ser lançada.

Em alguns casos, o método `TryParse` pode ser preferido em relação a `To*` porque ele verifica se a conversão é bem-sucedida antes de tentar usá-la.

A classe `Convert` também oferece métodos para conversão de outros tipos de dados, como arrays e enumerações.

Exemplos:

Convertendo string para inteiro:

```
string valorString = "100";  
int valorInteiro = Convert.ToInt32(valorString);
```

Convertendo string para booleano:

```
string valorString = "true";  
bool valorBooleano = Convert.ToBoolean(valorString);
```

Convertendo valor para string com formatação:

```
double valor = 1234.5678;  
string valorString = Convert.ToString(valor);
```

Use a documentação do .NET para obter informações detalhadas sobre todos os métodos disponíveis na classe Convert.

Sempre verifique se a conversão é possível antes de tentar usá-la para evitar erros.

Considere usar o método TryParse para conversões onde o valor de entrada pode ser inválido.

MÉTODO PARSE

O método Parse é usado para converter uma representação de string de um valor em seu tipo de dados correspondente.

É comumente empregado para analisar strings em tipos básicos como números inteiros (int), números de ponto flutuante (double), booleanos (bool) e datas (DateTime).

Funcionalidade

- **Entrada:** O Parse método recebe um argumento de string que contém o valor a ser convertido.
- **Conversão:** tenta interpretar a string com base nas regras de formatação do tipo de dados de destino.
- **Saída:** Se a conversão for bem-sucedida, o método retornará o valor convertido no tipo de dados desejado.

O Parse método geralmente oferece versões de sobrecarga que permitem especificar a cultura (localidade) para formatar e analisar números de maneira diferente com base em convenções regionais (por exemplo, vírgula versus separador decimal).

Para cenários de análise mais complexos, você pode explorar lógica de análise personalizada ou bibliotecas especializadas.

Ao compreender o método Parse e suas possíveis armadilhas, você pode converter com eficiência representações de cadeia de caracteres em tipos de dados em seus aplicativos C#.

Exemplos:

Conversões Numéricas Básicas:

```
// conversão Integer  
string intString = "100";  
int intValue = int.Parse(intString);  
Console.WriteLine("Converted integer: {0}", intValue);  
  
// conversão Short  
string shortString = "-128";  
short shortValue = short.Parse(shortString);  
Console.WriteLine("Converted short: {0}", shortValue);
```

```

// conversão Byte
string byteString = "255";
byte byteValue = byte.Parse(byteString);
Console.WriteLine("Converted byte: {0}", byteValue);

// conversão Long
string longString = "9223372036854775807";
long longValue = long.Parse(longString);
Console.WriteLine("Converted long: {0}", longValue);

// conversão Unsigned integer
string uintString = "4294967295";
uint uintValue = uint.Parse(uintString);
Console.WriteLine("Converted unsigned int: {0}", uintValue);

// conversão Ushort
string ushortString = "65535";
ushort ushortValue = ushort.Parse(ushortString);
Console.WriteLine("Converted ushort: {0}", ushortValue);

// conversão Ubyte
string ubyteString = "255";
byte ubyteValue = byte.Parse(ubyteString);
Console.WriteLine("Converted ubyte: {0}", ubyteValue);

// conversão Ulong
string ulongString = "18446744073709551615";
ulong ulongValue = ulong.Parse(ulongString);
Console.WriteLine("Converted ulong: {0}", ulongValue);

```

Condicionais

As estruturas condicionais em C# permitem que o programador controle o fluxo de execução do código com base em determinadas condições. Elas são ferramentas essenciais para tomar decisões e executar diferentes ações em diferentes cenários.

Conditional IF

A instrução básica if verifica uma condição. Se a condição for avaliada como true, o bloco de código entre chaves "{}" após a if execução da instrução. Se a condição for false, esse bloco de código será ignorado

```

if ( // condição)
{
    // código a ser executado se a condição for verdadeira
}

```

Condicional IF-else

A instrução if-else expande a instrução if fornecendo um bloco de código alternativo a ser executado se a condição for falsa.

```
if (condição)
{
    // código a ser executado se a condição for verdadeira
}
else
{
    // código a ser executado se a condição for falsa
}
```

Condicional Switch

A instrução switch é usada para avaliar uma expressão e executar um bloco de código correspondente ao valor da expressão. É útil para lidar com múltiplas condições de forma organizada.

```
switch (expressão)
{
    case valor1:
        // código a ser executado se a expressão for igual a valor1
        break;
    case valor2:
        // código a ser executado se a expressão for igual a valor2
        break;
    default:
        // código a ser executado se a expressão não corresponder a nenhum valor
        break;
}
```

Condicional ternário (?:)

O operador condicional ternário (?:) é uma maneira concisa de escrever uma instrução if-else em uma única linha. Ele avalia uma expressão booleana e retorna um dos dois valores, dependendo do resultado da expressão.

valor = condição ? valorVerdadeiro : valorFalso;

Aplicações das condicionais

As estruturas condicionais são usadas em diversos cenários na programação C#, como:

- Validar dados de entrada do usuário
- Verificar se um arquivo existe antes de acessá-lo
- Implementar lógica de negócios complexa com base em diferentes condições
- Controlar o fluxo de navegação em uma interface gráfica
- Manipular diferentes tipos de dados e estruturas de dados

Benefícios do uso de condicionais:

- **Maior clareza e legibilidade do código:** As estruturas condicionais facilitam a compreensão do fluxo de execução do código, tornando-o mais organizado e fácil de manter.
- **Tomada de decisões dinâmica:** Permitem que o programa se adapte a diferentes situações e execute ações específicas com base em condições variáveis.
- **Reutilização de código:** Blocos de código comuns podem ser encapsulados em instruções condicionais e reutilizados em diferentes partes do programa.
- **Gerenciamento de erros e exceções:** Possibilitam identificar e tratar erros e exceções de forma eficiente, garantindo a robustez do programa.

