Algoritmo - Sequência lógica e finita de passos (instruções) para resolver um determinado problema

Exemplo: as pessoas utilizam-se de algoritmos de maneira intuitiva

- preparar o bolo
- troca de um pneu furado
- resolver uma equação

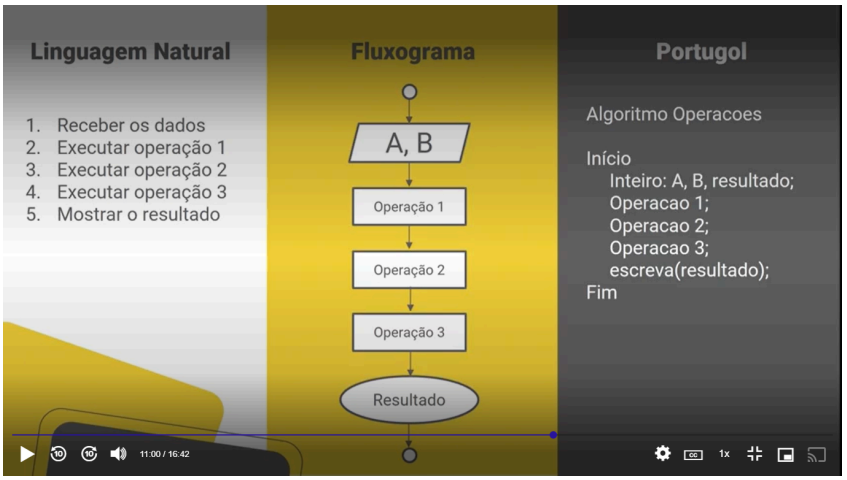
Algoritmo TrocaPneusCarro
1: desligar o carro
2: pegar as ferramentas (chave de roda)
3: pegar o estepe
4: suspender o carro com o macaco
5: desenroscar os 4 parafusos do pneu furado
6: colocar estepe
7: enroscar os 4 parafusos
8: baixar o carro com o macaco
9: guardar as ferramentas

Componentes de um algoritmo:
entrada - Dados ou elementos para processamento
Processamento - etapas para alcançar o resultado final
Saída - Resultado a ser alcançado

