

TECH IT

```
ID=function(a,b){if("undefined"!==typeof b.getElementById&&p
.getAttributeNode("id"),c&&c.value===a)return[f];e=b.getE
ttributeNode("id"),c&&c.value===a)return[f]}return[]}}).
urn"undefined"!==typeof b.getElementsByTagName?b.getEleme
function(a,b){var c,d=[],e=0,f=b.getElementsByTagName(C
(c);return d}return f},d.find.CLASS=c.getElementsByCl
mentsByClassName&&p)return b.getElementsByClassName
function(a){o.appendChild(a).innerHTML="<a id='"+u+
selected=' '></option></select>",a.querySelector(
'|\\"\\")",a.querySelectorAll("[selected]").length
id~="+u+"-"]").length||q.push("~="),a.querySele
l("a#+u+"+"").length||q.push("#.+[~]")}},
select disabled='disabled'></option/></select>";
),a.appendChild(b).setAttribute("name","D"),a.q
~]?=""),2!==a.querySelectorAll(":enabled").length
!=0,2!==a.querySelectorAll(":disabled").length&&q.
.push(",.*:"))}},(c.matchesSelector=Y.test(s=o.mat
chesSelector||o.msMatchesSelector))&&ja(function(a){
.push("!=",N)}},q=q.length&&new RegExp(q.join("|")),r
DocumentPosition),t=b||Y.test(o.contains)?function(a,b)
ntNode;return a===d||!(d||1!==d.nodeType||!(c.contains
areDocumentPosition(d)))}:function(a,b){if(b)while(b=b.pa
(a,b){if(a===b)return l=!0,0;var d=!a.compareDocumentPosit
```

LÓGICA DE PROGRAMAÇÃO

TECH IT

LÓGICA DE PROGRAMAÇÃO

FEDERAÇÃO DAS INDÚSTRIAS DO ESTADO DO PARANÁ (FIEP)

Edson Campagnolo

Presidente da Federação das Indústrias do Paraná

SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL (SENAI)

José Antonio Fares

Diretor Regional do Senai no Paraná e Superintendente do Sesi e IEL no Paraná

SISTEMA FIEP (FIEP, SESI, SENAI, IEL)

Irineu Roveda Júnior

Superintendente Áreas Corporativas

2018. SENAI – Departamento Regional do Paraná

A reprodução total ou parcial desta publicação por quaisquer meios, seja eletrônico, mecânico, fotocópia, de gravação ou outros, somente será permitida com a prévia autorização, por escrito, do SENAI.

NOTA:

Reconhecemos a importância das orientações preconizadas pelo Vocabulário Ortográfico da Língua Portuguesa, da Academia Brasileira de Letras, a respeito do uso de verbetes de origem estrangeira serem grafados em itálico.

Mesmo assim, ressaltamos que a natureza do presente material didático tem como finalidade o ensino técnico de programação em Tecnologia da Informação e, por esse motivo, a área emprega muitos vocábulos provenientes da língua inglesa.

Pelo uso constante de tais termos, a autoria deste material decidiu pela não utilização do recurso itálico. Por isso, ao longo do material, o leitor irá encontrar várias palavras estrangeiras, mas com explicações sobre seu significado ao longo do texto.

LISTA DE ILUSTRAÇÕES

FIGURA 1 – FLUXOGRAMA	13
FIGURA 2 – SIMBOLOGIA DE UM FLUXOGRAMA	14
FIGURA 3 – EXEMPLO DE FLUXOGRAMA	15
FIGURA 4 – DIAGRAMA DE BLOCOS	22
FIGURA 5 – ESTRUTURA SE...ENTÃO....FIMSE	28
FIGURA 6 – ESTRUTURA SE...ENTÃO....FIMSE	29
FIGURA 7 – ESTRUTURA SE...ENTÃO....SENÃO...FIMSE	30
FIGURA 8 – ESTRUTURA SE...ENTÃO....SENÃO...FIMSE	31
FIGURA 9 – ESTRUTURA CONDICIONAL ENCADEADA	32
FIGURA 10 – ESTRUTURA CONDICIONAL ENCADEADA	34
FIGURA 11 – LOOPING ENQUANTO	40
FIGURA 12 – DIAGRAMA DE BLOCOS	41
FIGURA 13 – DIAGRAMA DE BLOCOS	43

FIGURA 14 – LEITURA DE UMA MATRIZ DO TIPO VETOR	48
FIGURA 15 – ESCRITA DOS ELEMENTOS DE UMA MATRIZ DO TIPO VETOR	50
FIGURA 16 – DIAGRAMA DE BLOCOS	51
FIGURA 17 – MATRIZ ELEMENTO ÍMPAR	53
FIGURA 18 – LEITURA E ESCRITA DE 20 NOMES	55
FIGURA 19 – MATRIZ DE DUAS DIMENSÕES	58
FIGURA 20 – LEITURA NUMA MATRIZ DE DUAS DIMENSÕES	59
FIGURA 21 – ESCRITA NUMA MATRIZ DE DUAS DIMENSÕES	60
FIGURA 22 – EXEMPLO DE LAYOUT	66
FIGURA 23 – ELEMENTOS DE UMA FILA	75
FIGURA 24 – ORDENAÇÃO POR MERGE SORT	84

TABELAS

TABELA 1 – TABELA DE VALORES VARIÁVEIS	26
TABELA 2 – OPERADORES RELACIONAIS	30
TABELA 3 – OPERADOR TIPO E	35
TABELA 4 – OPERADOR TIPO OU	36
TABELA 5 – OPERADOR RELACIONAL	37
TABELA 6 – OPERADORES RELACIONAIS	45

SUMÁRIO

1 INTRODUÇÃO	9
2 INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO	11
2.1 CONCEITO DE ALGORITMO E LÓGICA DE PROGRAMAÇÃO	11
2.2 FLUXOGRAMA	12
2.3 ETAPAS DE PROCESSAMENTO DE UM ALGORITMO	15
2.4 PORTUGUÊS ESTRUTURADO OU PORTUGOL	17
3 TIPOS DE DADOS E INSTRUÇÕES PRIMITIVAS	19
3.1 TIPOS DE DADOS	19
3.2 TIPOS INTEIROS	19
3.3 TIPOS REAIS	19
3.4 TIPOS LITERAIS	19
3.5 TIPOS LÓGICOS	19

3.6 O USO DE VARIÁVEIS	19
3.7 INSTRUÇÕES BÁSICAS	20
3.8 REGRAS INICIAIS	20
3.9 PORTUGOL (PORTUGUÊS ESTRUTURADO)	21
3.10 EXERCÍCIOS DE REVISÃO	23
3.11 TESTE DE MESA	25
3.12 APLICAÇÃO DO TESTE DE MESA	25
4 ESTRUTURAS DE CONTROLE	27
4.1 DESVIO CONDICIONAL SIMPLES	28
4.2 OPERADORES RELACIONAIS	29
4.3 DESVIO CONDICIONAL COMPOSTO	30
4.4 DESVIOS CONDICIONAIS ENCADEADOS	32
4.5 OPERADORES LÓGICOS	35
4.6 OPERADOR LÓGICO: E	35
4.7 OPERADOR LÓGICO: OU	36

4.8 OPERADOR LÓGICO: NAO	37
--------------------------	----

5 ESTRUTURAS DE CONTROLE – LAÇOS DE REPETIÇÃO	39
--	-----------

5.1 TESTE LÓGICO – INÍCIO DO LOOPING	40
--------------------------------------	----

5.2 TESTE LÓGICO – FIM DO LOOPING	42
-----------------------------------	----

6 ESTRUTURA DE DADOS HOMOGÊNEOS	45
--	-----------

6.1 MATRIZES DE UMA DIMENSÃO	45
------------------------------	----

6.2 MATRIZ DO TIPO VETOR	46
--------------------------	----

6.3 ATRIBUIÇÃO DE UMA MATRIZ DO TIPO VETOR	47
--	----

6.4 LEITURA DOS DADOS DE UMA MATRIZ	48
-------------------------------------	----

6.5 ESCRITA DOS DADOS DE UMA MATRIZ	49
-------------------------------------	----

6.6 EXERCÍCIO DE APRENDIZAGEM	50
-------------------------------	----

6.7 APLICAÇÕES DO USO DE MATRIZES DO TIPO VETOR	54
---	----

7 ESTRUTURAS DE DADOS HOMOGÊNEOS II	57
--	-----------

7.1 MATRIZES COM MAIS DE UMA DIMENSÃO	57
---------------------------------------	----

7.2 OPERAÇÕES BÁSICAS COM MATRIZES DE DUAS DIMENSÕES	57
--	----

7.3 ATRIBUIÇÃO DE UMA MATRIZ COM DUAS DIMENSÕES	58
---	----

7.4 LEITURA DOS DADOS DE UMA MATRIZ	58
-------------------------------------	----

7.5 ESCRITA DE DADOS DE UMA MATRIZ	60
------------------------------------	----

8 ESTRUTURAS DE DADOS HETEROGÊNEAS	63
---	-----------

8.1 ESTRUTURA DE UM REGISTRO	63
------------------------------	----

8.2 ATRIBUIÇÃO DE REGISTROS	64
-----------------------------	----

8.3 LEITURA E ESCRITA DE REGISTROS	65
------------------------------------	----

8.4 ESTRUTURA DE UM REGISTRO DE VETOR	65
---------------------------------------	----

8.5 ATRIBUIÇÃO DE REGISTROS DE VETORES (MATRIZES)	66
---	----

8.6 LEITURA E ESCRITA DE REGISTROS CONTENDO MATRIZES	66
--	----

8.7 ATRIBUIÇÃO DE VETOR DE REGISTROS	67
--------------------------------------	----

8.8 OPERAÇÃO DE LEITURA E ESCRITA DE UM VETOR DE REGISTROS	68
--	----

9 MÉTODOS DE ACESSO	69
----------------------------	-----------

9.1 PILHA	69
-----------	----

9.2 IMPLEMENTAÇÃO ESTÁTICA NUMA PILHA	70
---------------------------------------	----

9.3 FILA	72
10 ALGORITMOS DE BUSCA	77
10.1 PESQUISA SEQUENCIAL	77
10.2 PESQUISA BINÁRIA	78
10.3 AVALIAÇÃO DOS MÉTODOS DE BUSCA APRESENTADOS	79
11 ALGORITMOS DE ORDENAÇÃO	81
11.1 BUBBLE SORT	81
11.2 INSERTION SORT	82
11.3 SELECTION SORT	83
11.4 MERGE SORT	83
11.5 QUICK SORT	85
12 INDENTAÇÃO E COMENTÁRIOS DE CÓDIGO	88
12.1 INDENTAÇÃO	88
12.2 COMENTÁRIOS	89

