

Construção de um algoritmo

Um algoritmo pode ser obtido seguindo o seguinte algoritmo (Ascencio and Campos 2007):

1. Compreender o problema a ser resolvido, destacando os aspectos mais importantes (abstração);
2. Definir os dados de entrada, ou seja, quais dados serão fornecidos;
3. Definir o processamento, quais operações devem ser efetuadas com os dados fornecidos e quais as restrições para essas operações (regras de negócio);
4. Definir os dados de saída, conforme solicitado no enunciado do problema;
5. Construir o algoritmo;
6. Testar o algoritmo (verificar se o algoritmo está correto).

Embora a **descrição narrativa**, utilizando o linguagem natural (idioma) possa ser utilizada para construir um algoritmo, um padrão formal contribui para facilitar o desenvolvimento e a compreensão de soluções algorítmicas.

Um padrão é utilizar um conjunto de instruções previamente definido, onde cada instrução representa uma ação específica, denominado **pseudocódigo**. Outra forma de explicitar uma solução algorítmica são os **fluxogramas**. Nesse caso, as ações são representadas por formas geométricas definidas. O Quadro 1 contém as principais instruções de um algoritmo para as representações em pseudocódigo e em fluxograma.

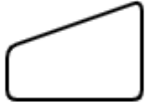


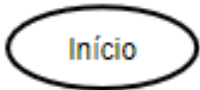

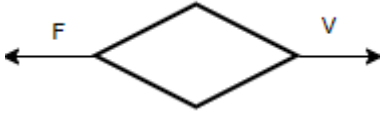
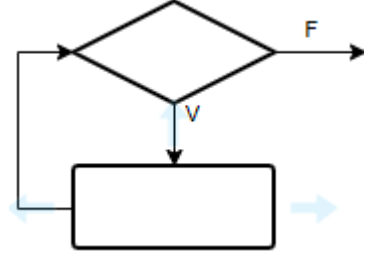
Variáveis e tipos de dados

Na matemática, uma variável é um símbolo que representa um determinado valor. Da mesma forma, os algoritmos utilizam variáveis para **representar os valores** que serão tratados, quer seja como dados de entrada, de saída ou necessários ao processamento. Cada variável tem um **identificador** único.

De modo geral, os algoritmos (em especial os fluxogramas) apenas utilizam os identificadores para representar os valores, sem os formalismos que são necessários nas linguagens de programação estruturadas.

No entanto, algoritmos em pseudocódigo costumam explicitar que tipo de valor cada variável irá armazenar. Os dados podem ser numéricos (**inteiro** ou **real**), literais (**caracter** ou **literal**) ou **lógicos**. No Quadro 2 estão listados os tipos primitivos de dados.

Quadro 1: Instruções algorítmicas em pseudocódigo e fluxograma.

Instrução	Pseudocódigo	Fluxograma
Entrada de dados	leia	
Saída de dados	escreva imprima	
Processamento	<- = expressões aritméticas e/ou atribuições	
Início do algoritmo	Início	
Término do algoritmo	Fim FimAlgoritmo	
Estrutura de seleção	se ... entao ... senao	
Estrutura de repetição	repita até ... enquanto ... faca para ... de ... até ... passo ...	

Antes de utilizar uma variável, é preciso dizer que tipo de valor ela irá armazenar. Em alguns padrões de pseudocódigo, há uma seção própria para a **declaração das variáveis**, antes do início das instruções. Uma instrução de declaração de variável inclui o identificador e o tipo de dado.

Exemplo de declaração de variáveis:

```
idade : inteiro
peso : real
nome : literal
```

Quadro 2: Tipos primitivos de dados.

Tipos primitivos de dados		
Dado numérico	Inteiro	<ul style="list-style-type: none"> ○ Não possuem componentes decimais ○ Positivos ou negativos ○ Ex: 13, 0, -473
	Real	<ul style="list-style-type: none"> ○ Possuem componentes decimais ou fracionário (em geral delimitados por ponto) ○ Positivos ou negativos ○ Ex: 0.0, -32.6, 48723.98, 12049578
Dado literal	Caractere	<ul style="list-style-type: none"> ○ Representa apenas uma letra, número ou símbolo ○ Delimitados por aspas simples ‘ ’ ○ Ex: ‘a’, ‘+’, ‘1’, ‘ ’
	Literal	<ul style="list-style-type: none"> ○ Sequência de caracteres (texto) ○ Delimitado por aspas duplas “” ○ Ex: “nome”, “Paulo da Silva”, “r2d2”, “Onde?”
Dado lógico (ou Booleano)		<ul style="list-style-type: none"> ○ Apenas dois valores: verdadeiro ou falso ○ Ex: Sim/Não, 1/0, true/false, V/F

Nomes de identificadores válidos

Os identificadores de variáveis, funções e mesmo o nome do algoritmo/programa em um pseudocódigo (ou programa) devem ser nomeados cumprindo as seguintes regras (Manzano and Oliveira 2010):

- Os nomes de identificação podem utilizar um ou mais caracteres. Algumas linguagens de programação podem ter limites quanto ao tamanho.
- Jamais uma variável (ou outro elemento) pode ser definida com o mesmo nome de uma palavra que represente um dos comandos ou instruções de uma linguagem de programação de computadores ou pseudocódigo (por exemplo, `leia` ou `escreva`).
- O primeiro caractere de identificação do nome não pode ser numérico ou de símbolo gráfico (cifrão, tralha, cachimbo, vírgula, ponto e vírgula, traço, parênteses, chaves, colchetes, entre outros). O primeiro caractere deve ser sempre alfabético. Os demais caracteres podem ser alfanuméricos (números ou letras).

- Na definição de um nome composto não pode haver espaços em branco. Caso deseje separar nomes compostos, pode-se utilizar o caractere de separação “_” (*underline*), ou o padrão de letras maiúsculas no início das demais palavras (*camel case*). Por exemplo: nomeDoCliente, nome_do_cliente.
- Não pode ser utilizado algum rótulo que já tenha sido usado para identificar o nome de um algoritmo (programa), função ou outra variável. Um nome torna-se exclusivo no algoritmo/programa em que foi definido.
- E a regra mais **importante**: use identificadores **significativos**, de acordo com o contexto do problema.

São identificadores válidos: nomeusuario, nome_usuario, nomeUsuario, n_usuario, fone1, fone_1, f1, f_1, x, delta25.

São identificadores inválidos: nome usuario, 1x, fone#, leia, escreva, inicio.

Operadores aritméticos e de atribuição

O Quadro 3 apresenta os operadores aritméticos utilizados nos algoritmos (Manzano and Oliveira 2010, Feofiloff 2009).

No quadro, os operadores estão em ordem de prioridade, e agrupados por cor (exponenciação e radiciação são de mesma prioridade, assim como a divisão e a multiplicação). Por exemplo, na instrução $x \leftarrow a + b * 4$, a primeira operação será a multiplicação, depois a soma e, por último, a atribuição do resultado à variável x .

No caso de operadores de mesma prioridade em uma mesma expressão, a ordem de execução é da esquerda para a direita. Por exemplo, na expressão $x \leftarrow z * y / 3$, a primeira operação será a multiplicação, depois a divisão e, por último a atribuição do resultado à variável x .

Quando um valor (ou o resultado de uma expressão) é atribuído a uma variável, o seu conteúdo anterior é sobrescrito e não pode mais ser recuperado. Na maioria das linguagens, o sinal de igualdade é usado para atribuição, em substituição à seta (\leftarrow).

Quadro 3: Operadores aritméticos e de atribuição.

	OPERADORES	OPERAÇÃO	DESCRIÇÃO
ARITMÉTICOS	\uparrow	$x \uparrow n$	Exponenciação de x^n
	\wedge	$x \wedge n$	
	$**$	$x ** n$	
	$\uparrow(1/n)$	$x \uparrow (1/n)$	Radiciação de $\sqrt[n]{x}$
	SQR	SQR (x)	Raiz quadrada de x
	/	x / n	Divisão de x por n
	DIV	$x \text{ DIV } n$	Divisão inteira de x por n (para x e n inteiros)
	/	x / n	
	MOD	$x \text{ MOD } n$	Resto da divisão inteira de x por n
	%	$x \% n$	
	*	$x * n$	Multiplicação de x por n
	+	$x + n$	Adição de x com n
	-	$x - n$	Subtração de n de x
	\leftarrow	$x \leftarrow n$	Atribuição do valor de n a x

Teste de mesa

O teste de mesa permite identificar se um algoritmo está correto quanto à sua funcionalidade. Para isso, deve-se avaliar o conteúdo das variáveis ao longo da execução do algoritmo.

Uma lista das variáveis e a atribuição dos valores é suficiente; é importante que o teste seja feito com mais de uma instância, avaliando especialmente o comportamento do algoritmo com os valores limites, conforme o contexto do problema.

Uma representação melhor de um teste de mesa é utilizar uma tabela, em que as **variáveis** e **condições** estão representadas nas colunas e as instâncias (conjunto de dados) nas linhas. Cada variável deve ser representada em uma coluna, da mesma forma que as todas as condições do algoritmo. Cada condição deve constituir uma coluna da

tabela, indicada pelo sinal de interrogação (por exemplo, $(a > b?)$ $(n \% 2 = 0?)$ $(mes \leq 12?)$).
Veja os exemplos a seguir.

Exemplo 1 - Construindo uma solução algorítmica

Considere o seguinte problema:

Joãozinho tem 32 figurinhas em seu álbum e ganhou mais 5 figurinhas de seu irmão. Quantas figurinhas Joãozinho tem?

Seguindo o algoritmo para construção de algoritmo:

1. Qual o problema? *Saber o total de figurinhas.*
2. Quais os valores de entrada? *As duas quantidades: 32 e 5.*
3. Qual o processamento? *A soma dos valores de entrada.*
4. Qual o dado de saída? *O resultado da soma.*
5. Algoritmo

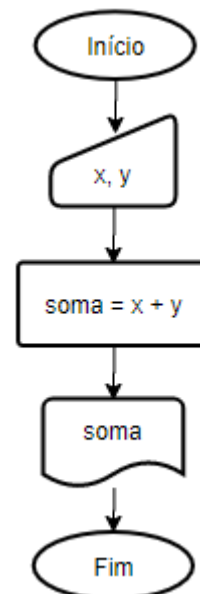
Pseudocódigo

```

1 Algoritmo "Soma"
2 // Descrição: Soma dois valores numericos
3 // Autor(a): aetp
4 // Data atual: 7/21/2021
5
6 Var
7   x, y, soma : inteiro
8
9 Inicio
10   leia (x)
11   leia (y)
12   soma <- x + y
13   escreva (soma)
14 Fimalgoritmo

```

Fluxograma

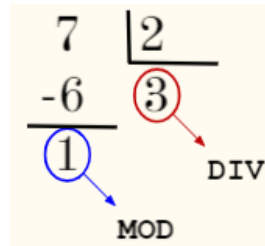


6. Teste de mesa

# teste	Entrada		Processamento	Saída
	x	y	soma	
1	32	5	37	37
2	7	-3	4	4
3	-3	0	-3	-3
4	3072	1234	4306	4306

Exemplo 2 - Divisão de inteiros

É possível obter quociente e resto de uma divisão de inteiros, por meio de operadores específicos. Se a divisão (`/` ou `DIV`) é usada com variáveis do tipo inteiro, o resultado será um inteiro; o resto pode ser obtido por meio do operador de módulo (`%` ou `MOD`).



Observe as seguintes expressões para valores inteiros:

$$6/2 = 3$$

$$7/2 = 3$$

$$14/3 = 4$$

$$33/5 = 6$$

$$6\%2 = 0$$

$$7\%2 = 1$$

$$14\%3 = 2$$

$$33\%5 = 3$$

Segue a solução algorítmica que retorna o quociente e o resto da divisão.