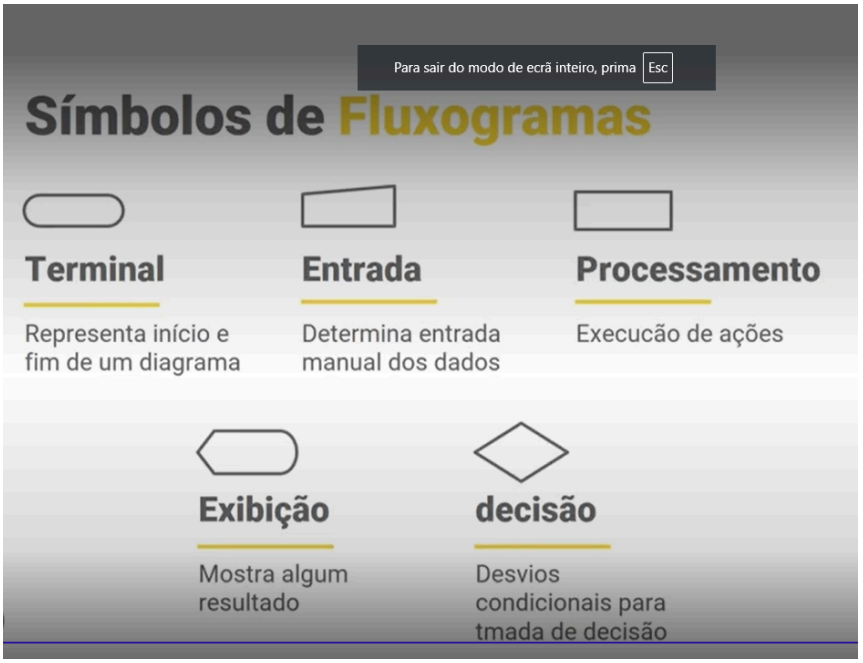
Variáveis

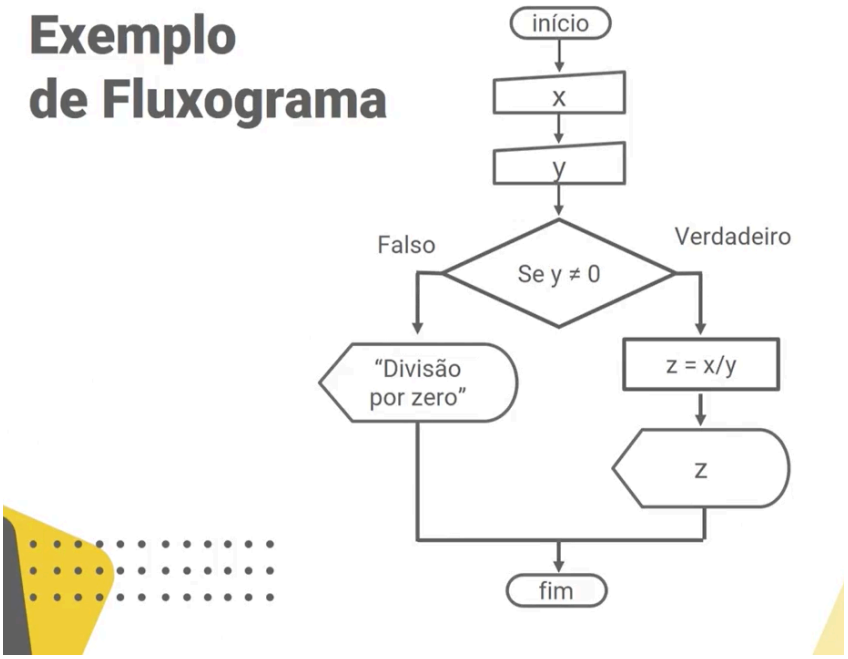
- As **variáveis** são elementos que podem ser alterados (sofrer variações) e estão relacionadas à identificação de informações
- Já uma atribuição (->) serve para designar valores às variáveis, ou seja, para atribuir informações a elas

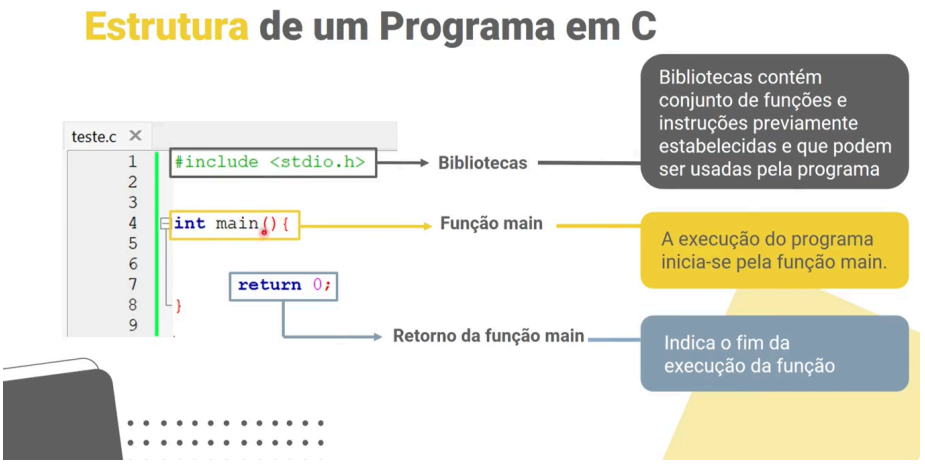
Exemplo: valor 1 <- 6
nome <- "joao"

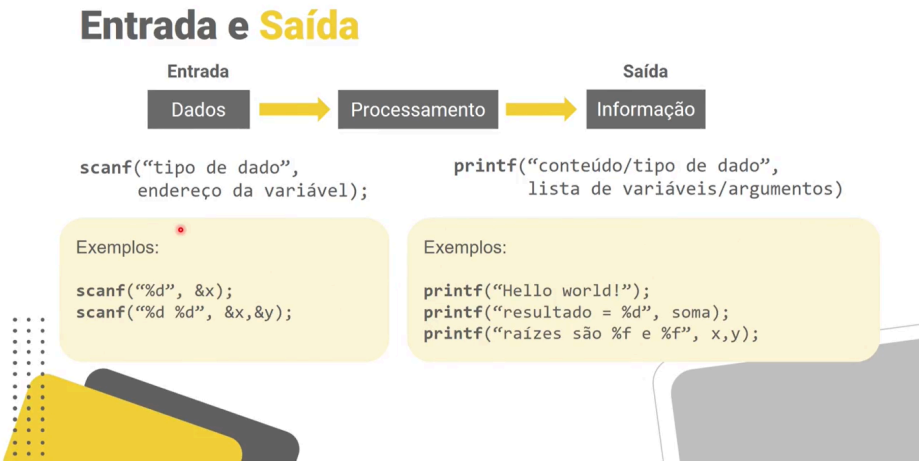


Fluxogramas - coleção de símbolos gráficos, onde cada um desses símbolos representa ações específicas a serem executadas pelo computador
símbolos padronizados pela ANSI (Instituto Nacional Americano de Padrões)









scanf -> leitura de dados
printf -> saída de dados
%d -> numero inteiro
%f -> numero real
& -> significa que eu quero o endereço da variável x