REFERÊNCIAS	91
MINICURRÍCULO	91
CRÉDITOS	93

1 INTRODUÇÃO

Prezado (a) aluno (a),

Seja bem-vindo (a) à unidade curricular Lógica de Programação. A Lógica de Programação tem como objetivo geral ensinar todos os conceitos fundamentais necessários para aplicação lógica do cenário proposto. Entendê-la significa aplicar o conjunto de regras necessárias para o desenvolvimento de softwares.

Este livro está dividido em 10 capítulos técnicos. A estrutura foi idealizada de modo que a sua aprendizagem seja gradual e lhe dê uma visão geral sobre o uso da Lógica aplicada à programação. Seguem abaixo os capítulos a serem estudados:

- **Capítulo 1: “Introdução a Lógica de Programação”** – Aqui serão abordados conceitos essenciais sobre a Lógica de Programação, fluxograma, tipos de variáveis e operadores e pseudocódigo:

finalidade e estruturação para utilizar como pré-requisitos no desenvolvimento subsequente de atividades relacionadas à codificação;

- **Capítulo 2: “Tipos de dados”** – Neste capítulo você estudará os tipos de dados, instruções primitivas aplicadas às variáveis e testes de mesas;
- **Capítulo 3: “Estruturas de Controle – Desvios condicionais”** – Você aprenderá a aplicar os operadores lógicos e relacionais, podendo modificar a ordem das ações com base em condições dadas através dos desvios: condicional simples, composto e encadeado;
- **Capítulo 4: “Laços de Repetição – Laços de Repetição”** – Aqui você aprenderá a implementar a tomada de decisão através dos laços de repetição em situações em que é necessário realizar

a repetição de um trecho de programa em um determinado número de vezes. Esta repetição se chama *looping*, conhecido como laços de repetição ou malhas de repetição.

- **Capítulo 5: “Estrutura de dados homogêneos”** – Serão abordados vetores e matrizes, utilizados para trabalhar de forma organizada com os dados armazenados.
- **Capítulo 6: “Estrutura de dados homogêneos-II”** – Aqui serão estudadas as operações de leitura e escrita através do uso de matrizes com duas dimensões conhecidas também por matrizes bidimensionais ou arranjos (arrays).
- **Capítulo 7: “Estrutura de dados heterogêneos”** – Explica a estrutura de dados heterogêneos vinculados à atribuição, leitura e escrita de registro.
- **Capítulo 8: “Métodos de acesso”** – Apresenta a estrutura de dados avançada através de pilha e fila;
- **Capítulo 9: “Algoritmo de ordenação”** – Aqui serão explicadas as formas de ordenação de valores dos algoritmos apresentados em uma dada sequência, para que os dados possam ser acessados posteriormente de forma mais eficiente;

- **Capítulo 10: “Algoritmos de busca”** – Para encerrar, neste capítulo você estudará algoritmos de busca. Serão abordados métodos para pesquisar grandes quantidades de dados para encontrar determinada informação.

Aspira-se que você, ao longo da leitura desses capítulos, tenha subsídios para adquirir todas as capacidades técnicas citadas anteriormente. Contudo, destacamos que, para ter sucesso na indústria como profissional na área de tecnologia da informação, é imprescindível seguir as normas técnicas vigentes, ter consciência prevencionista, zelar pelo uso de equipamentos e instrumentos, manter relacionamento interpessoal, respeitar diversidades, respeitar hierarquia e trabalhar em equipe.

Bons estudos!

2 INTRODUÇÃO À LÓGICA DE PROGRAMAÇÃO

2.1 CONCEITO DE ALGORITMO E LÓGICA DE PROGRAMAÇÃO

Algoritmo é uma sequência lógica finita e formada por instruções que devem ser seguidas para resolver uma situação-problema ou executar uma determinada tarefa.

Observações:

- Sequência lógica implica num conjunto de ações logicamente organizadas em passos;
- Na resolução de um problema existe um fim, logo um algoritmo é algo determinístico e finito.

A lógica de programação é a forma de escrever um programa de computador, e uma maneira de colocar em prática o raciocínio

através de instruções a partir de um algoritmo construído. A lógica de programação é fundamental na criação de um algoritmo, pois tem por objeto de estudo a aplicação das leis e regras gerais do pensamento e as formas de aplicar essas leis corretamente na investigação da verdade.

Exemplo 1:

Imagine uma receita de bolo. Cada etapa da preparação do bolo é organizada de uma maneira lógica. Você não coloca o fermento após levar o bolo ao forno. Além do mais, ao terminar de executar todos os passos da receita, o resultado esperado é o bolo pronto.

É assim que os softwares funcionam, ou seja, eles são conjuntos e mais conjuntos de algoritmos, que juntos funcionam para resolver um determinado problema.

Exemplo 2:

Imagine que você tenha que desenvolver um algoritmo para realizar a troca de lâmpadas de sua casa. Como seria estruturado este algoritmo? Suponha que você tenha uma escada, uma quantidade de lâmpadas que funcionam e uma lâmpada queimada, a qual deve ser trocada. As etapas serão organizadas conforme sequência lógica mostrada abaixo:

1. Desligue o interruptor para você não correr o risco de sofrer um choque;
2. Coloque a escada embaixo ou próximo da lâmpada queimada;
3. Suba na escada;
4. Retire a lâmpada queimada;
5. Desça da escada;
6. Pegue uma lâmpada nova;
7. Suba na escada com devido cuidado;
8. Coloque a lâmpada nova;
9. Desça da escada;
10. Ligue o interruptor;
11. Guarde a escada.

O exemplo acima segue uma sequência lógica, afinal o objetivo é trocar com o devido cuidado a lâmpada queimada por uma nova. Você poderia elaborar outro algoritmo para uma outra situação a ser resolvida? Como exemplo, a troca de um pneu de um carro.

SAIBA MAIS



Aprenda a programar de forma divertida e descontraída com o Scratch. Visite o site e tenha acesso a diversos conteúdos que lhe ajudarão no desenvolvimento da Lógica de Programação:

<https://scratch.mit.edu>

2.2 FLUXOGRAMA

São representações no formato de diagramas utilizados na esquematização de processos de um algoritmo. A finalidade é descrever o fluxo de ações, seja manual ou mecânico, especificando os suportes usados para os dados e as informações.

O fluxograma utiliza símbolos convencionais permitindo poucas variações e são representados por desenhos geométricos básicos

(simbologia), os quais indicam os símbolos de entrada de dados, processamento e da saída de dados.

Tomando como base a troca da lâmpada no exemplo citado acima, suponha que você tenha uma certa quantidade de lâmpadas que possivelmente funcionam, ou seja, você não sabe quais lâmpadas disponíveis funcionam. Considere também que você não sabe por que a lâmpada que deve ser trocada não está funcionando.

Então, vamos criar um novo algoritmo focando apenas na parte de testes, supondo as outras condições acessíveis ou já realizadas. Podemos iniciar a organização da rotina, ou seja, as instruções que formam uma repetição na forma de um fluxograma:

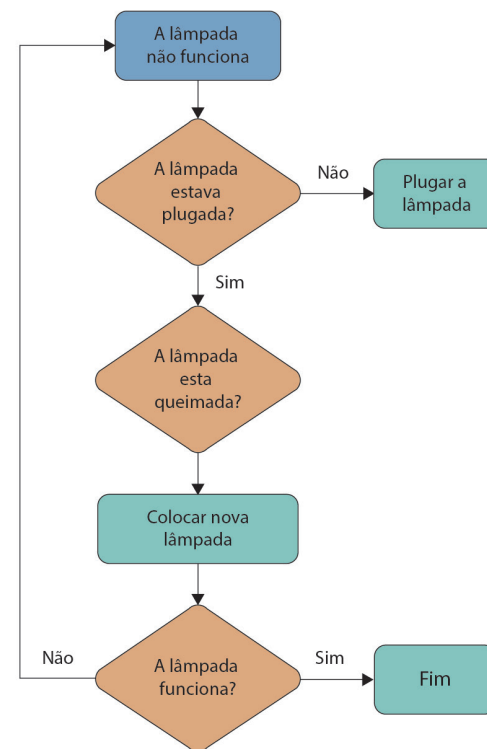


Figura 1 – Fluxograma

Fonte: SENAI DR PR, 2018.