Dominar as estruturas condicionais é essencial para qualquer programador C#. Elas fornecem a base para construir programas robustos, flexíveis e adaptáveis a diversos cenários.

Laços de repetição em c#

Os laços de repetição, também conhecidos como loops em inglês, são elementos fundamentais da linguagem C#, permitindo a execução repetitiva de blocos de código até que uma condição específica seja satisfeita. Eles são essenciais para automatizar tarefas e otimizar o código, tornando-o mais conciso e eficiente. No C#, existem quatro tipos principais de laços de repetição.

Laço for

O laço for é o mais utilizado e versátil, ideal para executar um bloco de código um número determinado de vezes. Ele possui três partes:

- **Inicialização:** Uma variável de controle é declarada e inicializada com um valor inicial.
- **Condição:** Uma expressão booleana é avaliada. Se verdadeira, o bloco de código é executado.
- **Atualização:** A variável de controle é incrementada ou decrementada.

```
for (int i = 0; i < 10; i++)  
{  
    Console.WriteLine($"Número: {i}");  
}
```


Sobre o Laço for do exemplo

Declaração da variável de controle: `int i = 0;`


- A variável `i` do tipo `int` é declarada e inicializada com o valor 0.

Condição de término: `i < 10;`

- A condição `i < 10` é verificada antes de cada iteração. Se verdadeira, o bloco de código é executado. Caso contrário, o laço termina.

Atualização da variável de controle: `i++`


```
for (int i = 0; i < 10; i++)  
{  
    Console.WriteLine($"Número: {i}");  
}
```



Após a execução do bloco de código, a variável `i` é incrementada em 1.

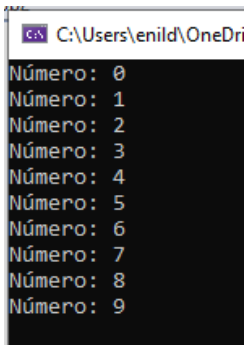
Bloco de código:

```
for (int i = 0; i < 10; i++)  
{  
    Console.WriteLine($"Número: {i}");  
}
```



A instrução `console.WriteLine` imprime uma mensagem na tela do console.

A string formatada `"Número: {i}"` intercala o valor da variável `i` na string, resultando na mensagem "Número: 0", "Número: 1", "Número: 2" e assim por diante.



Funcionamento do laço

O laço inicia com a inicialização da variável `i` como 0.

A condição `i < 10` é verificada. Como `i` é 0, a condição é verdadeira e o bloco de código é executado.

A mensagem "Número: 0" é impressa na tela.

A variável `i` é incrementada para 1.

A condição `i < 10` é verificada novamente. Como `i` agora é 1, a condição ainda é verdadeira e o bloco de código é executado novamente.

A mensagem "Número: 1" é impressa na tela.

O processo se repete até que a condição `i < 10` se torne falsa, ou seja, quando `i` atingir o valor 10.

O laço termina e a execução do programa continua após o bloco `for`.

Laço while

O laço while executa um bloco de código enquanto uma condição booleana for verdadeira. Ele é útil para situações em que o número de iterações não é conhecido previamente.

```
int numero = 1;
while (numero <= 10)
{
    Console.WriteLine($"Número: {numero}");
    numero++;
}
```

Sobre o Laço while do exemplo

Condição de término: `numero <= 10`

A condição `numero <= 10` é verificada antes de cada iteração. Se verdadeira, o bloco de código é executado. Caso contrário, o laço termina.

A instrução `Console.WriteLine` imprime uma mensagem na tela do console.

A string formatada `$"Número: {numero}"` intercala o valor da variável `numero` na string, resultando na mensagem "Número: 1", "Número: 2", "Número: 3" e assim por diante.

A instrução `numero++` incrementa a variável `numero` em 1.

Funcionamento do laço

O laço inicia com o valor inicial de `numero` (que não é mostrado no código fornecido).

A condição `numero <= 10` é verificada. Se verdadeira, o bloco de código é executado.

A mensagem "Número: 1" é impressa na tela (assumindo que o valor inicial de `numero` seja 1).

A variável `numero` é incrementada para 2.

A condição `numero <= 10` é verificada novamente. Como `numero` agora é 2, a condição ainda é verdadeira e o bloco de código é executado novamente.

A mensagem "Número: 2" é impressa na tela.

A variável `numero` é incrementada para 3.

O processo se repete até que a condição `numero <= 10` se torne falsa, ou seja, quando `numero` atingir o valor 11.

O laço termina e a execução do programa continua após o bloco while.

Laço do-while

O laço do-while é semelhante ao laço while, mas com a diferença de que o bloco de código é executado pelo menos uma vez, mesmo que a condição inicial seja falsa.

```
int numero = 11;
do
{
    Console.WriteLine($"Número: {numero}");
    numero++;
}
while (numero <= 10);
```

Sobre o Laço do-while do exemplo

Bloco de código: As instruções dentro do bloco do serão executadas pelo menos uma vez, mesmo que a condição seja falsa inicialmente.