Para criar um programa de computador, é necessário seguir certos passos:

1. Definir e **analisar** o problema a ser resolvido, destacando os pontos mais importantes.
2. Definir os dados de **entrada**: coleta de informações
3. Definir o **processamento**, ou seja, quais cálculos serão executados e as suas restrições. O processamento é responsável pela transformação dos dados de entrada em informações de saída.
4. Definir os dados de **saída**, ou seja, o que será gerado após o processamento: apresentação de todas as informações resultantes do processamento de dados em um dispositivo periférico (monitor, por exemplo).
5. Criar um **algoritmo** ou um diagrama de fluxo.
6. **Testar** o algoritmo realizando simulações.

Veja alguns exemplos de biblioteca em linguagem de programação C:

- **stdio** – essa biblioteca é responsável pelas funções de entradas e saídas, como é o caso da função printf e scanf que vamos aprender mais à frente. Utilização: #include <stdio.h>

- **stdlib** – essa biblioteca possui funções envolvendo alocação de memória, controle de processos, conversões e outras. Utilização: #include <stdlib.h>
- **string** – biblioteca responsável pela manipulação de strings. Utilização: #include<string.h>
- **time** – biblioteca utilizada para manipulação de horas e datas. Utilização: #include<time.h>
- **math** – biblioteca utilizada para operações matemáticas. Utilização: #include<math.h>

Os tipos de variáveis mais comumente utilizados são:

- **Inteiro (int)**: armazena os números inteiros (negativos ou positivos).

- **Real (float)**: permite armazenar valores de pontos flutuantes e frações. Quando se necessita do dobro de dados numéricos é utilizado o tipo "**double**" ou "**long double**".
- **Caractere (char):** permite armazenar caracteres, números e símbolos especiais. São delimitadas por aspas simples (‘).

Código	Função
%d	Exibe/lê um número inteiro
%f	Exibe/lê um número em ponto flutuante (decimal)
%c	Exibe/lê um caractere
%s	Exibe/lê uma sequência de caracteres (string)
%e	Exibe/lê um número em notação científica

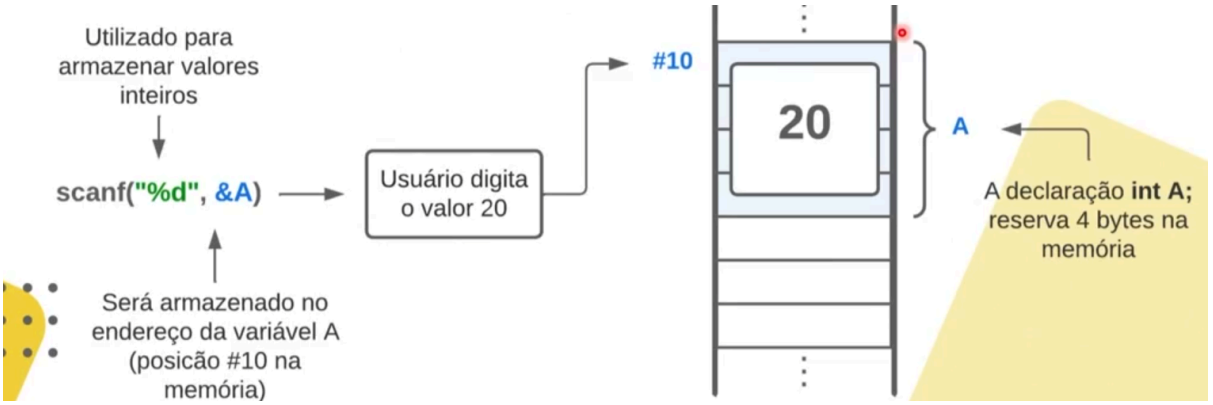
No **printf()** você pode pular linhas com o comando "**\n**" e pode também obter um

resultado numérico determinando a quantidade de casas decimais.
Veja o exemplo a seguir:

```
printf (“\n Resposta: a = %.2f e b = %.2f\n”, a, b);
```

Neste exemplo, antes de apresentar a frase o programa pulou uma linha “**\n**”, o “**%f**” é utilizado quando os dados numéricos são flutuantes, ou seja, valores fracionados. E quando usamos **%.2f** significa que o valor será arredondado em duas casas decimais, por exemplo: 2,45.