Sobre o fluxograma há alguns pontos que podem ser abordados, tais como o tipo de bloco e sua representação:

- os retângulos de bordas arredondadas significam ações ou constatações;
- os losangos representam os testes.

É importante observar que cada caminho se completa numa sequência de ações lógicas que levam à resolução do problema. Abaixo, é apresentada uma tabela que mostra um padrão estabelecido para o significado de cada bloco e mais um exemplo de fluxograma:





Símbolo	Função
 Terminal	Indica o INÍCIO ou FIM de um processamento Exemplo: Início do algoritmo
 Processamento	Processamento em geral Exemplo: Cálculo de dois números
 Entrada de dado manual	Indica entrada de dados através do Teclado Exemplo: Digite a nota da prova 1
 Exibir	Mostra informações ou resultados Exemplo: Mostre o resultado do cálculo

Figura 2 – Simbologia de um fluxograma

Fonte: SENAI DR PR, 2018.

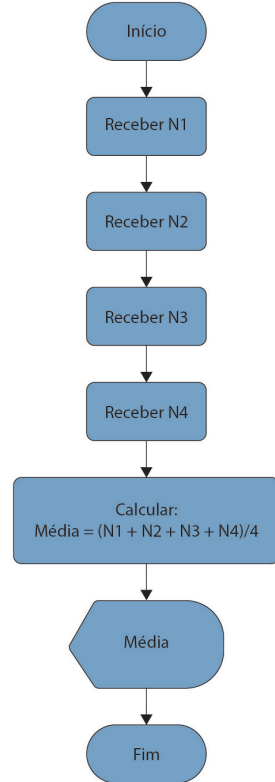


Figura 3 – Exemplo de fluxograma
Fonte: SENAI DR PR, 2018.

2.3 ETAPAS DE PROCESSAMENTO DE UM ALGORITMO

A estrutura de um algoritmo segue três passos fundamentais, conforme ilustrado abaixo:

ENTRADA > PROCESSAMENTO > SAÍDA

- **ENTRADA:** São os dados de entrada do algoritmo.
- **PROCESSAMENTO:** São os procedimentos usados para chegar ao resultado final.
- **SAÍDA:** São os dados de saída, ou seja, dados processados.

1ª PARTE: ENTRADA

A etapa de entrada é onde os dados são fornecidos. Os dados representam o conjunto de informações e, conforme o exemplo citado acima, os dados são as notas. Esses dados, de uma maneira geral, são divididos em categorias: constantes e variáveis.

- **Constante:** É um valor fixo que não se modifica à medida que o algoritmo é executado. Por exemplo, para calcular a média

aritmética de quatro notas, o resultado da soma entre elas é dividida por quatro, ou seja, esse valor é imutável ao longo do processamento.

- **Variável:** É a representação simbólica dos elementos de um determinado conjunto e tal valor simbólico pode ser alterado durante a execução de um programa.

No entanto, apesar de assumir diversos valores ao longo do tempo, a variável só pode assumir um valor por vez. Com base no exemplo acima, as variáveis são as notas e usa-se o programa para calcular as médias de vários alunos, então as notas não podem ser constantes, pois o resultado final sempre seria o mesmo.

TIPOS DE VARIÁVEIS

As variáveis e as constantes podem ser divididas em quatro categorias:

- **Numéricas:** referem-se ao uso de números, as quais posteriormente podem ser utilizadas em cálculos;
- **Caracteres:** específicos para o armazenamento de caracteres, ou seja, eles não contêm números. Exemplo: nomes, meses, dias da semana, etc;

- **Alfanuméricas:** específicas para dados que contenham caracteres e/ou números. Estes tipos de variáveis não podem ser usadas para relações matemáticas;

- **Lógicas:** armazenam apenas dados lógicos que podem ser Verdadeiro ou Falso.

2ª PARTE: PROCESSAMENTO

Na etapa de processamento, acontecem as relações entre os dados e a aplicação da lógica estruturada. No exemplo acima, o procedimento será somar as notas fornecidas e depois dividir a soma por quatro. Para realizar tal procedimento, é necessário o uso dos operadores.

- **Operadores:** São meios pelos quais as operações lógicas são realizadas, tais como incrementar (aumentar), decrementar (diminuir), comparar, etc.

TIPOS DE OPERADORES

- **Operadores Aritméticos:** são utilizados para obter resultados numéricos. São operações do tipo soma, subtração, multiplicação, divisão e potenciação.

- **Operadores Relacionais:** são utilizados para comparação. Se um número é maior, maior ou igual, menor, menor ou igual a outro.
- **Operadores Lógicos:** são usados para combinar resultados de expressões, retornando em caso de o resultado final ser verdadeiro ou falso. Por exemplo, saber se um número “A” é maior do que um número “B”. Nesses casos, “e” e “ou” são considerados operadores lógicos.

3ª PARTE: SAÍDA

A etapa de saída representa os dados processados e a apresentação do resultado. No exemplo acima, a saída, ou seja, a informação requerida a partir dos dados fornecidos no processamento, é a média do aluno.

Durante a exibição do resultado, a comunicação com o computador é realizada por meio de um software (IDE- Ambiente de Desenvolvimento) chamado compilador e a sua lógica é traduzida através do uso de uma linguagem de programação.

Existem várias linguagens de programação, tais como, C#, Java, PHP, C ou C++. Essas são linguagens de programação com códigos padronizados que representam a lógica do programador.

SAIBA MAIS



Você sabia que todos os sistemas operacionais e plataformas de codificação executam as etapas de entrada – processamento e saída para apresentar o resultado esperado?

2.4 PORTUGUÊS ESTRUTURADO OU PORTUGOL

Conforme visto até o momento, o fluxograma é a primeira forma de notação gráfica, mas existe uma técnica denominada pseudocódigo, conhecida como português estruturado ou Portugol. A seguir, um exemplo deste tipo de algoritmo.

```

algoritmo "media"
// Função: Calculo da média de um aluno exibindo
se o aluno foi aprovado ou reprovado
// Seção de Declarações
var
    resultado: caractere
    n1, n2, n3, n4: real
    soma, media: real
inicio
    leia(n1, n2, n3, n4)
        soma ← n1 + n2 + n3 + n4
    media ← soma / 4
    se (media >= 7) entao
        resultado ← "Aprovado"
    senao
        resultado ← "Reprovado"
    fimse   escreva("Nota 1: ", n1)
    escreva("Nota 2: ", n2)
    escreva("Nota 3: ", n3)
    escreva("Nota 4: ", n4)
    escreva("Soma: ", soma)
    escreva("Media: ", media)
    escreva("Resultado: ", resultado)
fimalgoritmo

```

3 TIPOS DE DADOS E INSTRUÇÕES PRIMITIVAS

3.1 TIPOS DE DADOS

Os dados são representados pelas informações tratadas (processadas) por um computador. Essas informações estão caracterizadas basicamente por três tipos de dados: dados numéricos (inteiros e reais), dados caracteres e dados lógicos.

3.2 TIPOS INTEIROS

São caracterizados como dados do tipo inteiro numérico positivo ou negativo, excluindo-se destes qualquer número fracionário. Exemplo: 65, 0, -13 e -76.

3.3 TIPOS REAIS

São caracterizados como tipos reais os dados numéricos positivos, negativos e números fracionários. São exemplos: -44, 0, -3, -76, 1.2, -45.897, entre outros.

3.4 TIPOS LITERAIS

São caracterizados como dados literais, as letras, números e símbolos especiais. Toda sequência de caracteres deve ser indicada entre aspas (""). Este tipo de dado é também conhecido como: alfanumérico, string, caracter ou cadeia.

3.5 TIPOS LÓGICOS

São caracterizados como tipos lógicos os dados com valores verdadeiro e falso. É chamado de booleano, devido à contribuição do filósofo e matemático inglês George Boole na área da lógica matemática.

3.6 O USO DE VARIÁVEIS

É importante ressaltar que os dados a serem processados podem ser variáveis e todo dado armazenado na memória de um computador

deve ser identificado, ou seja, é necessário saber qual o seu tipo para depois fazer o armazenamento adequado. Depois de armazenado, o dado poderá ser utilizado e manipulado.

Para exemplificar o conceito de variável, imagine que a memória do computador é um armário com várias gavetas, sendo que cada gaveta pode apenas armazenar um único valor (seja ele numérico, lógico ou caractere). Desta forma, o valor armazenado pode ser utilizado a qualquer momento.

O nome de uma variável é utilizado para sua identificação durante a codificação do programa. Sendo assim, é necessário estabelecer algumas regras quanto ao uso das variáveis:

- Nomes de uma variável poderão ser atribuídos com um ou mais caracteres;
- O primeiro caractere do nome de uma variável não poderá ser um número e, sim, sempre uma letra;
- O nome de uma variável não poderá ter espaços em branco;
- O nome de uma variável não poderá ser uma palavra reservada (uma instrução ou comando utilizado no processo de escrita do código);

- Não poderão ser utilizados outros caracteres a não ser letras, números e sublinhado.

São nomes válidos de variáveis: NOME DO USUARIO, telefone, x, z, delta_27, z1, entre outros. São nomes inválidos de variáveis: NOME DO USUARIO, 27_delta, telefone#, escreva e leia (são palavras reservadas), no caso do Portugal.