Condição de término: `numero <= 10`.

A condição `numero <= 10` é verificada após a execução do bloco de código. Se verdadeira, o laço é executado novamente. Caso contrário, o laço termina.

```
int numero = 11;
do
{
    Console.WriteLine($"Número: {numero}");
    numero++;
}
while (numero <= 10);
```

A instrução `Console.WriteLine` imprime uma mensagem na tela do console.

A string formatada `$"Número: {numero}"` intercala o valor da variável `numero` na string, resultando na mensagem "Número: 1", "Número: 2", "Número: 3" e assim por diante.

A instrução `numero++` incrementa a variável `numero` em 1.

Funcionamento do laço

O laço inicia com o valor inicial de `numero` (que não é mostrado no código fornecido).

O bloco de código é executado pelo menos uma vez, independentemente da condição inicial.

A mensagem "Número: 1" é impressa na tela (assumindo que o valor inicial de `numero` seja 1).

A variável `numero` é incrementada para 2.

A condição `numero <= 10` é verificada. Se verdadeira, o laço é executado novamente.

A mensagem "Número: 2" é impressa na tela.

A variável `numero` é incrementada para 3.

O processo se repete até que a condição `numero <= 10` se torne falsa, ou seja, quando `numero` atingir o valor 11.

O laço termina e a execução do programa continua após o bloco do-while.

Laço foreach

O laço foreach é utilizado para iterar sobre os elementos de uma coleção, como arrays ou listas. Ele simplifica a iteração, não sendo necessário gerenciar manualmente a variável de controle.

```
int[] numeros = { 1, 2, 3, 4, 5 };  
foreach (int numero in numeros)  
{  
    Console.WriteLine($"Número: {numero}");  
}
```

Sobre o Laço foreach do exemplo

`int[] numeros`: Declara um array de inteiros chamado numeros.

`{ 1, 2, 3, 4, 5 }`: Inicializa o array com cinco valores inteiros: 1, 2, 3, 4 e 5.

`foreach`: Introduz o laço foreach.

`int numero`: Define uma variável temporária numero do tipo inteiro para armazenar o valor corrente de cada elemento do array durante a iteração.

`in numeros`: Indica o array numeros sobre o qual queremos iterar.

A instrução `Console.WriteLine` imprime uma mensagem na tela do console.

A string formatada `"Número: {numero}"` intercala o valor da variável numero na string, resultando na mensagem "Número: 1", "Número: 2", "Número: 3" e assim por diante.

Funcionamento do laço

O laço foreach itera automaticamente sobre cada elemento do array numeros.

Em cada iteração:

A variável temporária numero recebe o valor do elemento corrente do array.

O bloco de código é executado utilizando o valor armazenado em numero.

A mensagem "Número: 1" é impressa (primeira iteração com numero recebendo o valor 1).

A mensagem "Número: 2" é impressa (segunda iteração com numero recebendo o valor 2).

... e assim por diante para todos os elementos do array.

Este código demonstra como o laço foreach simplifica a iteração sobre os elementos de um array. Ele elimina a necessidade de gerenciar manualmente a variável de controle e o índice do array, tornando o código mais conciso e legível.

Resumo

A escolha do laço de repetição adequado depende do contexto e da necessidade específica do programa. O laço `for` é ideal para um número fixo de iterações, enquanto o laço `while` é útil para situações em que a condição de término é desconhecida. O laço `do-while` garante a execução pelo menos uma vez, e o laço `foreach` simplifica a iteração em coleções.

Utilize laços de repetição para evitar código repetitivo e melhorar a legibilidade do programa.

Escolha o tipo de laço de repetição correto para cada situação, considerando o número de iterações e a necessidade de verificar a condição de término.

Utilize variáveis de controle para acompanhar o progresso do laço e controlar a execução do código.

Combine laços de repetição aninhados para realizar tarefas mais complexas.

Lembre-se de que os laços de repetição são ferramentas poderosas que podem otimizar seu código e torná-lo mais eficiente. Dominá-los é essencial para se tornar um programador C# mais habilidoso.

.NET

.NET (pronunciado "dot NET") é uma plataforma de desenvolvimento de software gratuito e de código aberto criada pela Microsoft. Imagine o .NET como um grande conjunto de ferramentas que os programadores podem usar para construir diferentes tipos de aplicativos.

Características importantes do .NET:

- **Versátil:** Pode ser usado para criar aplicativos web, mobile, desktop, jogos e até mesmo aplicações para Internet das Coisas (IoT).
- **Multiplataforma:** Embora originalmente desenvolvido para Windows, o .NET agora permite a criação de aplicativos que rodam em Windows, macOS e Linux.
- **Código aberto:** O código fonte do .NET está disponível publicamente, o que significa que programadores do mundo todo podem contribuir para o seu desenvolvimento.
- **Rico em funcionalidades:** Oferece um vasto conjunto de bibliotecas e APIs (interfaces de programação de aplicativos) para auxiliar no desenvolvimento de software, tornando a vida dos programadores mais fácil.

Resumidamente, o .NET é uma plataforma poderosa e flexível que permite a criação de uma ampla variedade de aplicativos modernos.