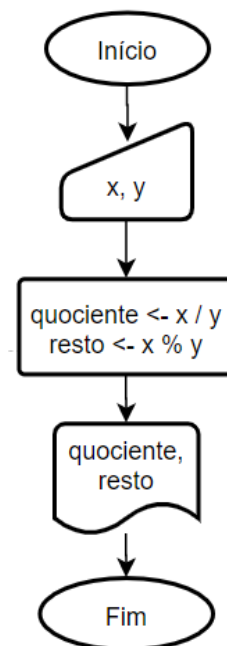
Pseudocódigo

```

1 Algoritmo "Divisão Inteira"
2 // Descrição: Obter quociente e resto de
3 //dois valores
4 // Autor(a): aetp
5 // Data atual: 7/21/2021
6 Var
7   x, y, quociente, resto : inteiro
8
9 Inicio
10   leia (x, y)
11   quociente <- x div y
12   resto <- x mod y
13   escreva (quociente, resto)
14 Fimalgoritmo

```

Fluxograma



Teste de mesa

#	Entrada		Processamento		Saída
	x	y	quociente	resto	
1	6	2	3	0	3 0
2	7	2	3	1	3 1
3	33	5	6	3	6 3

Uso inadequado da instrução de saída de dados (escreva)

É comum ver exemplo de algoritmos usando a instrução escreva para mostrar mensagem antes da leitura de uma variável, da seguinte forma:

```
escreva ("Digite o valor de A: ")
leia (A)
```

Seguem algumas razões que indicam a inadequação do uso da instrução escreva dessa forma:

1. Essas mensagens não auxiliam na resolução do problema, muitas vezes até a tornam mais complexa (aumentando a quantidade de instruções). Qual o conteúdo da variável A? A utilização de boas práticas de documentação (Feofiloff 2009) é muito mais adequada para esclarecimentos a respeito da função do algoritmo (**o que** o algoritmo faz) e das variáveis (se for o caso);
2. A instrução `leia` implica na entrada de dado, não especificamente um valor digitado pelo usuário. Outras fontes de entrada de dados podem ser: um dado lido de um componente de hardware do sistema (sensor, leitora ótica, leitor magnético, etc), de um arquivo, ou valor recebido de outra função. Veja o exemplo dos problemas utilizados nos sistemas de julgamento automático[1];
3. Arquitetura de software: à medida que se aumenta a complexidade de uma solução computacional, recomenda-se dividir e organizar o projeto em camadas [2] ou componentes [3] (dependendo do padrão de projeto utilizado). Por exemplo, em uma arquitetura de três camadas [4], o tipo de entrada poderá ser tratado na camada de apresentação (interface), enquanto a resolução do problema em si é tratada na camada de negócio (é nessa camada que estamos trabalhando, e não em soluções interativas (Farrell 2010)), enquanto os dados são armazenados em uma camada própria (camada de dados).

[1] https://en.wikipedia.org/wiki/Competitive_programming

[2] https://pt.wikipedia.org/wiki/Arquitetura_multicamada

[3] <https://pt.wikipedia.org/wiki/MVC>

[4] https://pt.wikipedia.org/wiki/Modelo_em_três_camadas.

Qualidades de um algoritmo

Todo bom algoritmo e programa tem três qualidades fundamentais (Feofiloff 2009):

- **Correção:** um algoritmo é correto se faz o que se espera dele, ou seja, cumpre o que a sua documentação promete;

- **Eficiência:** um algoritmo é eficiente se não desperdiça tempo (e espaço de memória). Dados dois algoritmos para um mesmo problema, o mais eficiente é aquele cuja execução consome menos tempo;
- **Elegância:** um algoritmo é elegante se for simples, limpo, bonito, sem enfeites. Um algoritmo elegante não trata casos especiais do problema em separado. Um algoritmo elegante não tem código supérfluo, nem variáveis desnecessárias, nem construções convoluídas e "espertas", nem sutilezas evitáveis.

Antes de aprender a construir algoritmos corretos, é preciso aprender a verificar se um algoritmo está correto. A verificação da correção de um algoritmo é uma atividade semelhante à prova de um teorema. A verificação depende do enunciado preciso **d'o que** o algoritmo deve fazer; esse enunciado constitui a **documentação** do algoritmo.

Uma das formas de provar a correção de um algoritmo é estudar a relação entre os valores das variáveis em pontos estratégicos do código. A técnica mais comumente usada é o **teste de mesa**.

A eficiência dos algoritmos - ou melhor, o consumo de tempo em função do tamanho das instâncias¹ - pode ser analisada, inicialmente, de maneira informal e intuitiva. Os métodos de análise de complexidade de algoritmos são importantes para a formação em computação, sendo assunto de disciplinas mais avançadas do curso.

Embora o conceito de elegância seja subjetivo, bons programadores concordam entre si quando julgam a elegância de um algoritmo. Consulte o apêndice intitulado Leiaute em (Feofiloff 2009).

Documentação: o que *versus* como

A documentação é **o que** o algoritmo faz, o código (pseudocódigo, fluxograma) é **como** faz (a especificação da solução) (Feofiloff 2009). Uma boa documentação evita sujar o código com comentários e limita-se a **explicar o que** cada uma das funções que compõem o programa faz.

A documentação de um programa (ou de uma função do programa) é um minimanual que dá instruções precisas e completas sobre o seu uso, que contém (Feofiloff 2009):

¹ Cada conjunto de dados (um exemplo concreto) de um problema define uma instância do problema.
Por exemplo: Problema: calcular a soma de dois números.
Instância: calcular a soma de 234 e 987.

- especificação das entradas: dados que o programa recebe;
- especificação das saídas: resultados que o programa devolve;
- descrição da relação entre os dados fornecidos e as saídas produzidas.

Com esses elementos, é possível identificar os erros de implementação do programa.

Em geral, uma boa documentação não perde tempo tentando explicar como um programa faz o que faz – isso é possível saber lendo o código. A distinção entre o que um programa faz e como faz o que faz é essencial: considere a analogia de uma empresa de entregas, que pega seu pacote e entrega no destino especificado em dois dias. Isto é o que a empresa faz. Como o serviço será feito – transporte terrestre, aéreo ou marítimo – é assunto interno da empresa. Veja o exemplo da documentação da função “filaDeNectarOuMel” da Lição “Funções com parâmetros na abelha” do Code.org:

The screenshot shows the 'Função' (Function) editor interface. At the top, there's a green header with the title 'Função' and a 'Fechar' (Close) button. Below the header, the form is divided into sections:

- Nomeie sua função:** A text input field containing 'filaDeNectarOuMel'.
- O que sua função deverá fazer?** A text input field containing the description: 'Verifica se há flor ou colmeia em quadros subsequentes (conforme o parâmetro fornecido), obtendo néctar ou fazendo mel.'
- Quais são os parâmetros necessários para sua função?** A section with a text input field and a button labeled 'Adicionar parâmetro'.
- comprimento** A dropdown menu showing 'comprimento'.
- filaDeNectarOuMel com: comprimento** A section showing a block of code using the 'comprimento' parameter. The code is written in a visual programming style with blocks: 'repita' (repeat) block with 'comprimento' and 'vezes' (times) inputs, followed by a 'faça' (do) block containing 'avance' (move forward), 'se na flor' (if on flower), 'faça obtenha néctar' (do get nectar), 'se na colmeia' (if on hive), and 'faça faça mel' (do make honey).

Bibliografia

- Ascencio, Ana Fernanda Gomes, and Edilene Aparecida Veneruchi de Campos. 2007. *Fundamentos da programação de computadores: algoritmos, pascal, c/c++ e java*. 2nd ed. São Paulo, SP: Pearson Prentice Hall.
- Farrell, Joyce. 2010. *Lógica e Design de Programação: Introdução*. 5th ed. São Paulo: Cengage Learning.
- Feofiloff, Paulo. 2009. *Algoritmos em Linguagem C*. Rio de Janeiro: Elsevier.
- Forbellone, André Luiz V., and Henri F. Eberspächer. 2005. *Lógica de Programação: A construção de algoritmos e estruturas de dados*. 3ª ed. São Paulo: Pearson.
- Manzano, José Augusto N., and Jayr F. Oliveira. 2010. *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores*. São Paulo: Érica.
- Wikipedia. 2021. "Algoritmo." Wikipedia. <https://pt.wikipedia.org/wiki/Algoritmo>.



Exercícios

A. Supondo que as variáveis **nome**, **idade**, **disciplina** e **nota** sejam usadas para armazenar o nome do aluno, a idade do aluno, o nome da disciplina e a nota, respectivamente, declare-as corretamente, associando o tipo primitivo adequado ao dado que será armazenado.

B. Sendo $x = 15$, considere a execução das seguintes instruções:

```
x <- x + 3
x <- x - 6
x <- x / 2
x <- 3 * x
```

Qual o valor de x ?

C. Supondo que A, B e C são variáveis do tipo **inteiro**, com valores iguais a 5, 10 e -8, respectivamente, e que D é uma variável do tipo **real**, com valor igual a 1.5, calcule as expressões aritméticas:

1. $2 * (A \% 3) - C =$
2. $((20 \text{ div } 3) \text{ div } 3) + \exp(2, 3) / 2 =$
3. $(30 \% 4 * \exp(3, 3)) * -1 =$

4. $\exp(-C, 2) + (D * 10) / A =$
5. $\text{raizq}(\exp(A, 2)) + C * D =$

D. Suponha as seguintes variáveis:

```
soma, numero, cont: real
nome, cor, dia, mes, ano: literal
teste, codigo, tudo: logico
```

Determine se as atribuições são válidas ou inválidas. Justifique as inválidas.

1. `nome <- 5`
2. `soma <- numero + 2 * cont`
3. `teste <- codigo`
4. `tudo <- soma`
5. `cor <- "Azul" - exp(cont, 2)`
6. `cont <- cont + 1`
7. `numero <- "*abC*"`
8. `dia <- "Segunda feira"`
9. `mes <- "Agosto"`
10. `ano <- 2006`
11. `soma + 2 <- cont - numero`

E. Considere o pseudocódigo:

```
1 Algoritmo "O que faz?"
2 // Descrição : ?
3 // Autor(a) : aetp
4 // Data atual: 7/27/2021
5 Var
6   x, y, z : inteiro
7
8 Inicio
9   leia (x)
10  y <- x div 60
11  z <- x mod 60
12  escreva (y, z)
13 Fimalgoritmo
```

1. O que o algoritmo faz? Faça a documentação (descreva sucintamente sua funcionalidade)².
2. As variáveis poderiam ter outros nomes, que auxiliassem a compreensão da funcionalidade do algoritmo?

² Dica: Faça o teste de mesa para um conjunto de instâncias para compreender o algoritmo.

F. Desenvolva algoritmos para:

1. Calcular o quadrado de um número;
2. Somar dois números;
3. Mostrar o sucessor e o antecessor de um número.

Há alguma relação entre a quantidade de dados de entrada, instruções de processamento e valores de saída? Explique.

G. Desenvolva a solução algorítmica e verifique sua solução (teste de mesa) para os seguintes problemas:

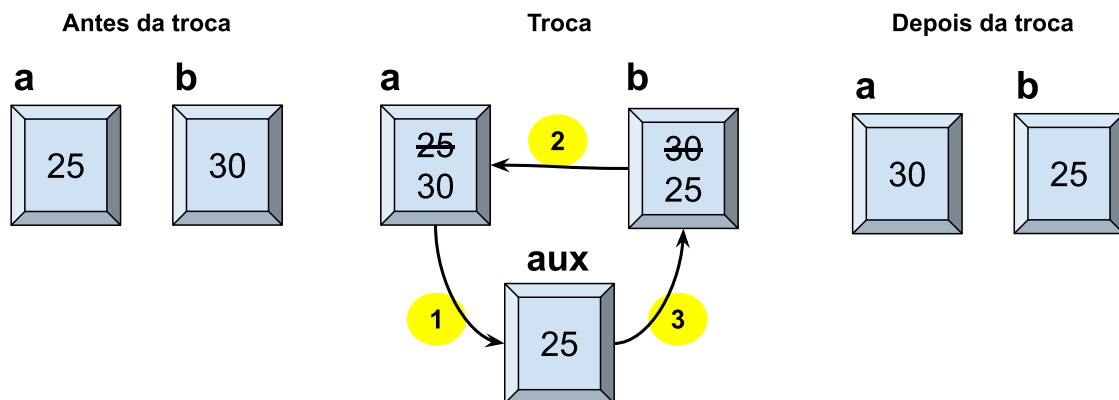
1. Calcule a média ponderada de quatro valores, considerando os pesos 1, 2, 3 e 4, conforme a ordem de entrada dos valores.
2. Sabendo que 100 *quilowatts* de energia custa um sétimo do salário mínimo, calcule, a partir do valor do salário mínimo e da quantidade de *quilowatts* gasta em uma residência, o valor em reais de cada *quilowatt*, o valor em reais da conta do mês e o valor a ser pago com desconto de 10%.
3. Inverta um número de três dígitos no formato CDU (mostre como resultado a sua inversão no formato UDC) (por exemplo, para 123, o resultado será 321).