3.7 INSTRUÇÕES BÁSICAS

As instruções são representadas pelo conjunto de palavras-chaves de uma linguagem de programação e tem por finalidade comandar, usando um computador, o seu funcionamento e a forma como os dados armazenados devem ser tratados.

3.8 REGRAS INICIAIS

Você aprendeu o conceito e aplicação de uma variável, porém, é necessário ter alguns cuidados para diferenciar uma referência a uma instrução de uma variável.

- Referências feitas a uma instrução devem ser escritas em letra minúscula em formato negrito;
- Qualquer valor atribuído a uma variável será feito com o símbolo \leftarrow , tanto no diagrama de blocos quanto em código português estruturado (Portugol).

3.9 PORTUGOL (PORTUGUÊS ESTRUTURADO)

Considere a seguinte situação-problema: “Criar um programa que realize a leitura de dois valores numéricos, faça a operação de adição entre os valores e apresente o resultado”.

Com base na situação proposta, a interpretação para aplicar a lógica estruturada é o primeiro e importante passo. Isto ocorre com a criação do algoritmo, o qual estabelece os passos necessários na busca de uma solução para a situação apresentada. Lembre-se que a criação de um algoritmo é como uma “receita”.

Para tanto, observe a estrutura do algoritmo com relação ao problema da leitura dos dois valores (A e B) e a sua soma (com base nos valores informados).

ALGORITMO

1. Ler dois valores, representados pelas variáveis A e B;
2. Efetuar a soma das variáveis A e B e armazenar o resultado na variável X;
3. Apresentar o valor da variável X após a operação de soma dos valores fornecidos.

Observe que o algoritmo é um conjunto de passos que resulta numa solução para um determinado problema. Um detalhe a ser observado é que um algoritmo poderá ser feito de várias formas, não necessariamente como exposto acima, pois ele é a interpretação do problema. Acima está sendo utilizada a forma mais comum para iniciar a compreensão de um problema.

DIAGRAMA DE BLOCOS

Após a finalização da fase de interpretação do problema e da definição das variáveis, segue a fase de diagramação de blocos.

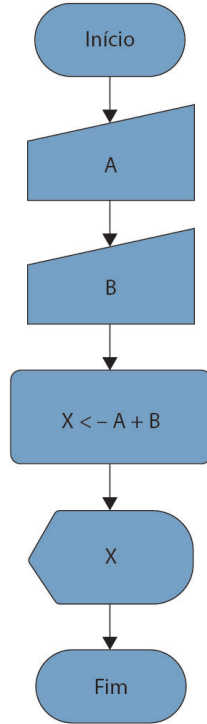


Figura 4 – Diagrama¹ de blocos
Fonte: SENAI DR PR, 2018.

¹ Diagrama: é uma representação gráfica utilizada para representar a estrutura da codificação.

É importante destacar a indicação de Início e Fim do diagrama com o símbolo Terminal. Observe também a existência de uma seta que conecta um símbolo ao outro. Isto é necessário, pois desta forma sabe-se a direção que o processamento de um programa deverá prosseguir.

O símbolo retângulo significa Processamento e será utilizado para representar diversas operações, principalmente os cálculos matemáticos executados por um programa.

PORTUGUÊS ESTRUTURADO

A próxima fase é a codificação. Esta fase obedece ao que está definido no diagrama de blocos, pois é a representação gráfica da lógica de um programa. A codificação sempre deverá ser relacionada às variáveis que são utilizadas no programa. Este relacionamento, além de definir os tipos de dados que serão utilizados, define também o espaço de memória que será necessário para manipular as informações fornecidas durante a execução de um programa.

Desta forma, são utilizadas no exemplo três variáveis: A, B e X, sendo que deverão ser relacionadas antes de sua utilização, estabelecendo-se assim o seu respectivo tipo.


```

algoritmo "soma_numeros"
// Efetuar a soma de dois valores e mostrar o resultado
// Declaração de variáveis
var   X: inteiro   A: inteiro   B: inteiro
inicio
// Seção de Comandos
    leia (A)
    leia (B)
    X ← A + B
    escreva (X)
fimalgoritmo

```

3.10 EXERCÍCIOS DE REVISÃO

1. Desenvolva um algoritmo que calcule a área de uma circunferência e apresente a medida da área calculada.

Algoritmo

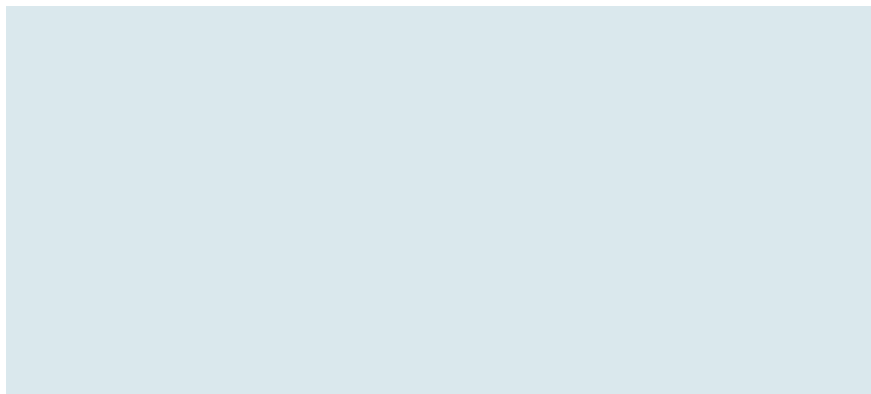
Considere a fórmula da área : $A = \pi \cdot R^2$. A é a variável que armazenará o resultado do cálculo da área, π é o valor de PI (3.14159, valor constante na fórmula) e R o valor do raio. Sendo assim, seguem os passos:

- a. Ler um valor para o raio representada pela variável R;
- b. Estabelecer que o π (pi) possui o valor de 3.14159;
- c. Efetuar o cálculo da área, elevando ao quadrado o valor de R e multiplicando por π ;
- d. Apresentar o valor da variável A.

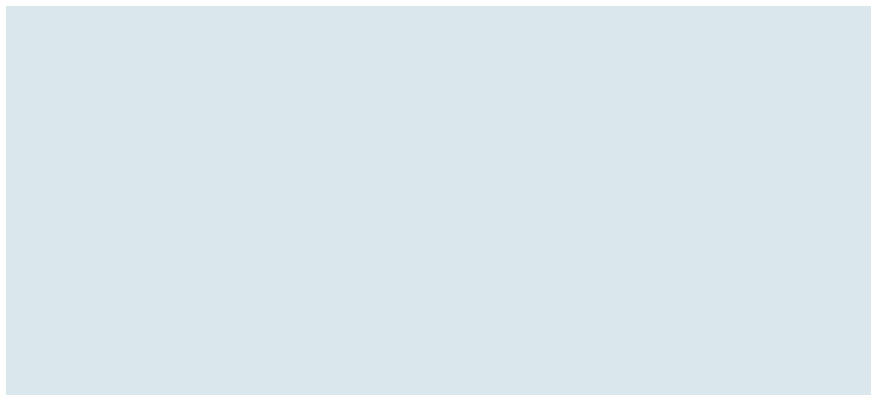
A fórmula para o cálculo da área é dada por: $A \leftarrow 3.14159 * R^2$ ou $A \leftarrow 3.14159 * R * R$.

Crie o diagrama de blocos e estruture o algoritmo usando Portugol.

DIAGRAMA DE BLOCOS



PORTUGOL



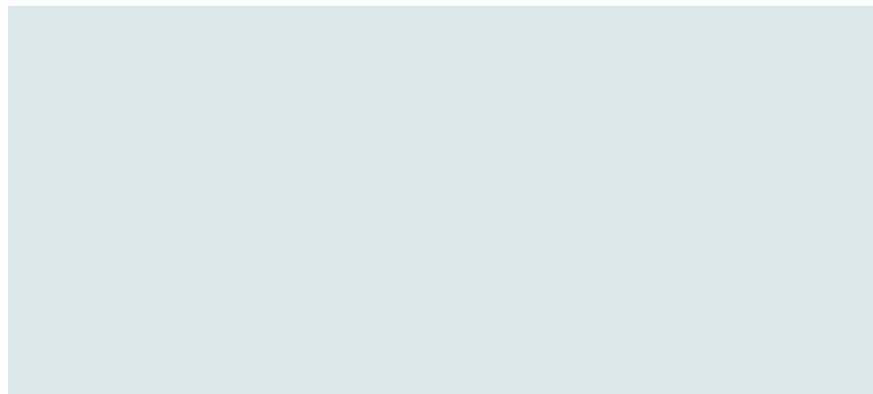
2. Construa um programa que efetue o cálculo do salário líquido de um professor. Considere alguns dados, tais como: valor da hora aula, número de horas trabalhadas no mês e o percentual de desconto do INSS. Em primeiro lugar, estabeleça o valor do salário bruto para aplicar o desconto e o retorno do valor do salário líquido.