Conceitos: RAD, C#, UML e IDE

- O ambiente de desenvolvimento **C#** enquadra-se na categoria de ferramentas do tipo **RAD – Rapid Application Development** (Desenvolvimento Rápido de Aplicações).
- **C#** é uma linguagem orientada a “objetos” e eventos, com suporte a UML, criada pela Microsoft e distribuída com o Visual Studio.
- A “Unified Modeling Language (**UML**) é um modo de padronizar as formas de modelagem”.
- “**IDE**” um ambiente integrado para desenvolvimento de software.

Programação Orientada a Objetos (POO)

A POO, sigla para Programação Orientada a Objetos, é um paradigma de programação que organiza o código em torno de objetos, entidades que representam elementos do mundo real e possuem características (atributos) e comportamentos (métodos). Imagine um objeto como um bloco de construção modular, com suas próprias propriedades e funcionalidades, que se unem para formar sistemas complexos.

Os Pilares da POO

Para entender a essência da POO, vamos mergulhar em seus quatro pilares fundamentais:

Abstração: A arte de focar no essencial! A abstração permite que você se concentre nas características e comportamentos mais importantes de um objeto, ignorando detalhes irrelevantes. É como criar um modelo simplificado do objeto, capturando sua essência sem se perder em minúcias.

Encapsulamento: Mantenha seus segredos seguros! O encapsulamento protege os dados internos de um objeto, permitindo que apenas seus métodos autorizados acessem e modifiquem essas informações. Imagine um cofre para seus dados valiosos, acessível apenas por chaves específicas.

Herança: O legado dos pais para os filhos! A herança permite que novos objetos sejam criados a partir de objetos já existentes, herdando suas características e comportamentos. É como um filho que nasce com as características dos pais, mas com a possibilidade de desenvolver suas próprias habilidades únicas.

Polimorfismo: Uma forma, várias faces! O polimorfismo permite que objetos de diferentes classes respondam à mesma mensagem de maneiras distintas. Imagine um jogo onde vários personagens podem realizar a ação "atacar", mas cada um com seu estilo único, seja com um soco, um chute ou um raio mágico.

A POO oferece diversas vantagens que a tornam uma escolha popular para o desenvolvimento de software:

Reutilização de código: Evite reinventar a roda! Com a POO, você pode criar componentes reutilizáveis que podem ser usados em diferentes partes do seu programa, economizando tempo e esforço.

Modularidade: Quebre o monstro em pedaços menores! A POO divide o código em módulos independentes e bem definidos, facilitando a compreensão, o desenvolvimento e a manutenção do software.

Manutenção facilitada: Faça as pazes com os bugs! A POO facilita a identificação e correção de erros, pois os objetos encapsulam seus próprios dados e comportamentos, isolando problemas em áreas específicas.

Flexibilidade e extensibilidade: Adapte-se às mudanças! A POO torna o software mais flexível e extensível, permitindo que novas funcionalidades sejam facilmente adicionadas ou modificadas sem afetar o código existente.

Exemplos de Linguagens POO

Diversas linguagens de programação populares utilizam o paradigma da POO, como:

Java: Uma linguagem robusta e versátil, amplamente utilizada para desenvolvimento de aplicações web, desktop e mobile.

C++: Uma linguagem poderosa e performática, ideal para aplicações que exigem alto desempenho, como jogos e sistemas embarcados.

Python: Uma linguagem simples e fácil de aprender, utilizada para desenvolvimento web, análise de dados e machine learning.

C#: Uma linguagem moderna e multiplataforma, desenvolvida pela Microsoft e utilizada para diversos tipos de aplicações, desde jogos até sistemas empresariais.

A POO é um paradigma fundamental para o desenvolvimento de software moderno, oferecendo organização, modularidade, reutilização de código e flexibilidade para criar sistemas complexos e escaláveis. Se você deseja desbravar o mundo da programação, dominar a POO é um passo essencial para se tornar um desenvolvedor habilidoso e versátil.

Windows Forms

Windows Forms é uma estrutura de interface gráfica de usuário (GUI) desenvolvida pela Microsoft. Ele permite que você crie aplicativos de desktop para sistemas operacionais Windows.

- **Fácil de usar:** especialmente para iniciantes, o Windows Forms oferece um designer visual com funcionalidade de arrastar e soltar para organizar os elementos da interface do usuário em seus formulários.
- **Tecnologia madura:** com uma longa história, há muitos recursos e bibliotecas disponíveis para ajudá-lo a desenvolver aplicativos Windows Forms.
- **Ampla gama de funcionalidades:** Suporta um rico conjunto de recursos para criar aplicativos complexos com menus, botões, grades de dados e muito mais.
- **Idiomas:** usa principalmente C# e Visual Basic .NET para desenvolvimento.

Criando um aplicativo C# windows Forms.