H. Pesquise o algoritmo para o cálculo do dígito verificador de uma sequência numérica. Reescreva em pseudocódigo. elabore o fluxograma e faça o teste de mesa.



Exemplo Extra - Troca de valores

Considere a necessidade de trocar o conteúdo de duas variáveis. Para trocar o conteúdo de duas variáveis entre si é necessário utilizar outra variável (imagine essa situação com dois aquários!). Essa terceira variável guardará, temporariamente, o conteúdo de uma delas, permitindo a substituição dos conteúdos sem que nenhum dos valores seja perdido.



Na Figura, os círculos amarelos indicam as instruções de atribuição das variáveis:

- (1) `aux <- a`
- (2) `a <- b`
- (3) `b <- aux`

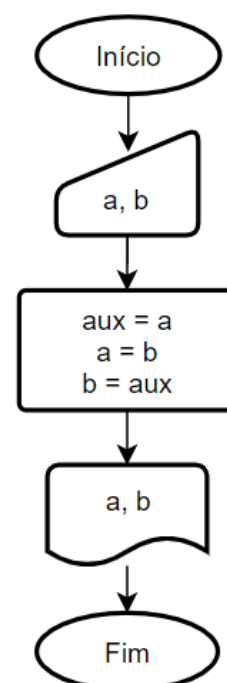
Segue a solução algorítmica para a troca de valor de duas variáveis.

Pseudocódigo

```

1 Algoritmo "Troca"
2 // Descrição : Troca de conteúdo de duas variáveis
3 // Autor(a) : aetp
4 // Data atual: 7/27/2021
5 Var
6   a, b, aux : inteiro
7
8 Inicio
9   leia (a, b)
10  aux <- a
11  a <- b
12  b <- aux
13  escreva (a, b)
14 Fimalgoritmo
  
```

Fluxograma



Teste de mesa

# teste	Entrada		Processamento	Saída
	a	b	aux	
1	3 6	6 3	3	6 3
2	8 2	2 8	8	2 8
3	423 15	45 123	123	15 123

Exercício proposto

Considere a solução para a troca de valores de duas variáveis. O que acontecerá se o bloco de instruções das linhas 9 a 13 for substituído pelos blocos abaixo?

Para responder, explore os conceitos de programação sequencial, operador de atribuição e inicialização de variáveis.

Bloco A

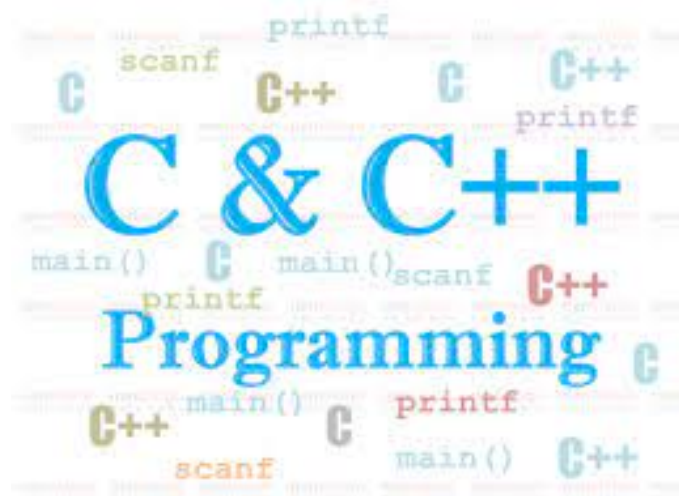
```

9  leia (a, b)
10 a <- b
11 aux <- a
12 b <- aux
13 escreva (a, b)
```

Bloco B

```

9  leia (a)
10 a <- b
11 leia (b)
12 aux <- a
13 b <- aux
14 escreva (a, b)
```

Programação em C/C++: Introdução

Programação de computadores

“Programação é o processo de escrita, teste e manutenção de um programa de computador” (“Programação de Computadores” 2021). Um programa deve ser escrito em uma linguagem que pode ser transformada em um código executável pelo *hardware*.

Uma linguagem de programação consiste em um conjunto de regras **sintáticas** e **semânticas** para a implementação padronizada de um código fonte³. A linguagem permite que um programador especifique as instruções de um **algoritmo** de forma mais **precisa** (sobre quais dados o computador irá atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas) (“Programação de Computadores” 2021).

As linguagens de programação podem estar mais próximas das instruções do processador (linguagem de máquina) (denominadas linguagens de baixo nível), ou distantes, abstraindo os detalhes do hardware e sendo mais similares a linguagem natural (idioma) (denominadas linguagens de alto nível).

Em uma **linguagem de alto nível**, os elementos que se aproximam da linguagem natural são fáceis de usar, ou podem automatizar requisitos importantes (como por exemplo o gerenciamento de memória), tornando o processo de desenvolvimento de um

³ Um código fonte pode ser compilado (a compilação produz um programa executável) ou interpretado (o interpretador informará as instruções de processamento ao computador) (“Programação de Computadores” 2021).

programa mais simples e compreensível do que usando uma linguagem de baixo nível. A abstração é proporcional ao nível da linguagem (“Programming Language” 2021).

Linguagem C/C++

Desenvolvida entre 1969 e 1973 como linguagem de programação para o sistema operacional Unix, ainda está em uso devido a velocidade de execução e aplicações enxutas (“Programming Language” 2021).

C é uma linguagem de propósito geral, procedural, que suporta programação estruturada. O conceito de programação estruturada reside em que todos os programas possíveis podem ser reduzidos a três estruturas: sequência, decisão e repetição.

Embora para muitos a linguagem C++ seja uma escolha melhor, devido a ter mais características, mais aplicações e ser mais fácil de aprender, aprender C pode melhorar a forma como programar em C++ (“C Tutorial” 2021).

Nesta disciplina, usaremos a estrutura da linguagem C, com alguns recursos do C++ (ou seja, não seguiremos o paradigma da orientação a objetos, para o qual a linguagem C++ foi desenvolvida). Esses recursos são, inicialmente, as instruções de entrada e saída de dados, com sintaxe mais simples para facilitar a familiaridade com a linguagem (quem souber utilizar `scanf` e `printf`, pode ficar à vontade para utilizar essas funções em C).

Peculiaridades da Linguagem C/C++

Algumas características que requerem atenção ao escrever um programa em C/C++ são:

1. *Case sensitive*: a linguagem diferencia letras maiúsculas e minúsculas, portanto, as variáveis `N` e `n` são variáveis distintas (embora as maiúsculas, por padrão, sejam utilizadas como constantes). Variáveis com caixas incorretas serão reportadas na compilação como não declaradas;
2. Fim de linha de código: todas as linhas de código (exceto as que iniciam um bloco de seleção ou repetição) devem ser finalizadas com `;` (ponto-e-vírgula). O compilador pode apontar um erro em uma linha quando o problema é a falta do ponto-e-vírgula na linha anterior;
3. Demarcadores de blocos de código (`{ }`): os blocos, que no pseudocódigo são iniciados por “Início” ou outra instrução de início de bloco (*se, caso, repita,*

para) e finalizados por “Fim” (ou fim-se, fim-escolha, fim-para, fim-enquanto), em C são marcados por chaves ({ e }).

Sintaxe da Linguagem C/C++

Observe o programa abaixo.

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main(){
6     int a, b, soma;
7
8     cin >> a >> b;
9     soma = a + b;
10    cout << soma;
11 }
```

Para utilizar as funções de entrada e saída de dados, é necessário incluir a biblioteca padrão `iostream`. Isso é feito por meio da diretiva `#include` (linha 1). Além disso, as bibliotecas, por questões de melhor organização, possuem seções (denominadas `namespace`). As instruções de entrada e saída que utilizamos são definidas na `namespace std`. Para não precisar especificar essa seção a cada vez que as instruções `cin` e `cout` forem utilizadas (`std::cin` e `std::cout`), utilizamos a declaração `using` (linha 3).

Um programa em C/C++ inicia a execução a partir da função principal. A implementação de uma solução simples estará implementada nessa função, entre as chaves. Podemos dizer que o cabeçalho `int main(){` equivale ao termo *inicio* de um pseudocódigo (linha 5), enquanto o fecha chaves `}` equivale ao termo *fim*.

Diferentemente do pseudocódigo, no qual há uma seção própria para a declaração de variáveis, em C/C++ a declaração é feita nas primeiras linhas de código (linha 6).

Entrada de dados

A instrução para entrada de dados, equivalente ao *leia* em pseudocódigo, é `cin`. Trata-se de um objeto que representa o fluxo de entrada padrão, que obtém dados de uma fonte externa ao programa (por exemplo, do teclado ou de um arquivo) (“C++ Language” 2000).

O fluxo obtido deve ser direcionado para uma variável, para isso é utilizado o operador de redirecionamento `>>`. Se for necessário ler mais de um valor, operadores de redirecionamento podem ser concatenados na mesma instrução de entrada, mesmo que os dados sejam de tipos diferentes.

```
cin >> numero;
cin >> x1 >> x2;
```

Saída de dados

A instrução para saída de dados, equivalente ao `escreva` em pseudocódigo, é `cout`. Trata-se de um objeto que representa o fluxo de saída padrão, que envia dados para uma fonte externa ao programa (por exemplo, para o monitor ou para um arquivo) (“C++ Language” 2000).

Para que as informações sejam direcionadas, é necessário o operador `<<`. Vários conteúdos podem ser concatenados na mesma instrução `cout` utilizando operadores de redirecionamento. A quebra de linha pode ser obtida com a instrução `endl` ou com o caractere `'\n'`.

```
cout << numero << '\n';
cout << "Resultado: " << result << endl;
```

Operadores

O quadro abaixo contém os principais operadores aritméticos, relacionais e lógicos da linguagem C/C++.

Aritméticos	Relacionais	Lógicos
<p>+ Soma</p> <p>- Subtração</p> <p>* Multiplicação</p> <p>/ Divisão</p> <p>% Módulo (resto)</p>	<p>= Atribuição</p> <p>== Igualdade</p> <p>!= Diferença</p> <p>>= Maior ou igual</p> <p><= Menor ou igual</p>	<p>! Não</p> <p>&& E</p> <p> Ou (símbolo pipe)</p>

Em C/C++, é comum o uso da forma abreviada de algumas expressões aritméticas (Feofiloff 2009). Por exemplo, a expressão `x = x + 1` pode ser representada, conforme a ordem de precedência dos operadores (da esquerda para a direita, nesse caso, primeiro a soma, depois a atribuição), por `x += 1` ou, sendo `x` uma variável do tipo inteiro, simplesmente por `x++` (em qualquer dos formatos, o resultado será o incremento do valor

de x em uma unidade) O mesmo vale para o operador de subtração ($x--$ irá decrementar o valor de x em uma unidade).

Observe a sequência de instruções:

```
n = 6;
cout << "n++: " << n++ << endl;
cout << "n: " << n << endl;
cout << "++n: " << ++n << endl;
```

O resultado mostrado será:

```
n++: 6
n: 7
++n: 8
```

A instrução `n++` só incrementa o valor de `n` depois de usá-lo, ou seja, primeiro envia o valor de `n` para a saída e depois o atualiza. É possível verificar isso na próxima linha, quando o valor de `n` é mostrado (`n: 7`). A última atualização (`++n`) é feita antes que o valor de `n` seja enviado para a saída, mostrando, portanto, o valor 8 (`n` já incrementado).

Tipos de dados primitivos

O quadro abaixo contém os três tipos de dados mais utilizados. Para saber mais sobre os tipos primitivos em C++, consulte <https://www.cplusplus.com/doc/tutorial/variables/>.

Tipo de dado	C/C++	Valor armazenado/Exemplo
inteiro	<code>int</code>	Inteiros positivos e negativos <code>int contador = 0;</code>
real	<code>float</code>	Casas decimais separadas por ponto <code>float PI = 3.1415;</code>
literal	<code>char</code>	Apenas um caractere, delimitado por aspas simples <code>char l = 'A';</code>

Declaração e inicialização de variáveis

Em C/C++, quando uma variável é declarada, é alocado um bloco de memória e vinculado o endereço ao nome da variável. No entanto, esse bloco de memória pode conter algum conteúdo prévio, que não nos interessa... Então, para evitar que valores indesejados resultem em um erro no programa, é importante prestar atenção ao valor inicial de uma variável. A atribuição de um valor inicial, neutro para o problema que está sendo abordado, é denominada **inicialização da variável**.

Uma boa prática é, no momento de declarar a variável, atribuir um valor inicial a ela, conforme o seu tipo:

```
int n = 0;
float total = 0;
char letra = '';
```

Teste de Operador Lógico

Na execução de uma expressões lógicas, cada parte será traduzida para um valor lógico.

Quando se trata de uma variável inteira, apenas se ela for igual a 0 será falsa, qualquer outro valor retornará verdadeiro para a expressão.

Observe o código abaixo, teste também para os valores:

a = 0

b = 5

```
1 #include <iostream>
2 using namespace std;
3
4 //Teste de condições com
5 operador ||
6
7 int main(){
8     int a, b;
9     a = 1;
10    b = 5;
11
12    if (a < 0 || b < 0){
13        cout << "true" <<
endl;
    }
    else{
        cout << "false" <<
endl;
    }
    if (a || b < 0){
        cout << "true" <<
endl;
    }
}
```

```
        else{  
            cout << "false" <<  
endl;  
        }  
    }
```

Uso de bibliotecas

Funções específicas, disponibilizadas por outras bibliotecas da linguagem, podem ser necessárias para resolver (ou facilitar a resolução) um problema.

As bibliotecas mais comuns são a `cmath`, que contém funções matemáticas, a `locale`, que disponibiliza funções para formatos de idioma e moeda, e a `iomanip`, com funções que permitem manipular a entrada e saída de dados. As funções dessas e outras bibliotecas podem ser consultadas em <https://cplusplus.com/reference/library/>.

Para utilizar as funções de uma biblioteca, ela deve ser incluída no código, com o uso da diretiva `#include`, nas primeiras linhas do código, assim como é feito com a biblioteca `<iostream>`.

Biblioteca `<cmath>`

A biblioteca `cmath` disponibiliza funções matemáticas, tais como trigonométricas, exponenciais, potenciação e arredondamento.

Algumas funções aceitam e devolvem valores de diferentes tipos, conforme os valores que são fornecidos para o cálculo (parâmetros). Isso pode ser consultado na documentação da função. Veja, por exemplo, a função de potência (`pow`):

```
double pow (double base      , double exponent);  
float  pow (float  base      , float  exponent);
```

Se os valores da base e da potência para o cálculo forem do tipo `double`, o retorno será `double`; se os valores forem do tipo `float`, o resultado também será `float`. Veja o exemplo abaixo.

```
1 #include <iostream>  
2 #include <cmath>  
3  
4 using namespace std;  
5  
6 int main(){  
7     float b, e, p;  
8  
9     b = 4;  
10    e = 2;  
11    p = pow (b, e);  
12    cout << p << endl;  
13 }
```


A função `pow(b, e)` irá retornar a potência de `b` elevado a `e`, que será armazenada na variável `p`. Como `b` e `e` são do tipo `float`, a variável que irá receber o resultado também deve ser do tipo `float`.

Outra função da biblioteca `<cmath>` retorna a raiz quadrada de um número. Da mesma forma que a função `pow`, o resultado será do mesmo tipo que o valor fornecido para o cálculo da raiz.

```
double sqrt (double x);
float sqrt (float x);
```

No exemplo abaixo, o valor fornecido é um `int` (linha 11), mas a variável `raiz`, que recebe o cálculo da função `sqrt` deve ser `double` ou `float`. O tipo do valor fornecido deve ser comportado pelo tipo que a função espera.

```
1 #include <iostream>
2 #include <cmath>
3
4 using namespace std;
5
6 int main(){
7     int n;
8     double raiz;
9
10    n = 225;
11    raiz = sqrt (n);
12    cout << raiz << endl;
13 }
```

Formatação de saída de dados

Considere a necessidade de limitar as casas decimais de valores do tipo `float`. A instrução `setprecision`, da biblioteca `<iomanip>`, permite limitar o número de casas decimais. Também é preciso indicar que se espera a notação com ponto decimal, por meio da instrução `fixed`. Veja o exemplo a seguir.

```
1 #include <iostream>
2 #include <iomanip>
3
4 using namespace std;
5
6 int main(){
7     float pi;
8     pi = 3.14159265359;
9     cout << pi << endl;
10    cout << fixed << setprecision(8) << pi << endl;
11    cout << setprecision(2) << pi << endl;
```

```
12     cout << setprecision(4) << pi << endl;  
13 }
```

Bibliografia

"Biblioteca Padrão do C." 2020. Biblioteca Padrão do C.

https://pt.wikipedia.org/wiki/Biblioteca_padr%C3%A3o_do_C.

"C++ Language." 2000. C++ Tutorials. <http://www.cplusplus.com/doc/tutorial/>.

"C Library." 2000. C Library. <http://www.cplusplus.com/reference/clibrary/>.

"C Tutorial." 2021. Tutorials Point. <https://www.tutorialspoint.com/cprogramming/index.htm>.

"Programação de Computadores." 2021. Wikipedia.

https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_de_computadores.

"Programming Language." 2021. Wikipedia. https://en.wikipedia.org/wiki/Programming_language.

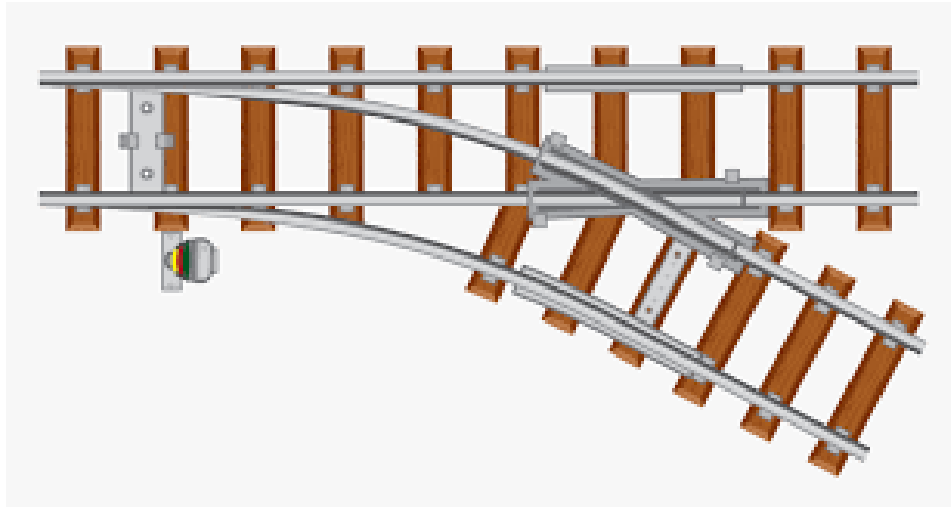


Exercícios

São recomendados os seguintes problemas da plataforma Beecrowd:

- 1004 - Produto Simples
- 1930 - Tomadas
- 2374 - Pneu
- 1003 - Soma Simples
- 1007 - Diferença
- 1016 - Distância
- 1020 - Idade em Dias
- 1018 - Cédulas
- 1010 - Cálculo Simples
- 1006 - Média 2
- 1002 - Área do Círculo





Estruturas de Seleção

Desvio de fluxo de execução

Todos os algoritmos elaborados até aqui seguem a estrutura sequencial, na qual as instruções são executadas uma a uma, da primeira até a última (desde que estejam sintaticamente corretas). No entanto, há problemas que requerem que algumas instruções sejam executadas ou não, dependendo de determinadas condições.

As **estruturas de seleção** (ou estruturas condicionais) permitem o tratamento de decisões em um programa. Essas estruturas permitem que diferentes ações sejam tomadas a partir da avaliação de uma **expressão lógica** (ou condição).

Expressões lógicas

Uma **expressão lógica** (ou booleana) resulta em um valor lógico (**verdadeiro ou falso**). O valor pode ser resultante de uma constante lógica, de uma variável booleana, de uma função booleana, uma expressão relacional ou de um conjunto desses elementos, unidos por meio de **operadores lógicos**.

São exemplos de expressões lógicas:

$5 + 3 = 4 + 4$	$x1 > 0 \text{ e } x2 > 0$	$a > b \text{ e } a > c \text{ e } b > c$
$3 * x > (x / 2) * 5$	$x1 \neq 0 \text{ ou } x2 \neq 0$	$(a + b) / 2 \geq \text{media}$
$a < b$	$x1 <> 0 \text{ ou } x2 <> 0$	$\text{num} \leq 100 \text{ e } \text{op} = 's'$

O valor de uma expressão lógica é calculado da esquerda para a direita. Assim que o valor da expressão é definido, a avaliação é interrompida. Com isso, muitas vezes a

última parte de uma expressão pode não ser avaliada (Feofiloff 2009). Como exemplo, considere as variáveis `comorbidade` (tipo lógico), `idade` (tipo inteiro) e `limite` (tipo inteiro) e a seguinte expressão lógica:

```
comorbidade ou idade >= limite
```

Apenas se `comorbidade` for falso é que a segunda parte (referente à `idade`) será avaliada. Nesse caso, a inversão das partes da expressão seria pouco significativa; no entanto, em situações em que a segunda parte da expressão possa estar atrelada à primeira, essa ordem pode representar a execução bem sucedida do algoritmo.

Os operadores relacionais têm precedência sobre os operadores lógicos. Observe a expressão:

```
a > b e a > c
```

Nesse caso, é avaliada a primeira parte (`a > b`), se for verdade, a segunda parte é avaliada (da esquerda para a direita) (`a > c`) e, por fim, é resolvido o operador lógico `e`.

Dentre os operadores lógicos, a ordem de prioridade é: `não`, `e`, `ou`. Se for necessário forçar a ordem dos operadores, devem ser usados parênteses.

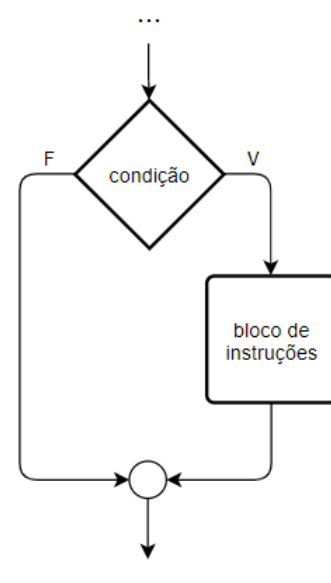
Estruturas de seleção simples e compostas

Em uma **estrutura de seleção simples** (`se...então...fim_se`), há um bloco a ser executado apenas quando o resultado da condição é **verdadeiro**; caso a condição seja falsa, a execução continuará a partir da próxima instrução depois do fim da estrutura. Segue a representação em pseudocódigo e fluxograma.

Pseudocódigo

```
1 Algoritmo "Exemplo selecao simples"
2 // Descrição : Estrutura de selecao simples
3 // Autor(a) : aetp
4 // Data atual: 8/17/2021
5 Var
6   ...
7
8 Inicio
9   ...
10  se (condicao) entao
11    ...
12    ...
13  fimse
14  ...
15 Fimalgoritmo
```

Fluxograma



No pseudocódigo, observe que as instruções dentro do bloco devem estar mais à direita do que as demais instruções. No fluxograma, veja a indicação dos fluxos (V e F) a partir da condição; por padrão, utilize o bloco para a condição verdadeira sempre à direita. Também observe que o conector (círculo) encerra o bloco da seleção (deve estar alinhado ao losango que representa a estrutura de seleção), equivalendo ao `fimse` do pseudocódigo.

Em uma **estrutura de seleção composta** (`se...então...senão...fim_se`), há um bloco de instruções para a ser executado caso a condição seja **verdadeira** e outro bloco caso a condição seja **falsa**. Segue a representação em fluxograma e pseudocódigo.

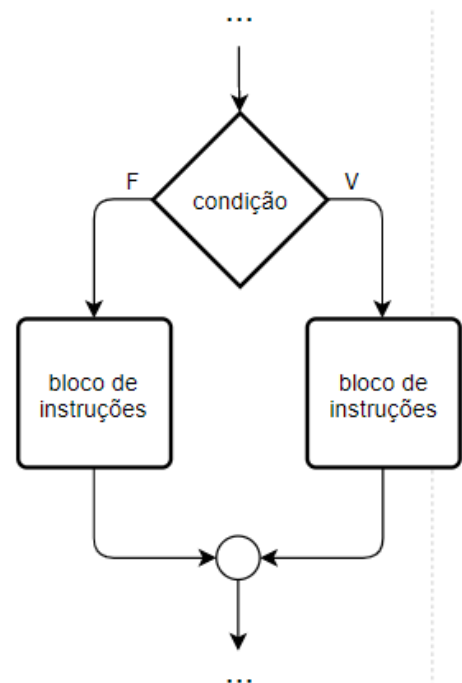
Pseudocódigo