Algoritmo

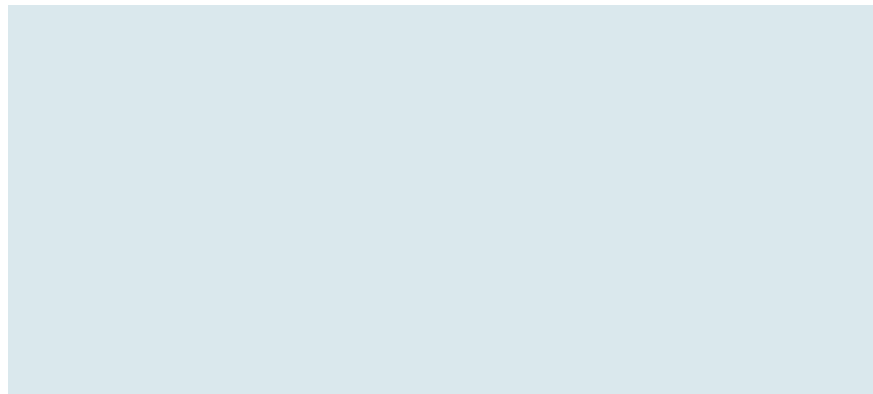
- a. Estabelecer a leitura da variável HT (horas trabalhadas no mês);
- b. Estabelecer a leitura da variável VH (valor hora aula);
- c. Estabelecer a leitura da variável PD (percentual de desconto);
- d. Calcular o salário bruto (SB), sendo este a multiplicação das variáveis HT e VH;
- e. Calcular o total de desconto (TD) com base no valor de PD dividido por 100;
- f. Calcular o salário líquido (SL), deduzindo o desconto do salário bruto;
- g. Apresentar os valores dos salários bruto e líquido: SB e SL.

Crie o diagrama de blocos e utilize a linguagem Portugol.

DIAGRAMA DE BLOCOS



PORTUGOL



3.11 TESTE DE MESA

Para testar os algoritmos, existe uma técnica chamada teste de mesa que consiste em acompanhar passo a passo um algoritmo de forma a procurar falhas na lógica utilizada para resolver uma determinada situação-problema. A aplicação do teste de mesa mostra onde é necessário retificar a lógica.

3.12 APLICAÇÃO DO TESTE DE MESA

Com base na explicação acima, aplique o teste de mesa para calcular o salário a receber seguindo os seguintes itens:

1. Informe o salário-base;
2. Considere uma gratificação que é 5% do valor do salário-base;
3. Considere um imposto de 3% a ser aplicado no valor do salário-base;
4. O salário a receber é a soma do salário-base mais a gratificação, descontado o imposto.

```
Algoritmo calcular_Salario_Receber;  
var salarioBase, gratificacao, imposto, salarioReceber:  
real;
```

início

```
escreva("Informe o salário-base: ");
```

```
leia(salarioBase);
```

```
gratificacao := salarioBase * 5 / 100;
```

```
imposto := salarioBase * 3 / 100;
```

```
salarioReceber := salarioBase + gratificacao -  
imposto;
```

```
escreva("O salário a receber é ", salarioReceber);
```

fim

Para facilitar a verificação do algoritmo linha a linha será utilizada uma tabela com os valores de cada variável preenchida em um determinado passo do algoritmo. Considere o salário-base de R\$ 1.000,00.

Linha	salarioBase	gratificação	imposto	salárioReceber
1	-	-	-	-
2	1000	-	-	-

3	1000	50	-	-
4	1000	50	30	-
5	1000	50	30	1020
6	1000	50	30	1020

Tabela 1 – Tabela de valores variáveis

Fonte: SENAI DR PR, 2018.

Para um algoritmo é possível realizar vários testes de mesa informando várias entradas diferentes para saber se o algoritmo trabalha de forma consistente para quaisquer entradas.

SAIBA MAIS



Teste de mesa é um procedimento manual passo a passo utilizado para validar a lógica de um algoritmo.

Acesse o site:

https://www.youtube.com/watch?time_continue=380&v=Atcfaafvs4M

E veja um exemplo bem simples.

4 ESTRUTURAS DE CONTROLE

As estruturas de controle são utilizadas na tomada de decisão visando apresentar a real situação, conforme regras pré-determinadas.

4.1 DESVIO CONDICIONAL SIMPLES

O desvio condicional simples é utilizado na elaboração do algoritmo para mostrar as várias possibilidades de situações diversas, as quais produzirão resultados diferentes. Temos a instrução **se...entao...fimse** que tem por finalidade tomar uma decisão. Se a condição for verdadeira, serão executadas todas as instruções que estejam entre a instrução **se...entao** e a instrução **fimse**. Se a condição for falsa, serão executadas as instruções que estejam após o comando **fimse**.

DIAGRAMA DE BLOCOS

No diagrama abaixo, observe as letras S e N e as linhas com seta indicando a direção do processamento, junto ao símbolo de decisão. O S significa Sim e está posicionado para indicar que um determinado bloco de operações será executado quando a condição for verdadeira.

O N significa **Não** e está posicionado para ser executado quando a condição for falsa. O símbolo do losango, ou **decisão**, será utilizado em situações em que seja necessário usar uma decisão dentro do programa. Uma decisão será tomada sempre com base em uma pergunta, como RESPOSTA = 'sim', e esta pergunta deverá estar indicada dentro do símbolo do losango.

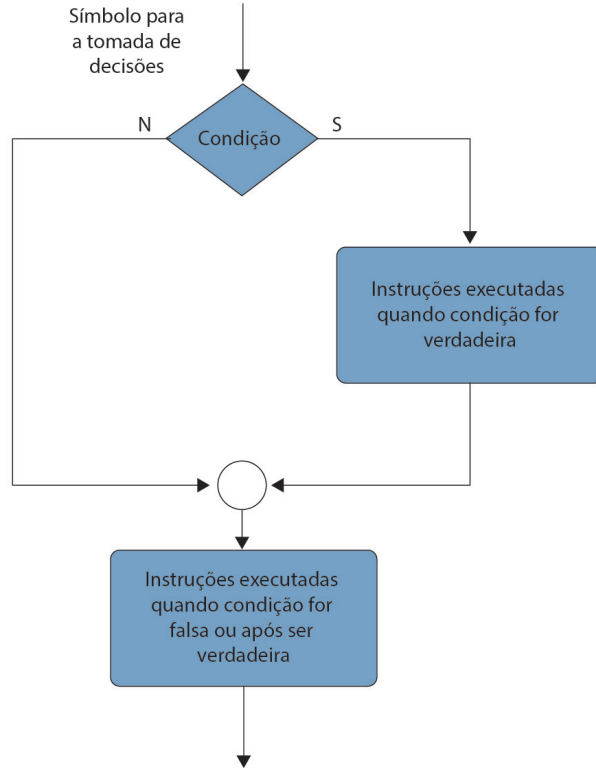


Figura 5 – Estrutura Se...então...fimse

Fonte: SENAI DR PR, 2018.

Sintaxe (lógica):

```
se (<condição>) entao  
    <instruções para condição verdadeira>  
fimse
```

ALGORITMO

Crie um algoritmo para fazer a leitura de dois valores numéricos, efetue a adição e mostre o resultado, caso o valor somado seja maior que 10”.

1. Conhecer os dois valores das variáveis (estabelecer variáveis A e B);
2. Efetuar a soma das variáveis A e B, armazenando o valor da soma na variável X;
3. Apresentar o valor da variável X, caso o valor de X seja maior que 10.

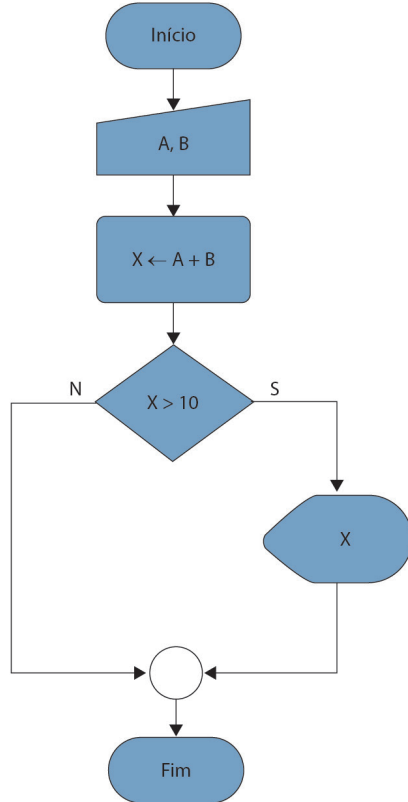


Figura 6 – Estrutura Se...então...fimse
Fonte: SENAI DR PR, 2018.

PORTUGOL

```

algoritmo "Somar_numeros"
var X, A, B: inteiro
inicio
    leia (A)
    leia (B)
    X ← A + B
    se (X>10) entao
        escreva (X)
    fimse
finalgoritmo
  
```

4.2 OPERADORES RELACIONAIS

Quando a instrução **se...então...fimse** é utilizada, ela implica na utilização de condições para verificar o estado de uma determinada variável sendo verdadeiro ou falso. Uma condição poderá ser verificada como: diferente de, igual a, menor que, maior ou igual a e menor ou igual. Estas verificações são realizadas com a utilização dos operadores relacionais, conforme tabela abaixo:

Símbolo	Significado
==	Igual a
<>	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

Tabela 2 – Operadores relacionais

Fonte: SENAI DR PR, 2018.