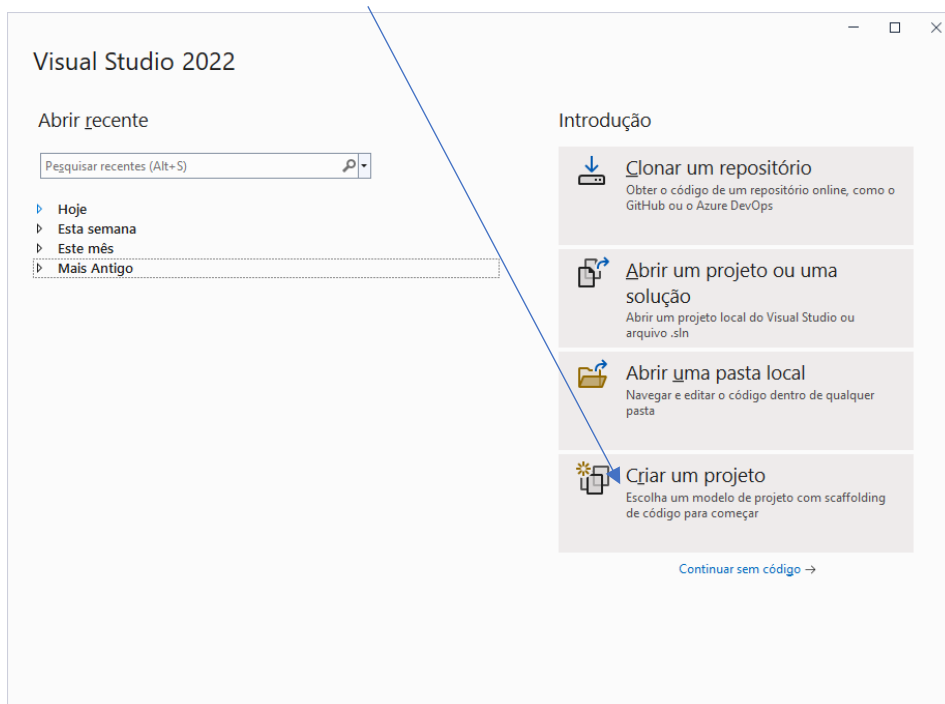
Crie um projeto C# no Visual Studio 2022

Primeiro, crie um projeto de aplicativo C#. O tipo de projeto vem com todos os arquivos de modelo necessários para criar seu aplicativo.

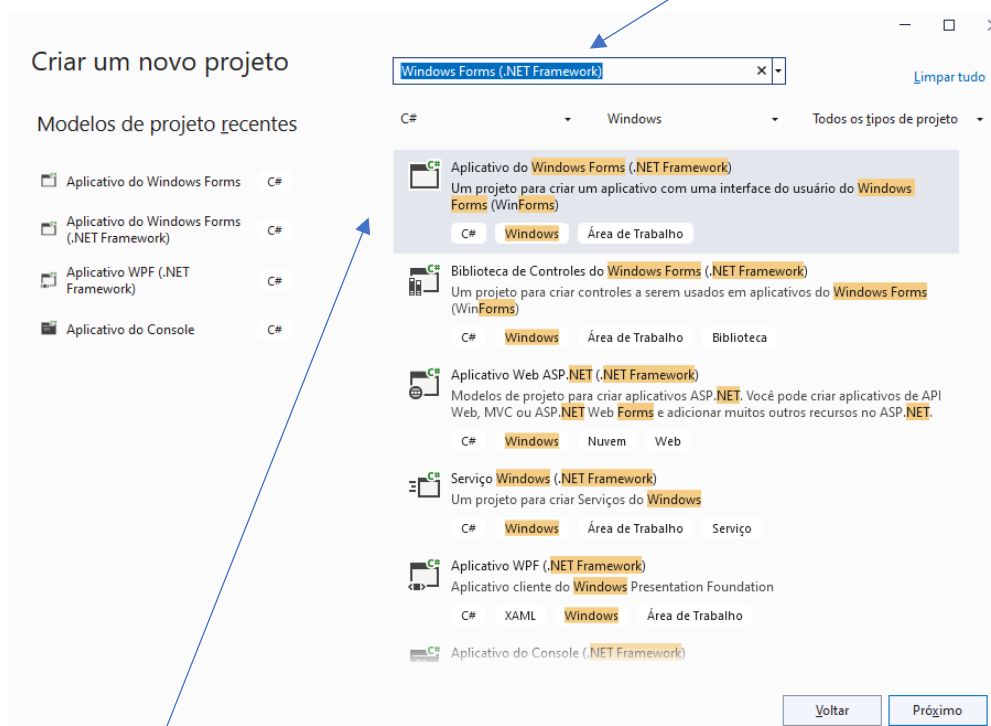
Abra o Visual Studio.



Na janela inicial, selecione “Criar um novo projeto”.

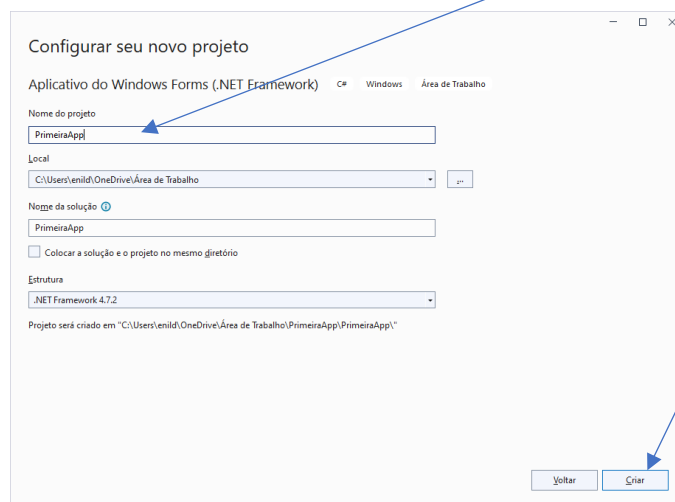


Em “Criar um novo projeto”, selecione o modelo Aplicativo Windows Forms (.NET Framework) para C#.