```

1 Algoritmo "Exemplo selecao composta"
2 // Descrição : Estrutura de selecao composta
3 // Autor(a) : aetp
4 // Data atual: 8/17/2021
5 Var
6     ...
7
8 Inicio
9     ...
10    se (condicao) entao
11        ...
12    senao
13        ...
14    fimse
15    ...
16 Fimalgoritmo

```

Fluxograma



Teste de mesa de estruturas de seleção

Para fazer o **teste de mesa** de uma estrutura de seleção, recomendamos indicar as condições em uma das colunas da tabela, entre parênteses e com um sinal de interrogação (veja abaixo). Essa coluna receberá os valores da avaliação da condição para cada caso de teste (V ou F).

Em algumas situações, é adequado um teste de mesa exaustivo, que consiste em explorar todos os fluxos de execução possíveis de um algoritmo.

#	Entrada			Processamento		Saída
	var_1	...	var_x	var_p	(condição)?	
1					V	
2					F	
3					...	

Estrutura de seleção em C/C++

Observe a sintaxe de uma estrutura de seleção em C/C++:

```

1  ...
2  if (condicao){
3      //bloco de instruções para quando a condição for verdadeira
4  }
5  else{
6      //bloco de instruções para quando a condição for falsa
7  }
8  ...

```

Cada bloco é demarcado por chaves. Veja que o bloco do `if` inicia na linha 2 (`{`) e é finalizado (`}`) na linha 4, enquanto o bloco do `else` inicia na linha 5 e acaba na linha 7. Observe a indentação dos blocos, as instruções das linhas 3 e 6 estão mais à direita das demais.

Uma estrutura de seleção simples requer apenas o bloco do `if` (linhas 2 a 4), omitindo a instrução `else`.

Para a sintaxe dos operadores relacionais e lógicos, veja o quadro dos operadores disponibilizado previamente. Para garantir a precedência da avaliação, devem ser utilizados parênteses. Seguem alguns exemplos de condições usando a sintaxe da linguagem:

<code>a + b == 100</code>	<code>x1 > 0 && x2 > 0</code>	<code>a > b && a > c && b > c</code>
<code>3 * x > (x/2) * 5</code>	<code>x1 != 0 x2 != 0</code>	<code>a > b (a > c && b > c)</code>
<code>a < b</code>	<code>(a + b)/2 >= media</code>	<code>num <= 100 && op == 's'</code>

Exemplo 1 - Aprovação

Considere o seguinte problema:

O programa deve informar se o aluno foi aprovado, ou seja, obteve média maior ou igual a 7. A média (aritmética) deve ser calculada a partir de duas notas.

Veja o pseudocódigo, o fluxograma, o teste de mesa e a implementação em C/C++.

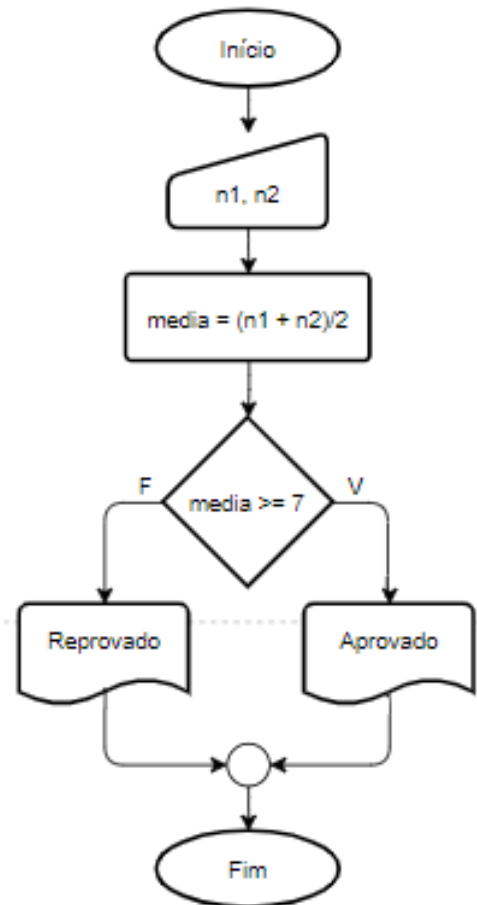
Pseudocódigo

```

1 Algoritmo "Calcula media"
2 // Descrição : Exemplo de uso de selecao
3 // Autor(a) : aetp
4 // Data atual: 8/17/2021
5 Var
6   n1, n2 : inteiro
7   media : real
8 Inicio
9   leia (n1, n2)
10  media <- (n1 + n2)/2
11  se (media >= 7) entao
12    escreva ("Aprovado")
13  senao
14    escreva ("Reprovado")
15  fimse
16 Fimalgoritmo

```

Fluxograma



Teste de mesa

#	Entrada		Processamento		Saída
	n1	n2	media	(media >= 7)?	
1	6	5	5.5	F	Reprovado
2	8	8	8	V	Aprovado
3	5	9	7	V	Aprovado

```

1 #include <iostream>
2
3 using namespace std;
4
5 int main(){
6     int n1, n2;
7     float media;
8
9     cin >> n1 >> n2;
10    media = (n1 + n2)/2;
11    if (media >= 7){
12        cout << "Aprovado" << endl;
13    }
14    else{
15        cout << "Reprovado" << endl;
16    }
17 }

```

Exercícios propostos (Manzano and Oliveira 2010, Cap 4)

→ Determine o resultado lógico das expressões, considerando os seguintes valores: X = 1, A = 3, B = 5, C = 8 e D = 7:

- a) .não. (X > 3)
- b) (X < 1) .e. .não. (B > D)
- c) .não. (D < 0) .e. (C > 5)
- d) .não. (X > 3) .ou. (C < 7)
- e) (A > B) .ou. (C > B)
- f) (X >= 2)
- g) (X < 1) .e. (B >= D)
- h) (D < 0) .ou. (C > 5)
- i) .não. (D > 3) .ou. .não. (B > 7)
- j) (A > B) .ou. .não. (C > B)

→ Indique qual instrução será executada (1 ou 2) a partir da avaliação da expressão lógica, considerando os seguintes valores: A = 2, B = 3, C = 5 e D = 9. Não é necessário calcular os valores da variável X.

a)

```
se .não. (D > 5) então
    X = (A + B) * D; //1
senão
    X = (A - B) / C; //2
fim_se;
```

b)

```
se (A > 2) .e. (B < 7) então
    X = (A + 2) * (B - 2); //1
senão
    X = (A + B) / D * (C + D); //2
fim_se;
```

c)

```
se (A = 2) .ou. (B < 7) então
    X = (A + 2) * (B - 2); //1
senão
    X = (A + B) / D * (C + D); //2
fim_se;
```

d)

```
se (A > 2) .ou. .não. (B < 7) então
    X = A + B - 2; //1
senão
    X = A - B; //2
fim_se;
```

e)

```
se .não. (A > 2) .ou. .não. (B < 7) então
    X = A + B; //1
senão
    X = A / B; //2
fim_se;
```

f)

```
se .não. (A > 3) .e. .não. (B < 5) então
    X = A + D; //1
senão
    X = D / B; //2
fim_se;
```

Exemplo 2 - Adivinhação

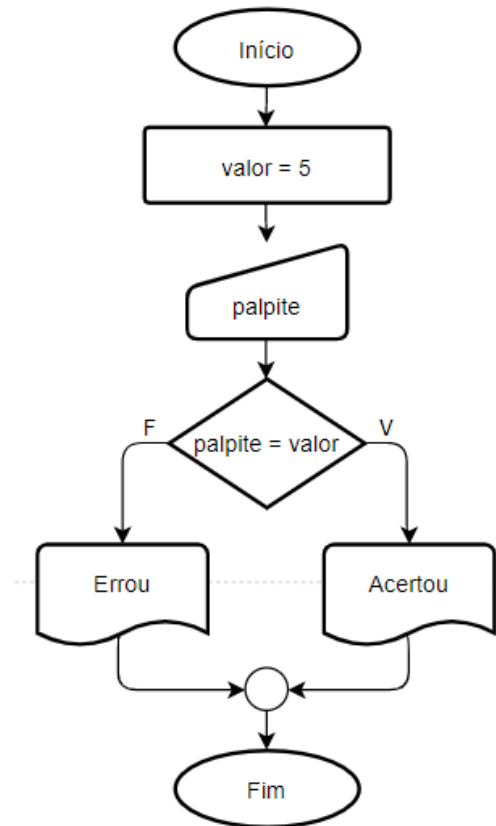
Pseudocódigo

```

1 Algoritmo "Adivinhacao"
2 // Descrição : Exemplo de selecao composta
3 // Autor(a) : aetp
4 // Data atual: 8/18/2021
5 Var
6     palpite, valor : inteiro
7 Inicio
8     valor <- 5
9     leia (palpite)
10    se (palpite = valor) entao
11        escreva ("Acertou")
12    senao
13        escreva ("Errou")
14    fimse
15 Fimalgoritmo

```

Fluxograma



Teste de mesa

#	Entrada	Processamento		Saída
	palpite	valor	(palpite = valor)?	
1	6	5	F	Errou
2	5	5	V	Acertou
3	423	5	F	Errou

```

1 #include <iostream>
2 using namespace std;
3
4 int main(){
5     int palpite, valor = 5;
6
7     cin >> palpite;
8     if (palpite == valor){
9         cout << "Acertou" << endl;
10    }
11    else{
12        cout << "Errou" << endl;
13    }
14 }

```

Exemplo 3 - Maior de dois valores

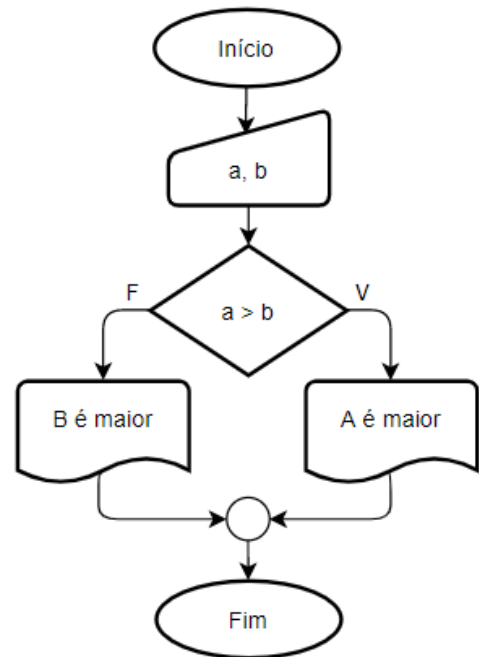
Pseudocódigo

```

1 Algoritmo "Maior de dois"
2 // Descrição : Exemplo de uso de selecao
3 // Autor(a) : aetp
4 // Data atual: 8/18/2021
5 Var
6   a, b : inteiro
7 Inicio
8   leia (a, b)
9   se (a > b) entao
10      escreva ("A é maior")
11   senao
12      escreva ("B é maior")
13   fimse
14 Fimalgoritmo

```

Fluxograma



Implementação em C/C++