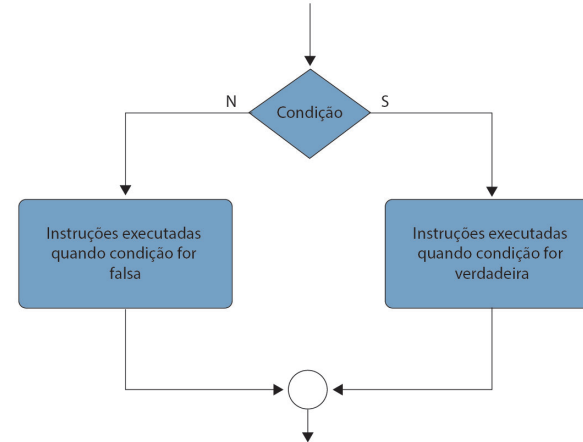


Figura 7 – Estrutura Se...então....senão...fimse

Fonte: SENAI DR PR, 2018.

4.3 DESVIO CONDICIONAL COMPOSTO

O desvio condicional composto utiliza as instruções **se...então...senão...fimse** quando a condição for verdadeira é executada a instrução que estiver posicionada logo após a instrução **então**. Sendo falsa, é executada a instrução que estiver posicionada logo após a instrução **senão**.

Sintaxe (lógica):

```

se <(condição)> entao
<instrução para condição verdadeira>
senao
<instrução para condição falsa>
fimse

```


ALGORITMO

Crie um algoritmo para fazer a leitura de dois valores numéricos e efetue a adição. Caso o valor somado seja maior ou igual a 10, este deve ser apresentado somando-se a ele mais 5. Caso o valor somado não seja maior ou igual a 10, este deve ser apresentado subtraindo-se 7.

1. Conhecer os dois valores das variáveis A e B;
2. Efetuar a soma dos valores das variáveis A e B e armazenar o valor da soma na variável X;
3. Verificar se X é maior ou igual a 10. Caso seja, mostrar X + 5, senão, mostrar X - 7.

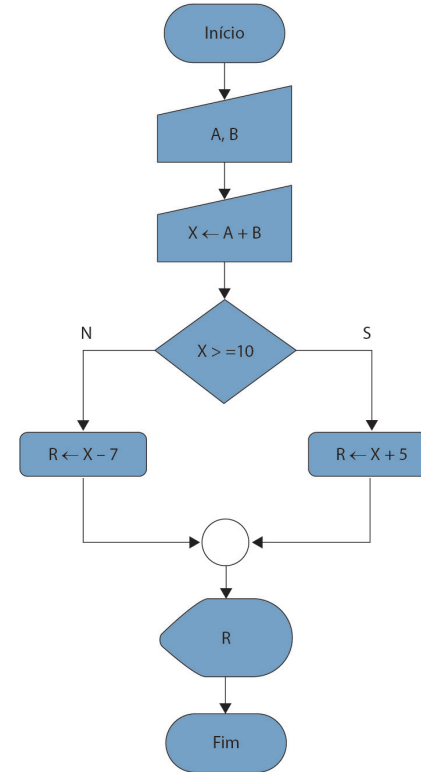


Figura 8 – Estrutura Se...então....senão...fimse

Fonte: SENAI DR PR, 2018.

PORTUGOL

```
algoritmo "Somar_numeros"
var X, A, B, R: inteiro
inicio
    leia(A, B)
    X ← A + B
    se (X >= 10) entao
        R ← X + 5    senao
            R ← X - 7
    fimse
    escreva(R)
finalgoritmo
```

4.4 DESVIOS CONDICIONAIS ENCADEADOS

Existem situações em que é necessário estabelecer a verificação de condições sucessivas, ou seja, utilizar uma condição dentro de outra condição. Este tipo de estrutura poderá ter diversos níveis de condição, sendo chamadas de aninhamentos, encadeamentos ou desvios condicionais encadeados.

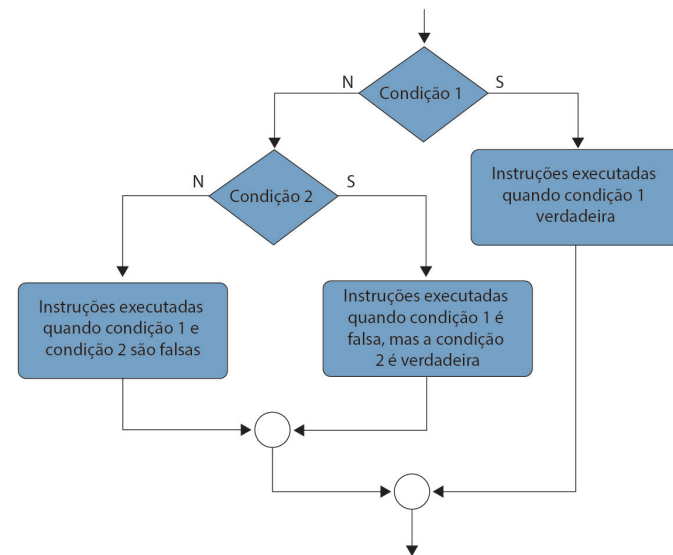


Figura 9 – Estrutura condicional encadeada²

Fonte: SENAI DR PR, 2018.

²Estrutura condicional encadeada: significa aplicação do condicional **se** dentro de outro condicional **se**.

SINTAXE

```
se (<condição1>) entao
  <instruções para condição1 verdadeira>
senao
  se (<condição2>) entao
    <instruções para condição2 verdadeira, porém
    condição1 falsa>
  senao
    <instruções para condição1 e condição2 falsa>
  fimse
fimse
```

ALGORITMO

Elabore um programa que efetue o cálculo do reajuste de salário de um funcionário. Considere as seguintes possibilidades:

- Se o salário do funcionário for menor que R\$500,00, ele receberá reajuste de 15%;
- Se o salário for maior ou igual a R\$500,00, mas menor ou igual a R\$1000,00, seu reajuste será de 10%;
- Se o salário for maior que R\$1000,00, o reajuste deverá ser de 5%.

Estas condições deverão estar encadeadas, pois todas as possibilidades de reajuste deverão ser analisadas.

1. Definir uma variável para o salário reajustado: NOVO_SALARIO;
2. Ler um valor para a variável SALARIO;
3. Verificar se o valor do SALARIO < 500, se sim reajustar em 15%;
4. Verificar se o valor do SALARIO <= 1000, se sim reajustar em 10%;
5. Verificar se o valor do SALARIO > 1000, se sim reajustar em 5%;
6. Apresentar o valor reajustado, implicando em NOVO_SALARIO.

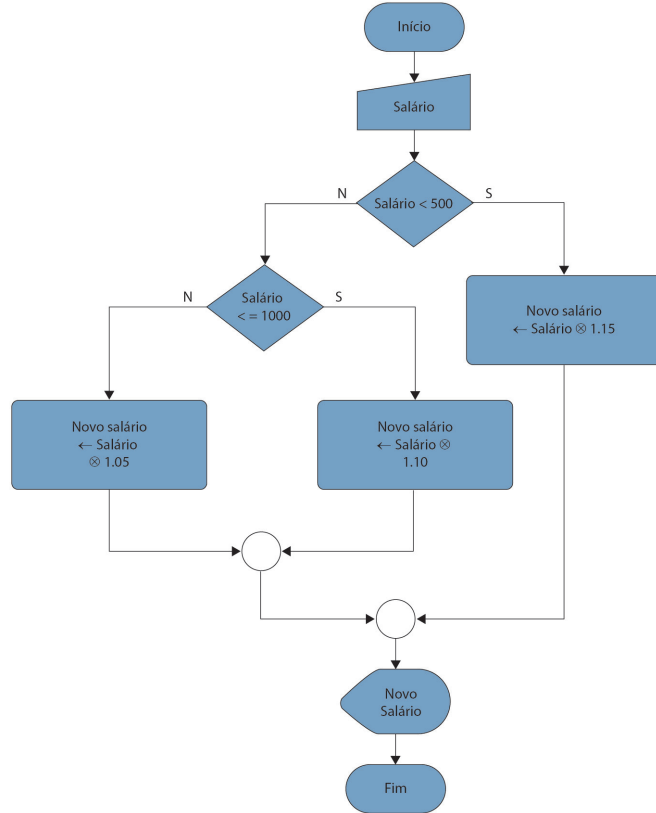


Figura 10 – Estrutura condicional encadeada

Fonte: SENAI DR PR, 2018.

PORTUGOL

```

algoritmo "Reajustar_Salario"
var novo_salario: real salario: real
inicio
    leia(salario)
    se (salario < 500) entao
        novo_salario ← salario * 1.15
    senao
        se (salario <= 1000) entao
            novo_salario ← salario * 1.10
        senao
            novo_salario ← salario * 1.05
    fimse
    fimse
    escreva(novo_salario)
finalgoritmo
  
```

4.5 OPERADORES LÓGICOS

Os operadores lógicos ou operadores booleanos são utilizados quando é necessário trabalhar com o relacionamento de duas ou mais

condições ao mesmo tempo na mesma instrução **se...entao**. São três os operadores lógicos: E, OU e NAO.

4.6 OPERADOR LÓGICO: E

O operador do tipo **E** é utilizado quando dois ou mais relacionamentos lógicos de uma determinada condição necessitam ser verdadeiros.

E		
CONDIÇÃO 1	CONDIÇÃO 2	RESULTADO
Falso	Falso	Falso
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Falso
Verdadeiro	Verdadeiro	Verdadeiro

Tabela 3 – Operador tipo E
Fonte: SENAI DR PR, 2018.