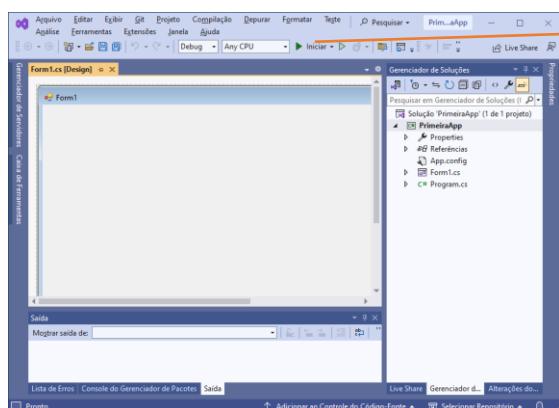


Você pode refinar sua pesquisa para chegar rapidamente ao modelo desejado. Por exemplo, digite Aplicativo Windows Forms na caixa de pesquisa. Em seguida, selecione C# na lista de idiomas e selecione Windows na lista de plataformas.

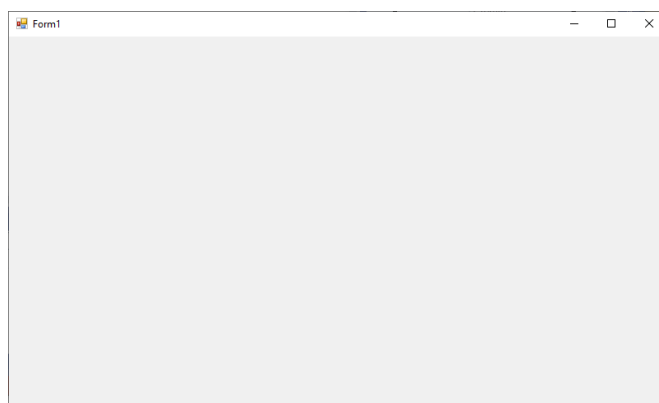
Na janela Configurar seu novo projeto , em Nome do projeto , insira PrimeiraApp e selecione Criar .



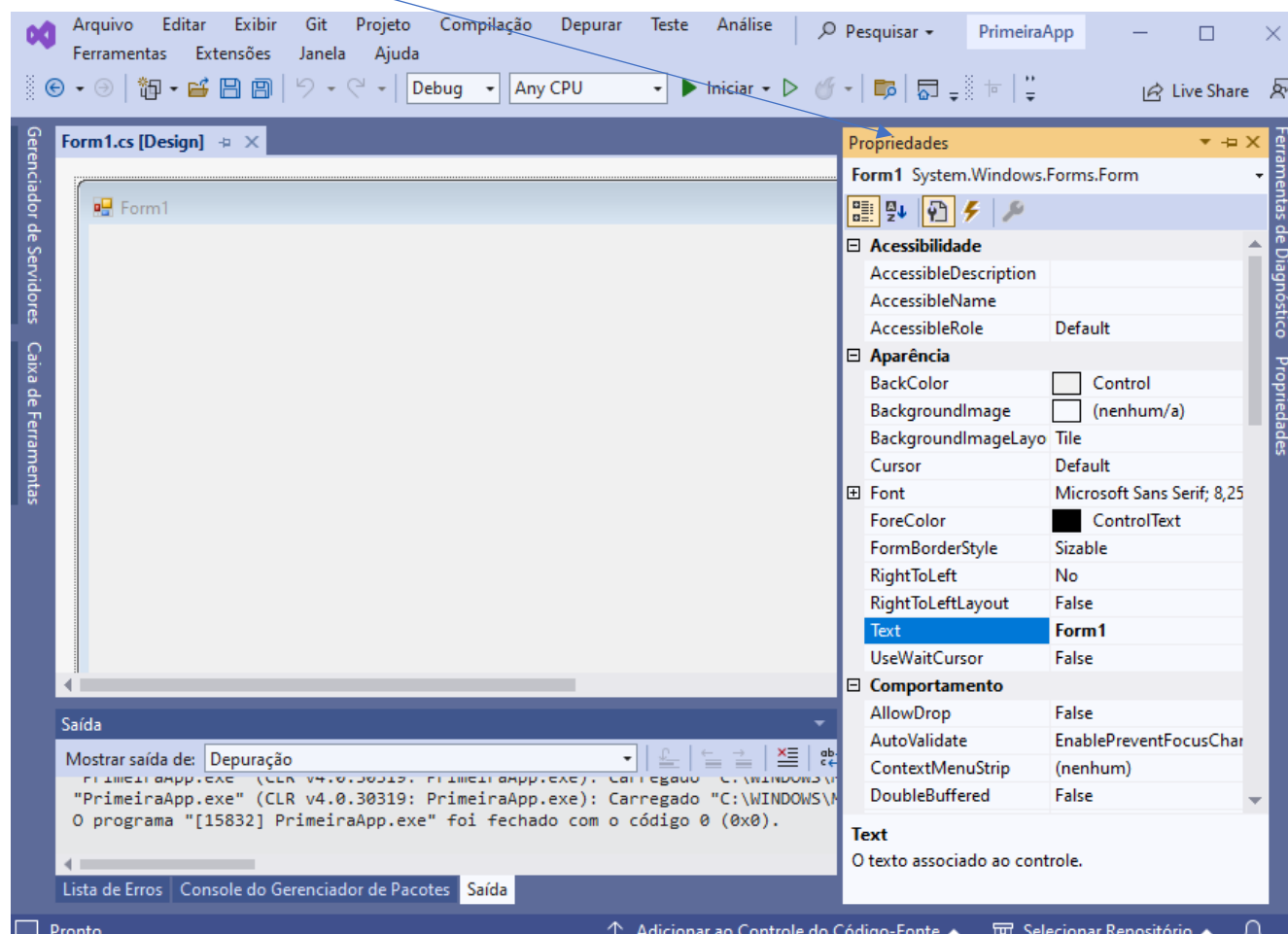
Visual Studio abre seu novo projeto.