```

1 #include <iostream>
2
3 using namespace std;
4
5 int main(){
6     int a, b;
7
8     cin >> a >> b;
9     if (a > b){
10         cout << "A é maior" << endl;
11     }
12     else{
13         cout << "B é maior" << endl;
14     }
15 }

```

Teste de mesa

#	Entrada		Processamento	Saída
	a	b	(a > b)?	
1	6	5	V	A é maior
2	5	6	F	B é maior
3	5	5	F	B é maior



Observe a última instância do teste de mesa... A resposta está adequada?

Como corrigir esse problema, a fim de obter uma resposta semanticamente correta?

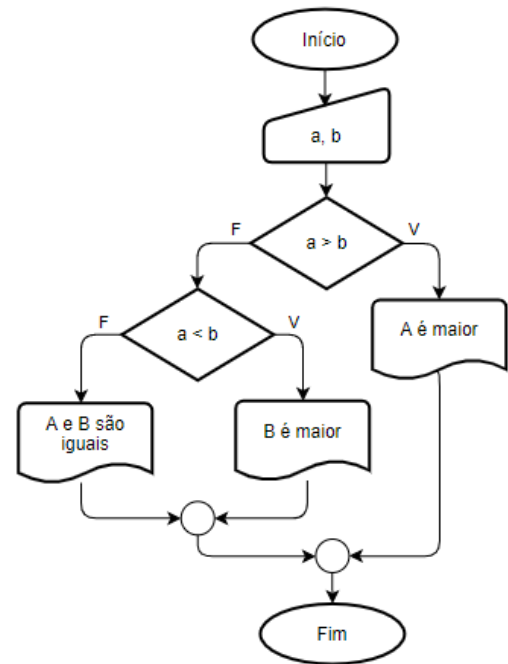
Pseudocódigo

```

1 Algoritmo "Maior de dois"
2 // Descrição : Exemplo de uso de selecao
3 // Autor(a) : aetp
4 // Data atual: 8/18/2021
5 Var
6   a, b : inteiro
7 Inicio
8   leia (a, b)
9   se (a > b) entao
10    escreva ("A é maior")
11  senao
12    se (a < b) entao
13     escreva ("B é maior")
14    senao
15     escreva ("A e B sao iguais")
16    fimse
17  fimse
18 Fimalgoritmo

```

Fluxograma



Implementação em C/C++

```

1 #include <iostream>
2 using namespace std;
3
4 int main(){
5     int a, b;
6     cin >> a >> b;
7     if (a > b){
8         cout << "A é maior" << endl;
9     }
10    else{
11        if (a < b){
12            cout << "B é maior" << endl;
13        }
14        else{
15            cout << "A e B são iguais" << endl;
16        }
17    }
18 }

```

Teste de mesa

#	Entrada		Processamento		Saída
	a	b	(a > b)?	(a < b)?	
1	6	5	V	-	A é maior
2	5	6	F	V	B é maior
3	5	5	F	F	A e B são iguais



Estruturas de seleção aninhadas (*Nested if statements*)

Algumas soluções requerem o uso de estruturas de seleção dentro de outras. Uma ou mais estruturas de seleção podem ser aninhadas ("C Tutorial" 2021) (ou encadeadas (Forbellone and Eberspächer 2005)).

No Exemplo 3, esse tipo de estrutura deve ser usada porque apenas uma das condições deve ser satisfeita (A é maior, ou B é maior, ou são iguais; apenas uma afirmação é verdadeira). Observe no fluxograma e nos fragmentos do pseudocódigo e da implementação os limites das estruturas mais internas.

Pseudocódigo

```

9   se (a > b) entao
10      escreva ("A é maior")
11   senao
12      se (a < b) entao
13         escreva ("B é maior")
14      senao
15         escreva ("A e B sao iguais")
16   fimse
17   fimse

```

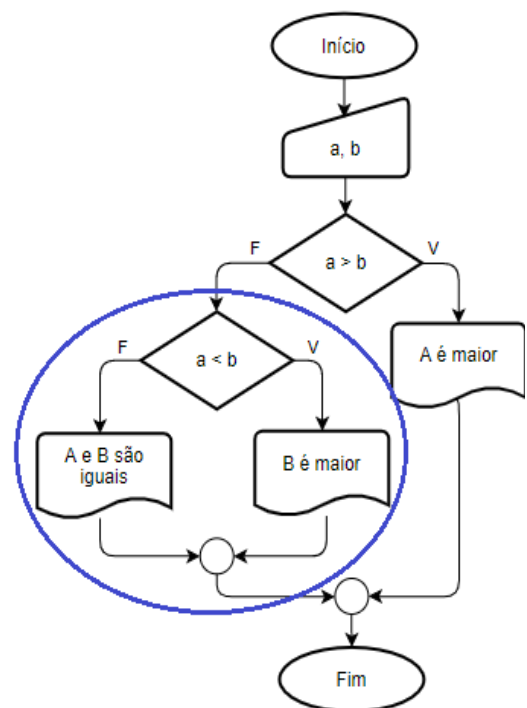
Implementação em C/C++

```

7   if (a > b){
8       cout << "A e maior" << endl;
9   }
10  else{
11      if (a < b){
12          cout << "B é maior" << endl;
13      }
14      else{
15          cout << "A e B são iguais" << endl;
16      }
17  }

```

Fluxograma



A condição da estrutura mais interna (linhas 12 a 16 no pseudocódigo, linhas 11 a 16 em C/C++) somente será avaliada quando a condição da estrutura mais externa (linha 9 no pseudocódigo, linha 7 em C/C++) for falsa. Essa situação é fácil de ser observada no fluxograma.

Há algoritmos em que é possível identificar um padrão no encadeamento das estruturas (encadeamento homogêneo), enquanto em outros não (encadeamento heterogêneo) (Forbellone and Eberspächer 2005).

Exercício proposto

Analise o pseudocódigo:

```

1 Algoritmo "Pre-fix"
2 // Descrição : Avaliação de estruturas de selecao
3 // Autor(a) : aetp
4 // Data atual: 8/23/2021
5 Var
6   op: character
7   a, b, r : real
8 Inicio
9   leia (op)
10  se (op = "+" ) ou (op = "-" ) ou (op = "*" ) ou (op = "/" ) entao
11    leia (a, b)
12    se (op = "+") entao
13      r <- a + b
14    senao
15      se (op = "-") entao
16        r <- a - b
17      senao
18        se (op = "*") entao
19          r <- a * b
20        senao
21          se (op = "/" ) entao
22            r <- a / b
23          fimse
24        fimse
25      fimse
26    fimse
27    escreva (a, op, b, " = ", r)
28  senao
29    leia (a)
30    se (op = "sqr") entao
31      r <- a^(1/2)
32    senao
33      se (op = "^3") entao
34        r <- a^3
35      senao
36        se (op = "^2") entao
37          r <- a^2
38        fimse
39      fimse
40    fimse
41    escreva (a, op, " = ", r)
42  fimse
43 Fimalgoritmo

```

→ Elabore o fluxograma para esse algoritmo.

→ Complete o teste de mesa do algoritmo Pré-fix:

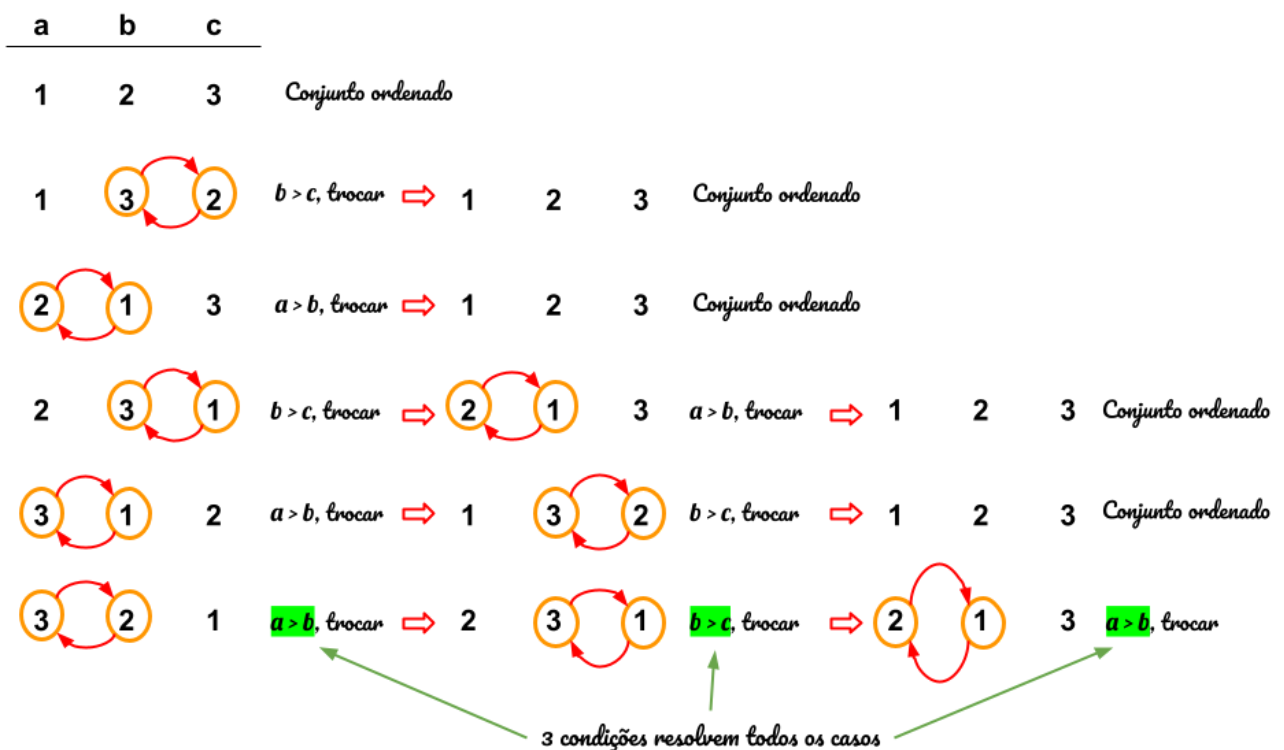
#	Entrada			Processamento									Saída
	op	a	b	L10?	L12?	L15?	L18?	L21?	L30?	L33?	L36?	r	
1	+	5	3	V	V	-	-	-	-	-	-	8	$5 + 3 = 8$
2	-	22	6	V	F	V	-	-	-	-	-	16	$22 - 6 = 16$
3	*												
4													
5													
6													
7													

→ Descreva a funcionalidade do algoritmo Pré-fix.

Considere as boas práticas para a documentação de um algoritmo (Feofiloff 2009), como um mini manual contendo as entradas, saídas e a relação entre os dados fornecidos e a saída.

Exemplo 4 - Ordenação de três valores numéricos

Para três valores, quais as combinações possíveis? Quais trocas são necessárias para ordenar cada um desses conjuntos?



Pseudocódigo

```

1 Algoritmo "Ordena"
2 // Descrição : Estruturas de selecao alinhadas
3 // Autor(a) : aetp
4 // Data atual: 8/24/2021
5 Var
6   a, b, c, aux : inteiro
7 Inicio
8   leia (a, b, c)
9   se (a > b) entao
10      aux <- a
11      a <- b
12      b <- aux
13   fimse
14   se (b > c) entao
15      aux <- b
16      b <- c
17      c <- aux
18   fimse
19   se (a > b) entao
20      aux <- a
21      a <- b
22      b <- aux
23   fimse
24   escreva (a, b, c)
25 Fimalgoritmo

```

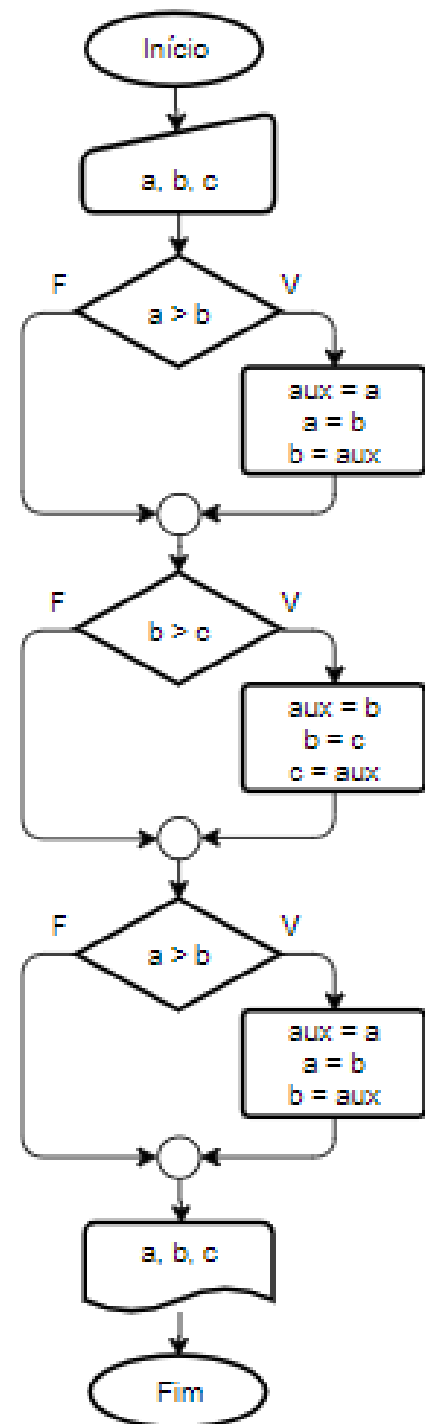
Implementação em C/C++