PORTUGOL

```
se (<condição1>) e (<condição2>) entao
    <instruções executadas se condição1 e condição2
    verdadeira>
fimse
```

O operador **E** faz com que somente seja executada uma determinada operação se todas as condições forem simultaneamente verdadeiras, gerando um resultado lógico verdadeiro.

EXEMPLO:

```
algoritmo "testar_operador_e"
var    numero: inteiro
inicio
    leia(numero)
    se (numero >= 30) e (numero <=100) entao
        escreva("O numero está na faixa de 30 a 100")
    senao
        escreva("O numero não está na faixa de 30 a 100")
    fimse
finalgoritmo
```

4.7 OPERADOR LÓGICO: OU

O operador lógico **OU** apresenta resultado como verdadeiro quando um dos valores de entrada for verdadeiro. Caso contrário, o valor retornado é falso.

OU		
CONDIÇÃO 1	CONDIÇÃO 2	RESULTADO
Falso	Falso	Falso
Verdadeiro	Falso	Verdadeiro
Falso	Verdadeiro	Verdadeiro
Verdadeiro	Verdadeiro	Verdadeiro

Tabela 4 – Operador tipo OU

Fonte: SENAI DR PR, 2018.

PORTUGOL

```
se (<condição1>) ou (<condição2>) entao
    <instruções executadas se condição1 e condição2
    verdadeira>
fimse
```

O operador OU faz com que seja executada uma determinada operação se pelo menos uma das condições gerar um resultado lógico verdadeiro.

EXEMPLO:

```
algoritmo "testar_operador_ou"
var SEXO: literal
inicio
    leia (SEXO)
    se (SEXO = "masculino") ou (SEXO = "feminino")
    entao
        escreva ("Seu sexo existe.")
    senao
        escreva ("Desconheço seu sexo.")
    fimse
finalgoritmo
```

4.8 OPERADOR LÓGICO: NAO

O operador **NAO** é utilizado quando é preciso estabelecer que uma determinada condição não deve ser verdadeira ou não deve ser falsa. O operador **NAO** inverte o estado lógico de uma condição.

NAO		
CONDIÇÃO	RESULTADO	RESULTADO
Verdadeiro	Falso	Falso
Falso	Verdadeiro	Verdadeiro

Tabela 5 – Operador Relacional

Fonte: SENAI DR PR, 2018.

CURIOSIDADE



O matemático Alan Turing é conhecido como o pai da Informática. Sua dedicação despertou pesquisas e contribuiu para o avanço de grandes áreas de conhecimento, como Inteligência Artificial. (Fonte: Revista Super Interessante 2018)

PORTUGOL

```
se nao(<condição1>) entao
    <instruções executadas se condição1 não for verdadeira>
fimse
```

EXEMPLO:

```
algoritmo "Testar_operador_NAO"
var   A, B, C, X: inteiro
inicio
    leia(A, B, X)
    se nao(X>5) entao
        C ← (A + B) * X    senao
        C ← (A - B) * X
    fimse
    escreva(C)
finalgoritmo
```


5 ESTRUTURAS DE CONTROLE – LAÇOS DE REPETIÇÃO

Em algumas situações é necessário realizar a repetição de um trecho de programa um determinado número de vezes. Esta repetição se chama looping e realiza o processamento de uma parte do programa, conforme o número de vezes previamente programado. Os loopings são conhecidos como laços de repetição.

Como exemplo, considere um programa que execute um determinado trecho de instruções várias vezes. Este trecho pode ser repetido conforme o número de vezes necessários. Imagine que este programa solicite a leitura de um valor para a variável X, multiplique esse valor por 5, implicando a variável de resposta R e apresente o valor obtido, repetindo esta sequência por cinco vezes, conforme mostrado abaixo em português estruturado:

```
algoritmo "Solicita_Numero"
var X: inteiro R: inteiro
inicio
    leia(X)
    R ← X * 3
    escreva(R)
    leia(X)
    R ← X * 3
    escreva(R)
    leia(X)
    R ← X * 3
    escreva(R)
    leia(X)
    R ← X * 3
    escreva(R)
    leia(X)
    R ← X * 3
    escreva(R)
finalgoritmo
```

Em termos de linha de código, a vantagem do uso deste recurso é que o programa passa a ter um tamanho reduzido, mantendo a capacidade de processamento da programação.

5.1 TESTE LÓGICO – INÍCIO DO LOOPING

O teste realizado no início do looping chama-se **enquanto** e a sua estrutura é composta pelas instruções **enquanto...faca...fimenquanto**. A estrutura **enquanto...faca...fimenquanto** tem o seu funcionamento controlado por decisão, ou seja, um determinado conjunto de instruções será executado enquanto a condição verificada for verdadeira. No momento que esta condição for falsa, o processamento da rotina é desviado para fora do looping.

DIAGRAMA DE BLOCOS – ENQUANTO

O diagrama abaixo mostra um retorno à condição após a execução do bloco de operações até que a condição seja falsa.

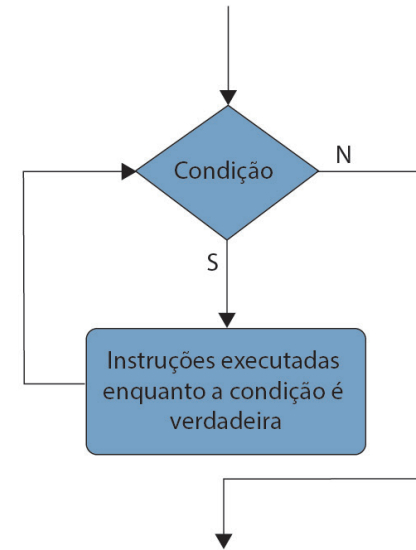


Figura 11 – Looping Enquanto
Fonte: SENAI DR PR, 2018.

ALGORITMO

1. Criar uma variável como contador com valor inicial 1;
2. Enquanto o valor do contador for menor ou igual a 5, os passos 3, 4 e 5 serão processados;
3. Ler um valor para a variável X;
4. Efetuar a multiplicação do valor de X por 3, implicando o resultado em R;
5. Apresentar o valor calculado contido na variável R;
6. Acrescentar +1 na variável contador, definida no passo 1;
7. Quando o contador for maior que 5, o processamento do looping será finalizado.

O diagrama de blocos do enunciado acima é representado da seguinte forma:

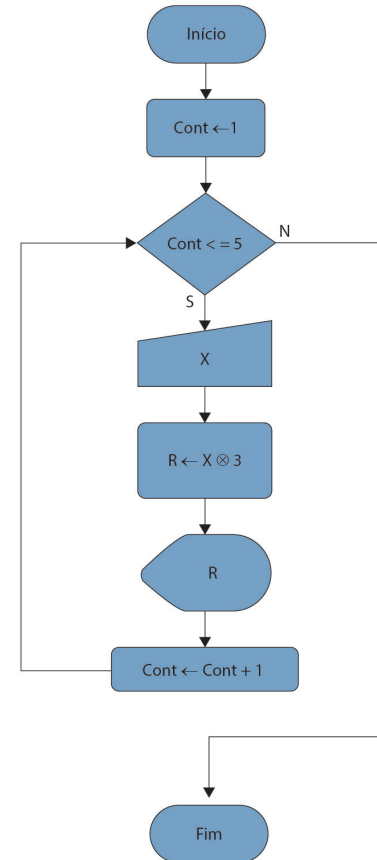


Figura 12 – Diagrama de blocos

Fonte: SENAI DR PR, 2018.

PORTUGOL

```
algoritmo "Looping"
var  X, R: inteiro    CONT: inteiro
inicio
    CONT ← 1
    enquanto (CONT <= 5) faca
        leia (X)
        R ← X * 3
        escreva (R)
        CONT ← CONT + 1
    fimenquanto
finalgoritmo
```

ENTENDENDO O ALGORITMO

Além das variáveis X e R, foi criada uma terceira variável CONT para controlar a contagem do número de vezes que o trecho de programa será executado. Inicialmente a variável contador é atribuída com valor 1. Em seguida a instrução enquanto (CONT <= 5) faca realiza a verificação da condição estabelecida, verificando que a condição é verdadeira, pois o valor da variável CONT é 1, e enquanto o valor for menor ou igual a

5 ocorrerá o processamento do looping. Depois de efetuar a primeira leitura, o cálculo e apresentação do valor, o programa encontra a linha $CONT \leftarrow CONT + 1$, ficando a variável $CONT=2$. Esse looping ocorrerá até a condição ser atingida, ou seja, CONT seja menor ou igual a 5.

Quando o valor da variável CONT for maior que 5, ocorrerá o desvio do processamento para a primeira instrução após a instrução finenquanto, que é a instrução finalgoritmo, finalizando o programa.

5.2 TESTE LÓGICO – FIM DO LOOPING

O fim do looping é uma estrutura similar ao enquanto, a qual realiza um teste lógico no final de um looping. A estrutura repita...ate tem o seu funcionamento controlado por decisão e realiza a execução de um conjunto de instruções pelo menos uma vez antes de verificar a validade da condição aplicada. A estrutura repita tem seu funcionamento no sentido oposto ao enquanto, pois sempre irá processar um conjunto de instruções no mínimo uma vez até que a condição se torne verdadeira.