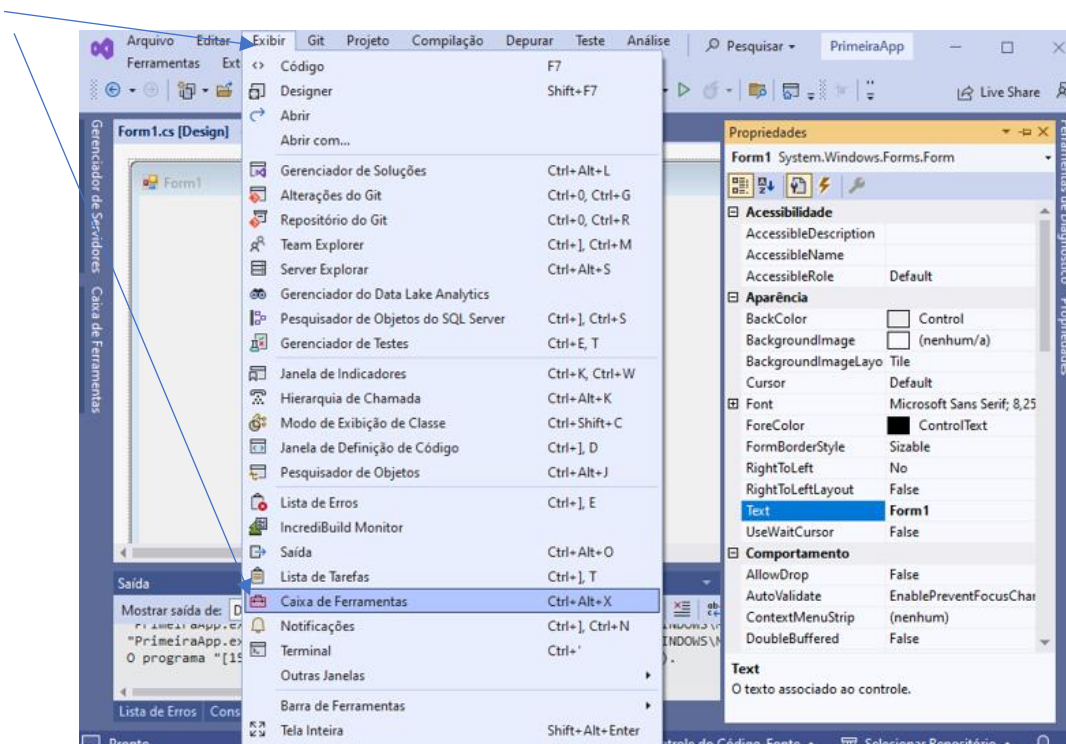
Teste
Clique em “Iniciar” ou F5



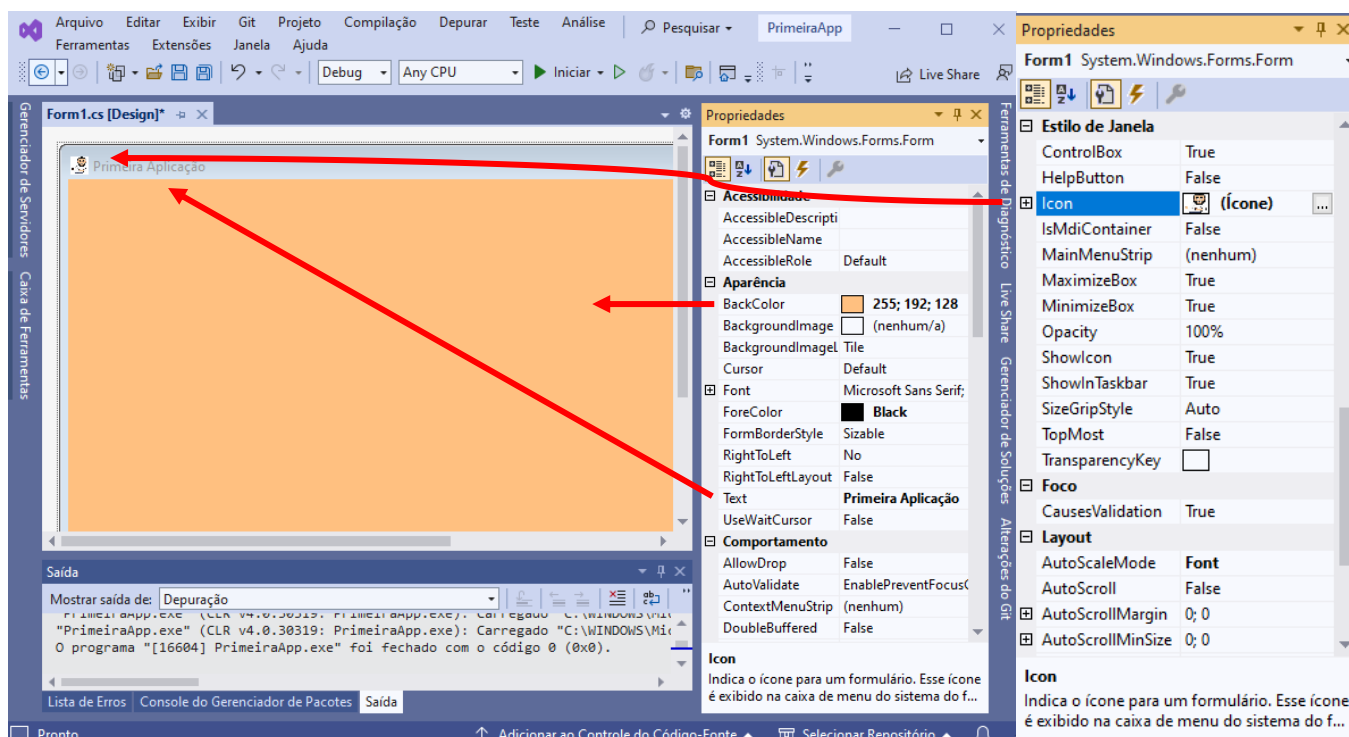
Localize a aba de “Propriedades” na lateral direita.