```

1 #include <iostream>
2 using namespace std;
3
4 int main(){
5     int a, b, c, aux;
6     cin >> a >> b >> c;
7     if (a > b){
8         aux = a;
9         a = b;
10        b = aux;
11    }
12    if (b > c){
13        aux = b;
14        b = c;
15        c = aux;
16    }
17    if (a > b){
18        aux = a;
19        a = b;
20        b = aux;
21    }
22    cout << a << " " << b << " " << c << endl;
23 }

```

Fluxograma



Teste de mesa

#	Entrada			Processamento				Saída
	a	b	c	aux	(a > b)?	(b > c)?	(a > b)?	
1	1	2	3		F	F	F	1 2 3
2	1	3 2	2 3	3	F	V	F	1 2 3
3	2 1	3 4 2	4 3	3 2	F	V	V	1 2 3
4	2 1	4 2	3	2	V	F	F	1 2 3
5	3 1	4 3 2	2 3	3 3	V	V	F	1 2 3
6	3 2 1	2 3 4 2	4 3	3 3 2	V	V	V	1 2 3

Estruturas de seleção alinhadas (*if statements*)

As estruturas de seleção alinhadas são usadas quando várias condições devem ser verificadas, independente do resultado das demais.

Considere o Exemplo 4. O que aconteceria se a estruturas das linhas 14 e 19 (no pseudocódigo) e 12 e 17 (em C/C++) estivessem aninhadas? A funcionalidade seria mantida?

É necessário encerrar cada estrutura antes que uma nova seja iniciada (*fimse* ou *}*). O alinhamento também deve ser observado, pois além do aspecto estético, contribui para a compreensão do algoritmo. Veja a implementação abaixo como exemplo.

```

1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      int a, b, c;
6      cin >> a >> b >> c;
7      if (a % 2 == 0 && a % 3 == 0){
8          cout << a << endl;
9      }
10     if (b % 2 == 0 && b % 3 == 0){
11         cout << b << endl;
12     }
13     if (c % 2 == 0 && c % 3 == 0){
14         cout << c << endl;
15     }
16 }
```

Veja o algoritmo a seguir, que compara as estruturas alinhadas e aninhadas.

```
1 algoritmo "Estrutura alinhada e aninhada"
2 // Descrição : Testa estruturas de selecao
3 // Autor(a) : aetp
4 // Data atual: 8/25/2021
5 Var
6     a, b, c, d : logico
7 inicio
8     a <- Falso
9     b <- Falso
10    c <- Verdadeiro
11    d <- Verdadeiro
12    //Estrutura alinhada
13    escreva ("Alinhada: ")
14    se (a) entao
15        escreva ("A")
16    fimse
17    se (b) entao
18        escreva ("B")
19    fimse
20    se (c) entao
21        escreva ("C")
22    fimse
23    se (d) entao
24        escreva ("D")
25    fimse
26    escreval ("")
27    //Estrutura aninhada
28    escreva ("Aninhada: ")
29    se (a) entao
30        escreva ("A")
31    senao
32        se (b) entao
33            escreva ("B")
34        senao
35            se (c) entao
36                escreva ("C")
37            senao
38                se (d) entao
39                    escreva ("D")
40                fimse
41            fimse
42        fimse
43    fimse
44 fimalgoritmo
```

Expressão condicional (if ternário)

A linguagem C/C++ fornece uma alternativa para escrever uma estrutura de seleção de forma mais concisa⁴, comumente chamada `if` ternário. Essa forma é denominada **expressão condicional** porque é uma condição que pode ser usada em qualquer lugar onde uma expressão é esperada.

A sintaxe é expressa pela condição seguida por `?` (denominado **operador condicional**), a instrução a ser executada caso a condição seja verdadeira separada por dois pontos (`:`) da instrução a ser executada caso a condição seja falsa ("C++ Language" 2000).

Estrutura de seleção

```
if (a < b){
    min = a;
}
else{
    min = b;
}
```

Expressão condicional

```
min = (a < b? a : b);
```

A diferença é que a expressão condicional deve ter uma instrução (e apenas uma) em cada caso (verdadeiro ou falso), e as duas possibilidades devem estar explicitadas. Da mesma forma que na estrutura de seleção, a condição da expressão pode ser composta (com operadores lógicos).

Veja alguns exemplos de uso do `if` ternário.

```
cout << (x % 2 == 0? "Par" : "Impar");

x % 2 == 0? cout << "Par": cout << "Impar";

r = (total != 0? soma/total : 0);

total != 0? r = soma/total : r = 0;

x = x < 0 ? x * (-1) : x;

x < 0 ? x = x * (-1) : x = x;
```

⁴ Esse é apenas um recurso da linguagem, não afeta em nada as demais formas de representação de uma solução algorítmica.

Exercícios propostos

Analise os pseudocódigos, elabore os fluxogramas e responda às questões:

I.

```

01 Algoritmo "Teste_1"
02 Var
03   A, B, C : logico
04
05 Inicio
06   I1
07   se A entao
08     se B entao
09       I2
10     senao
11       I3
12   fimse
13   se C entao
14     I4
15   senao
16     I5
17   fimse
18   I6
19   senao
20     I7
21   fimse
22 FimAlgoritmo

```

- Independente dos valores das variáveis, qual(is) instrução(ões) será(ão) sempre executada(s)?
- Supondo que I1 corresponde a um bloco com as seguintes instruções:
A = V, B = V, C = F
Quais instruções serão executadas?
- Supondo que I1 corresponde a um bloco com as seguintes instruções:
A = F, B = V, C = V
Quais instruções serão executadas?
- Quais os valores das variáveis A, B e C para que as instruções I1, I3, I5 e I6 sejam executadas?
- Quais os valores das variáveis A, B e C para que as instruções I1, I6 e I7 sejam executadas?

II.

```

01 Algoritmo "Teste_2"
02 Var
03   A, B, C, D : logico
04
05 Inicio
06   I1
07   se A entao
08     I2
09     se (B e C) entao
10       I3
11     se D entao
12       I4
13     fimse
14   senao
15     se nao D entao
16       I5
17     fimse
18   fimse
19   I6
20   fimse
21   I7
22 FimAlgoritmo

```

- Supondo que I1 equivale a um bloco com as seguintes instruções:
A = V, B = F, C = F, D = V
Quais instruções, a partir da condição da linha 7, serão executadas?
- Supondo que I1 equivale a um bloco com as seguintes instruções:
A = V, B = V, C = V, D = F
Quais instruções, a partir da condição da linha 7, serão executadas?
- Se A = V, quais os valores de B, C e D para que a instrução I5 seja executada?
- Quais condições devem ser avaliadas para que a instrução I5 seja executada?
- Quando a instrução I4 é executada, quais outras instruções também são executadas?

III.

```
01 Algoritmo "Teste_3"
02 Var
03   A, B : logico
04   X, Y : inteiro
05 Inicio
06   I1
07   se A entao
08     I2
09     se B e (X < Y) entao
10       I3
11     senao
12       I4
13     fimse
14   senao
15     se B e (X > Y) entao
16       I5
17     senao
18       I6
19     fimse
20   I7
21   fimse
22 FimAlgoritmo
```

- a) Supondo que I1 equivale a um bloco com as seguintes instruções:
A = F, B = V, X = 5, Y = 2.
Quais instruções, a partir da condição da linha 7, serão executadas?
- b) Supondo que I1 equivale a um bloco com as seguintes instruções:
A = F, B = F, X = 2, Y = 5
Quais instruções, a partir da condição da linha 7, serão executadas?
- c) Quais condições serão testadas para que a instrução I6 seja executada?
- d) Se B = V, quais os valores de A, X e Y para que a instrução I4 seja executada? Quais outras instruções serão executadas nesse caso?
- e) Quais linhas do pseudocódigo devem ser alteradas caso a instrução I7 tenha que ser sempre executada?

Seleção múltipla

Quando há várias estruturas de seleção aninhadas, pode ser utilizada uma seleção de múltipla escolha. A instrução **escolha-caso** é uma opção mais enxuta quando há condições simples, em que apenas uma variável, do tipo inteiro ou caractere, é avaliada ("Switch statement" 2021).

Pseudocódigo

```

1 algoritmo "Estrutura selecao multipla
2 // Função: Calculadora aritmética simples
3 // Autor : aetp
4 // Data : 09/06/2021
5 var
6   op : caractere
7   a, b, r : real
8 inicio
9   leia (op, a, b)
10  escolha (op)
11    caso "+"
12      r <- a + b
13    caso "-"
14      r <- a - b
15    caso "*"
16      r <- a * b
17    caso "/"
18      r <- a / b
19    outrocaso
20      r <- 0
21  fimescolha
22  escreva (a, op, b, " = ", r)
23 fimalgoritmo

```

Implementação em C/C++

```

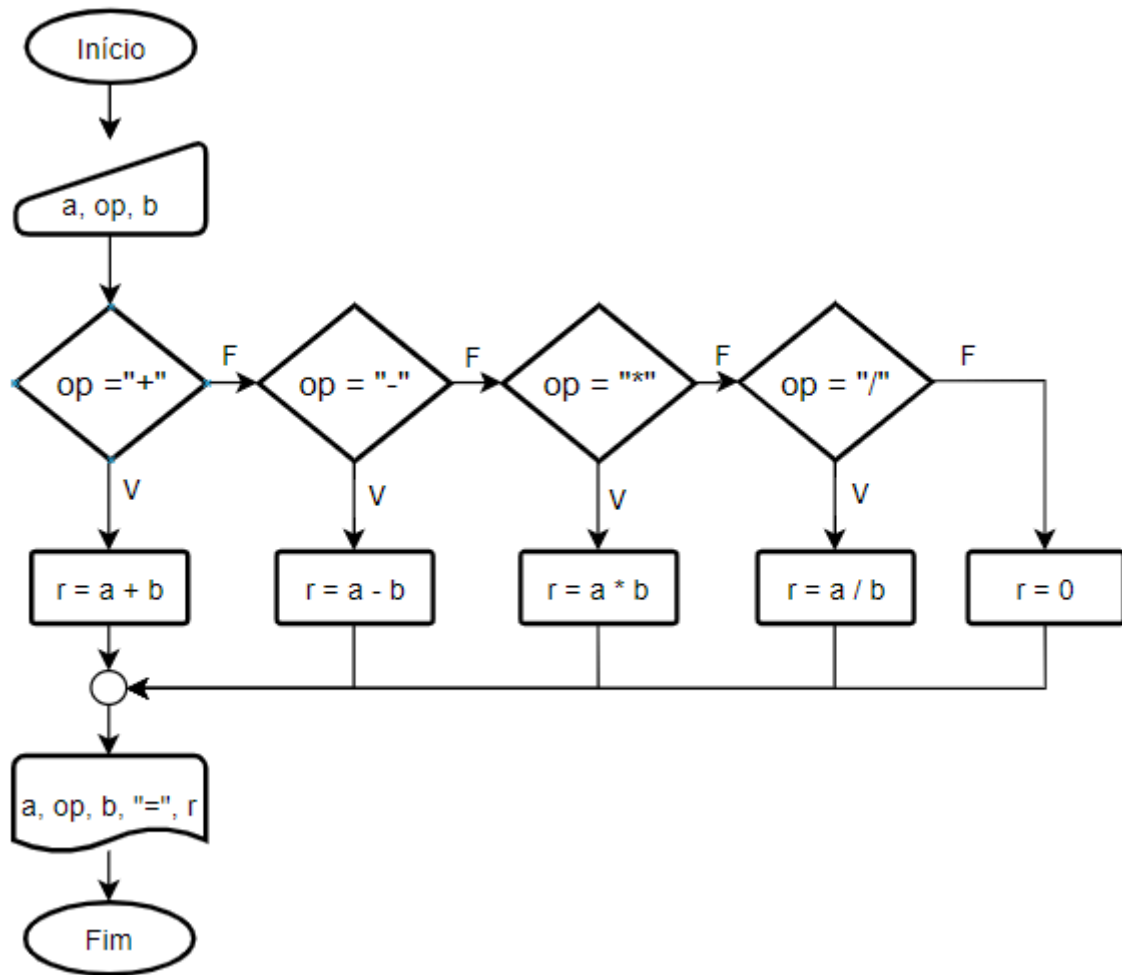
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5   char op;
6   float a, b, r;
7
8   cin >> op >> a >> b;
9   switch (op){
10    case '+':
11      r = a + b;
12      break;
13    case '-':
14      r = a - b;
15      break;
16    case '*':
17      r = a * b;
18      break;
19    case '/':
20      r = a / b;
21      break;
22    default:
23      r = 0;
24  }
25  cout << a << op << b
26       << " = " << r << endl;
27 }

```

Teste de mesa

#	Entrada			Processamento					Saída
	op	a	b	op = "+"	op = "-"	op = "*"	op = "/"	r	
1	+	5	3	V	-	-	-	8	5 + 3 = 8
2	-	22	6.5	F	V	-	-	15.5	22 - 6.5 = 15.5
3	*	4	5	F	F	V	-	20	4 * 5 = 20
4	/	9	4	F	F	F	V	2.25	9 / 4 = 2.25
5	&	3	9	F	F	F	F	0	3 & 9 = 0

Fluxograma



Para variáveis inteiras, é possível verificar um intervalo. Atenção à sintaxe, pois as reticências devem estar separadas dos valores por espaços ("Switch Statement in C/C++" 2021). Veja o exemplo a seguir:

Problema: Dada a idade de um esportista, classifique-o em uma das categorias:

Idade	Categoria
5 até 7 anos	Infantil A
8 até 10 anos	Infantil B
11 até 13 anos	Juvenil A
14 até 17 anos	Juvenil B
Maiores de 18 anos	Sênior