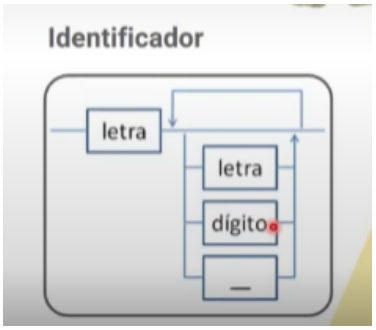
Perceba que aqui temos o uso da instrução “**system**(“pause”)”, que pausa o fluxo de execução do programa naquele ponto para que o resultado seja visualizado.



Tipos
Inteiro (int): ex.: 1, 3, -5, 198, 0
Real (float/double): ex.: 0.5, 5.0, -9.85
Caractere (char): ex.: 'c', '1', "joao"
Booleano(bool): ex.: Veradeira ou Falso -> Para utilizá-lo em C, devemos incluir a biblioteca **<stdbool.h>**.

Já para o **<nome_da_variável>** devemos seguir algumas regras, tais como:

- Devem começar com uma letra.
- Os próximos caracteres podem ser letras ou números.
- Não pode utilizar nenhum símbolo, exceto *underline* (_).
- Não pode conter espaços em branco.
- Não pode conter letras com acentos.



Constante se refere a um valor que permanece inalterado ao longo do programa

#define <nome_da_constante> <valor>
const <tipo> <nome_da_constante>
Os três principais modificadores são:

- **unsigned**: utilizado para especificar que a variável armazenará apenas a parte positiva do número.
- **short**: reduz o espaço reservado na memória.
- **long**: aumenta a capacidade padrão.

Podemos utilizar operadores aritméticos, relacionais e lógicos ao mesmo tempo em uma expressão;
Porém, devemos ficar atentos à sua ordem de precedência

Operador	Descrição	Exemplo	Resultado
+	Soma	8 + 4	12
-	Subtração	8 - 4	4
*	Multiplicação	8 * 4	32
/	Divisão	8 / 4	2
=	Atribuição	x = 8	x = 8
%	Módulo	8 % 4	0

Os operadores aritméticos seguem a seguinte ordem de execução:

1. Parênteses.
2. Potenciação e radiciação.
3. Multiplicação, divisão e módulo.
4. Adição e subtração.

- **Pré-incremento e pré-decremento**: quando o operador é colocado antes da variável (++x ou --x), o valor é alterado e utilizado na expressão original.
- **Pós-incremento e pós-decremento**: quando o operador é colocado após a variável (x++ ou x--), o valor é utilizado na expressão original e depois é alterado.

Operador	Descrição	Exemplo
==	Igual a	x == y
!=	Diferente de	x != y
>	Maior que	x > y
<	Menor que	x < y
>=	Maior ou igual a	x >= y
<=	Menor ou igual a	x <= y

Operador	Descrição	Exemplo
!	Negação (NOT)	!(x == y)

&&	Conjunção (AND)	(x > y) && (a == b)
||	Disjunção (OR)	(x > y) || (a == b)

- O operador de negação é usado para inverter o resultado de uma expressão.
- O operador de conjunção é utilizado para estabelecer condições em que todas as alternativas devem ser verdadeiras.
- O operador de disjunção é utilizado para criar condições em que basta uma das condições ser verdadeira para que o resultado seja verdadeiro também.
- Parênteses para forçar uma avaliação específica.
- Operadores unários, como os de incremento (++) e decremento (--).
- Multiplicação (*), divisão (/) e módulo (%).
- Adição (+) e subtração (-).
- Operadores relacionais, como menor que (<), maior que (>), menor ou igual (<=), maior ou igual (>=), igual (==) e diferente (!=).
- Operadores lógicos: 'E' (&&), 'OU' (||) e 'NÃO' (!).

ALGORITMOS

PASSO A PASSO

1 ANÁLISE

Definir e analisar o problema a ser resolvido, destacando os pontos mais **importantes**. Identificar qual o resultado esperado.

2 ENTRADA

Definir os dados de entrada: coleta de informações. Aqui identificamos as **variáveis** que serão criadas bem como seus respectivos **tipos** de acordo com a análise do problema. Também nos preocupamos com a estrada de dados.

3 PROCESSAMENTO

Definir quais **cálculos** serão executados e as suas restrições. O processamento é responsável pela **transformação** dos dados de entrada em informações de saída. Aqui verificamos a necessidade de realizar operações aritméticas, relacionais e/ou lógicas, por exemplo.

4 SAÍDA

Definir os dados de saída, ou seja, o que será gerado após o processamento: apresentação de todas as informações **resultantes** do processamento de dados em um dispositivo periférico (monitor, por exemplo).

5 DESENVOLVIMENTO E TESTES

Por fim, o algoritmo é desenvolvido com base nos requisitos de entrada, processamento e saída previamente identificados. Nesta etapa é importante manter uma prática de simulações e testes para avaliar o que está sendo desenvolvido.