Caso não ache a janela de propriedades na lateral da plataforma, pode achar, todas as janelas, no menu em “Exibir”.



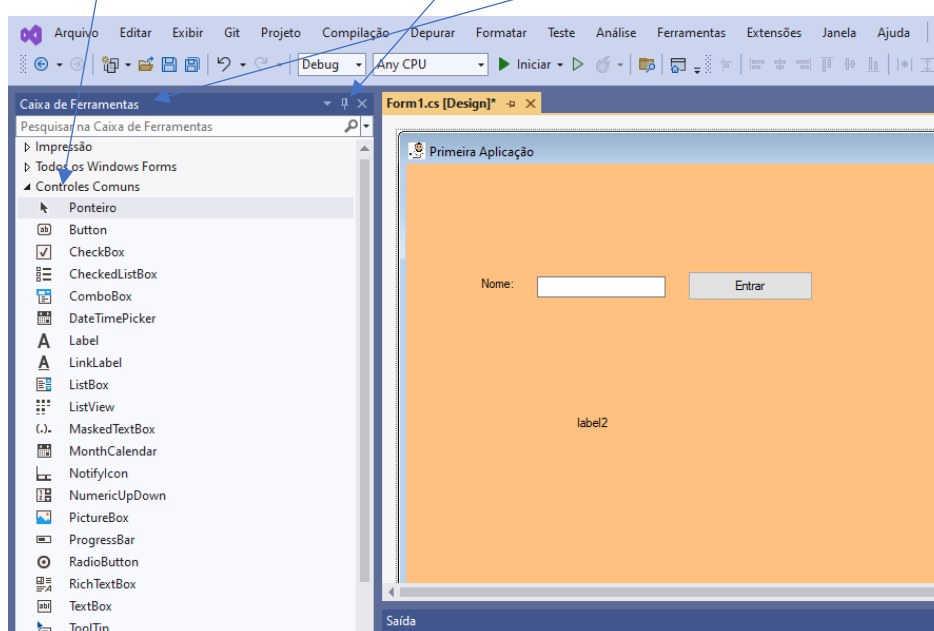
Todos os objetos têm propriedades, ao selecionar o objeto você terá as propriedades dele já janela de propriedades. Podendo alterar todas as propriedades conforme desejar. Veja no exemplo as mudanças de cor, texto e ícone.



Adicionando controles ao formulário

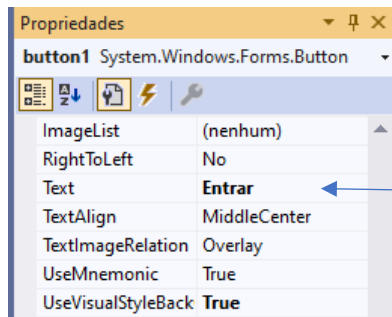
Selecione Caixa de ferramentas para abrir a janela suspensa Caixa de ferramentas .

Expanda Controles Comuns e selecione o ícone Fixar para encaixar a janela da Caixa de Ferramentas .

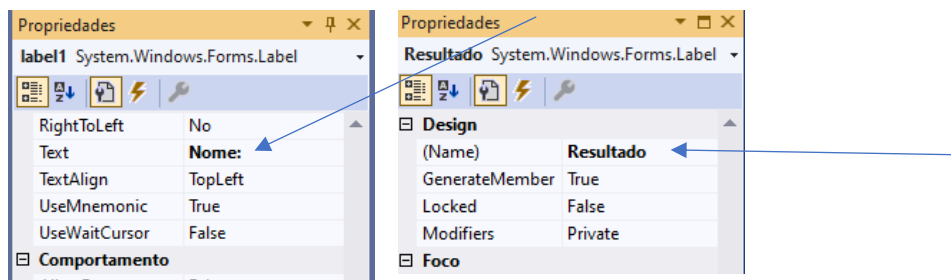


Adicione um botão ao formulário. Selecione o controle Button e arraste-o para o formulário.

Na janela Propriedades , localize Text . Altere o texto do botão



Adicione dois rótulos ao formulário.



Na janela Propriedades , localize Text e altere o texto do rotulo para “Nome:”.

Selecione o outro rotulo e na janela Propriedades , localize (Name) e altere para “Resultado”.