```

1 #include <iostream>
2 using namespace std;
3
4 int main(){
5     int idade;
6
7     cin >> idade;
8     switch (idade){
9         case 5:
10        case 6:
11        case 7:
12            cout << "Infantil A" << endl;
13            break;
14        case 8 ... 10:
15            cout << "Infantil B" << endl;
16            break;
17        case 11 ... 13:
18            cout << "Juvenil A" << endl;
19            break;
20        case 14 ... 17:
21            cout << "Juvenil B" << endl;
22            break;
23        default:
24            cout << "Senior" << endl;
25            break;
26    }
27 }

```

Exercícios propostos

Desenvolva os pseudocódigos, os fluxogramas, a implementação e o teste de mesa:

- I. Calcule o valor a ser pago por um produto, considerando o preço de etiqueta e a condição de pagamento. Utilize os códigos do quadro abaixo para ler a condição de pagamento e efetuar o cálculo adequado.

Código	Condição de pagamento
1	À vista em dinheiro ou cheque, recebe 10% de desconto
2	À vista no cartão de crédito, recebe 5% de desconto
3	Em 2 vezes, preço normal da etiqueta
4	Em 3 vezes, preço normal da etiqueta mais juros de 10%

II. Calcule a quantidade total de calorias de uma refeição a partir da escolha do cliente. O cliente deverá informar o prato, a sobremesa e a bebida, conforme a tabela:

Prato		Sobremesa		Bebida	
Vegetariano	180 cal	Abacaxi	75 cal	Chá	20 cal
Peixe	230 cal	Sorvete diet	110 cal	Suco de laranja	70 cal
Frango	250 cal	Mousse diet	170 cal	Suco de melão	100 cal
Carne	350 cal	Mousse chocolate	200 cal	Refrigerante diet	65 cal

Exemplo 5 - Cálculo da comanda

Calcule o preço de uma comanda para uma lanchonete, conforme o cardápio abaixo. Um cliente pode optar por **somente uma** promoção ou montar **somente um** combo com uma unidade de sanduíche, uma unidade de bebida e uma unidade de sobremesa. Entretanto, em um combo, nem sempre o cliente escolherá os três itens. Neste caso, a comanda conterà o valor "0" para o item.

Código	Sanduíche		Bebida		Sobremesa		Promoção	
1	X Salada	5,00	Água	2,00	Casquinha	3,00	X Picanha + Suco	11,00
2	X Frango	6,00	Milk Shake	7,00	Torta	3,50	X Salada + Milk Shake	10,00
3	X Picanha	8,50	Refrigerante	2,50			X Bacon + Refrigerante	7,50
4	X Bacon	6,00	Suco	3,50				

Entrada: quatro valores inteiros com os códigos, representando a promoção (P) (1 para promo, 0 para combo) e os itens escolhidos (OP1, OP2, OP3). No caso da escolha da promoção (P=1), só a primeira opção será considerada.

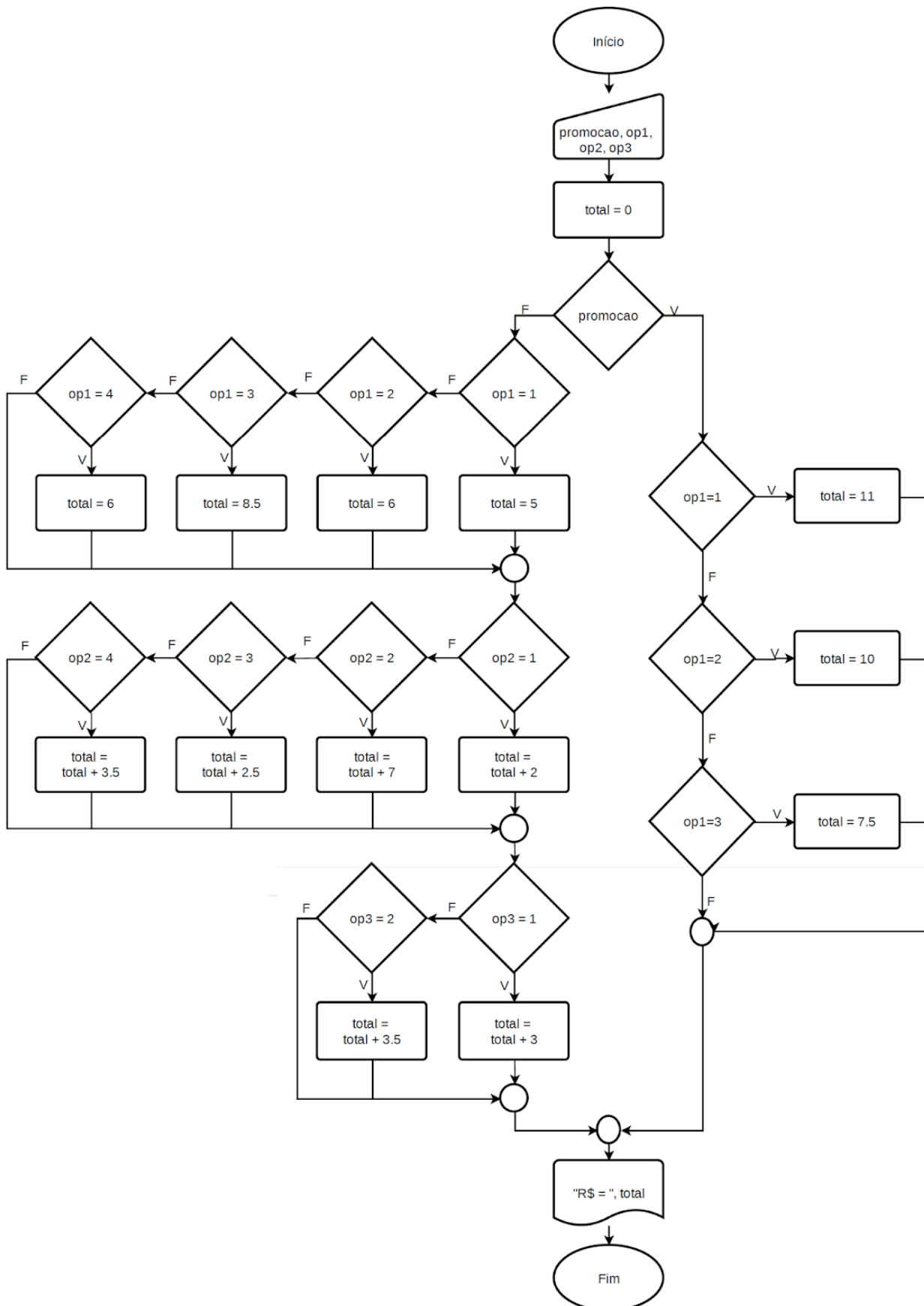
Saída: um valor real, com duas casas decimais.

Exemplo de entrada	Exemplo de saída
1 2 0 0	R\$ 10.00
0 0 2 2	R\$ 10.50
0 2 3 1	R\$ 11.50

Implementação

```
1 #include <iostream>
2 #include <cstdbool>
3 #include <iomanip>
4
5 using namespace std;
6
7 int main(){
8     bool promocao;
9     int op1, op2, op3;
10    double total = 0;           // inicialização da variável
11
12    cin >> promocao;           // 0 para combo, 1 para promoção
13    cin >> op1 >> op2 >> op3; // Itens do cardápio
14    if (promocao) {             // É promoção?
15        switch (op1) {         // qual é a promoção?
16            case 1 : total = 11;
17            break;
18            case 2 : total = 10;
19            break;
20            case 3 : total = 7.5;
21        }
22    } else {                     // É Combo?
23        switch (op1) {         // Qual é o sanduiche?
24            case 1 : total = 5;
25            break;
26            case 2 : total = 6;
27            break;
28            case 3 : total = 8.5;
29            break;
30            case 4 : total = 6.0;
31        }
32        switch (op2) {         // Qual é a bebida?
33            case 1 : total += 2; // Equivale a total = total + 2;
34            break;
35            case 2 : total += 7;
36            break;
37            case 3 : total += 2.5;
38            break;
39            case 4 : total += 3.5;
40        }
41        switch (op3) {         // Qual é a sobremesa?
42            case 1 : total += 3;
43            break;
44            case 2 : total += 3.5;
45        }
46    }
47    cout << fixed << setprecision(2);
48    cout << "R$ " << total << endl;
49    return 0;
50 }
```

Fluxograma



Bibliografia

"C++ Language." 2000. C++ Tutorials. <http://www.cplusplus.com/doc/tutorial/>.

Feofiloff, Paulo. 2009. *Algoritmos em Linguagem C*. Rio de Janeiro: Elsevier.

Forbellone, André Luiz V., and Henri F. Eberspächer. 2005. *Lógica de Programação: A construção de algoritmos e estruturas de dados*. 3ª ed. São Paulo: Pearson.

Manzano, José Augusto N., and Jayr F. Oliveira. 2010. *Algoritmos: Lógica para Desenvolvimento de Programação de Computadores*. São Paulo: Érica.

"Switch statement." 2021. Wikipedia. https://en.wikipedia.org/wiki/Switch_statement.

"Switch Statement in C/C++." 2021. GeeksForGeeks. <https://www.geeksforgeeks.org/switch-statement-cc/>.



Exercícios

Implemente soluções para os seguintes problemas:

A. "Capicua: sequência de algarismos que permanece a mesma se lida na ordem direta ou inversa (p.ex., 13231)" (Fonte: Dicionário Eletrônico Houaiss, 2004).

Um capicua é também chamado número palíndromo.

Dado um número de 4 dígitos, identifique se é um capicua.

B. Dados quatro valores numéricos, identificar qual deles é o menor valor par.

C. O Índice de Massa Corporal ($IMC = \text{peso}/(\text{altura})^2$) indica a condição de peso de uma pessoa adulta. Elabore um algoritmo que calcule o IMC e mostre a condição da pessoa, conforme o quadro.

IMC	Condição
Abaixo de 18,5	Abaixo do peso
Entre 18,5 e 25	Peso normal
Entre 25 e 30	Acima do peso
Acima de 30	Obeso



<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC67200/>

<https://www.urionlinejudge.com.br/judge/pt/problems/view/2454>

biblioteca <climits>

INT_MAX;