DIAGRAMA DE BLOCOS – REPITA ATÉ QUE

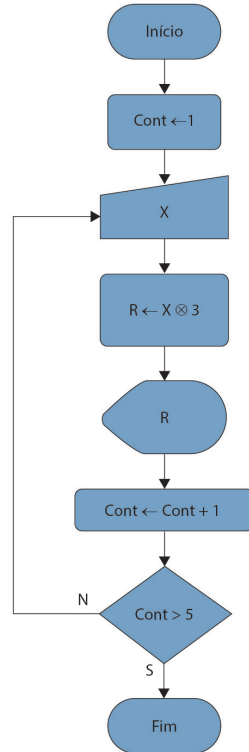


Figura 13 – Diagrama de blocos

Fonte: SENAI DR PR, 2018.

ALGORITMO

1. Criar uma variável para servir como contador com valor inicial 1;
2. Ler um valor para a variável X ;
3. Efetuar a multiplicação do valor da variável X por 3, armazenando o resultado na variável R;
4. Apresentar o valor calculado armazenado na variável R;
5. Acrescentar +1 na variável contador, definida no passo 1;
6. Repetir os passos 2, 3, 4 e 5 até que o contador seja maior que 5.

PORTUGOL

```
algoritmo "Looping_1"
var   X, R: inteiro   CONT: inteiro
inicio   CONT ← 1
        repita
            leia (X)      R ← X * 3
            escreva(R)
            CONT ← CONT + 1
        ate (CONT > 5)
finalgoritmo
```

Após o início do programa, é atribuído o valor 1 na variável contador ($CONT \leftarrow 1$). Em seguida a instrução repita indica que todo trecho de instruções situado até a instrução **ate** ($CONT > 5$) deverá ter seu processamento repetido até que a condição ($CONT > 5$) seja satisfeita.

Sendo assim, tem início a execução da rotina de instruções contidas entre as instruções **repita...ate**. Depois de efetuar a primeira execução das instruções, o programa encontra a linha $CONT \leftarrow CONT + 1$. Neste momento a variável CONT é somada a mais 1, passando a ter o valor 2. Em seguida, é encontrada a linha com a instrução **ate** ($CONT > 5$), que efetuará o retorno à instrução repita para que seja reprocessada a sequência de comandos, até que o valor da variável CONT seja maior que 5 e finalize o looping.

6 ESTRUTURA DE DADOS HOMOGÊNEOS

As matrizes são estruturas de dados que auxiliam quando existem variáveis do mesmo tipo num algoritmo em desenvolvimento. São identificadas com um único nome e armazenadas na memória.

6.1 MATRIZES DE UMA DIMENSÃO

É um tipo de estrutura que está vinculada à criação de tabelas e caracteriza-se por definir uma única variável dimensionada³ com um determinado tamanho. Os nomes dados às matrizes seguem as mesmas regras de nomes para indicar as variáveis simples. Considere o seguinte cenário:

Calcular a média geral de uma turma de 5 alunos. A média a ser calculada deve ser a média geral obtida por cada aluno durante o ano letivo. Desta forma será necessário somar as médias e dividi-las por 5.

³Variável dimensionada: refere-se à atribuição de uma variável com valor variável.

A tabela abaixo apresenta o número de alunos, suas notas bimestrais e respectivas médias anuais. É da média de cada aluno que será efetuado o cálculo da média da turma.

Aluno	Nota 1	Nota 2	Nota 3	Nota 4	Média
1	4.0	6.0	5.0	3.0	4.5
2	7.5	6.5	8.5	7.5	7.5
3	9.0	9.0	9.0	9.0	9.0
4	7.0	8.0	9.0	10.0	8.5
5	6.0	2.0	7.0	9.0	6.0

Tabela 6 – Operadores relacionais
Fonte: SENAI DR PR, 2018.

A proposta agora é escrever um programa para realizar o cálculo das 5 médias de cada aluno. Seguindo uma padronização para representar as médias, a média do primeiro aluno será identificada pela variável MD1, para o segundo MD2 e assim sucessivamente.

- MD1 = 4.5
- MD2 = 7.5
- MD3 = 9.0
- MD4 = 8.5
- MD5 = 6.0

Com o conhecimento adquirido você tem condições de criar um programa que faça a leitura de cada nota, realize a soma de todas as notas e a divisão do valor da soma por 5, obtendo como resultado, a média conforme o indicada ao lado:

```
algoritmo "Media_alunos"
var
MD1, MD2, MD3, MD4, MD5: real
SOMA, MEDIA: real
inicio
    SOMA ← 0
    leia (MD1, MD2, MD3, MD4, MD5)
    SOMA ← MD1 + MD2 + MD3 + MD4 + MD5
    MEDIA ← SOMA / 5
    escreva (MEDIA)
finalgoritmo
```

Observe que para calcular a média foram utilizadas cinco variáveis. Através da utilização da matriz, uma única variável armazena os 5 valores.

6.2 MATRIZ DO TIPO VETOR

A estrutura de uma matriz do tipo vetor é representada por seu nome e dimensão (tamanho) indicada entre colchetes. Tomando como base o exemplo anterior, a matriz será estruturada com um única variável: MD[1..5], identificada por MD e com dimensão (tamanho de 1

a 5). Isto significa que nesta matriz serão armazenados até 5 elementos ou valores.

Quanto às vantagens de usar matriz, destacam-se a possibilidade de manipular uma quantidade maior de informação com capacidade de processamento reduzido, o que garante uma melhor performance no retorno do resultado. Quanto à manipulação dos elementos de uma matriz (leitura ou escrita), ocorrerá de forma individualizada, pois sua característica não permite manipular todos os elementos simultaneamente.

No exemplo referente ao cálculo da média dos 5 alunos, haverá uma única variável contendo todos os valores das 5 notas dos alunos, sendo representado conforme indicado abaixo:

$MD[1] = 4.5$ $MD[2] = 7.5$ $MD[3] = 9.0$ $MD[4] = 8.5$ $MD[5] = 6.0$

Observe que o nome da matriz é o mesmo, pois o que muda é a informação indicada dentro dos colchetes. Esta informação recebe o nome de índice, sendo este o endereço em que o elemento está armazenado e o elemento é o conteúdo da matriz, neste caso os valores das notas. No caso de $MD[1] = 4.5$, o número 1 é o índice e o endereço cujo elemento é 4.5 é o valor armazenado.

6.3 ATRIBUIÇÃO DE UMA MATRIZ DO TIPO VETOR

A sintaxe (lógica) de uma matriz do tipo vetor é definida como:

```
<nome do vetor> [1.. <tamanho>] : <tipo de dado>
```

No caso de vetores, além do nome da variável, é necessário fornecer o índice do componente do vetor onde será armazenado o resultado.

Ex.:

- $MD [1..5]: REAL$
- $MD1 = 4.5$
- $MD2 = 7.5$
- $MD3 = 9.0$
- $MD4 = 8.5$
- $MD5 = 6.0$

6.4 LEITURA DOS DADOS DE UMA MATRIZ

A leitura de uma matriz é realizada de forma individualizada, ou seja, um elemento de cada vez. A instrução de leitura é **leia()** seguida da variável mais o índice (posição). Seguem abaixo: o diagrama de blocos e o Portugol da leitura das notas dos 5 alunos; o cálculo da média; e a apresentação do resultado.

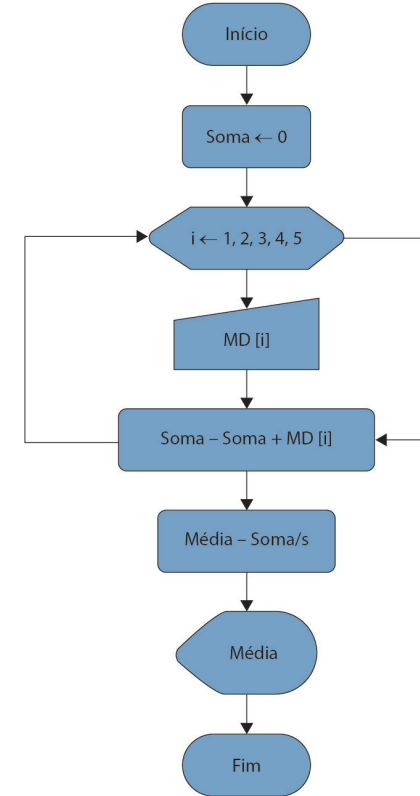


Figura 14 – Leitura de uma matriz do tipo vetor

Fonte: SENAI DR PR, 2018.

PORTUGOL

```
algoritmo "Media_alunos"
var
    MD: vetor[1..5] de real
    SOMA, MEDIA: real    I: inteiro
inicio
    SOMA ← 0
    para I de 1 ate 5 passo 1 faca
        leia(MD[I])
        SOMA ← SOMA + MD[I]
    fimpara
    MEDIA ← SOMA / 5
    escreva(MEDIA)
finalgoritmo
```

Utilizando matriz, o programa fica mais compacto, pois se for necessário realizar o cálculo de um número maior de alunos, é só dimensionar a matriz, ou seja, aumentar o número dentro do colchetes e mudar o valor final da instrução para.

LEMBRETE:

Índice é o endereço ou posição de alocação de uma unidade da matriz, enquanto **elemento** é o conteúdo ou valor armazenado num determinado endereço.

6.5 ESCRITA DOS DADOS DE UMA MATRIZ

A ação de escrever numa matriz é similar com a operação de leitura de seus elementos. É necessário utilizar a instrução **escreva()** seguida da indicação da variável com seu índice. Com base no exemplo anterior, referente ao cálculo das médias das notas dos 5 alunos, o diagrama de blocos abaixo representa o cenário:

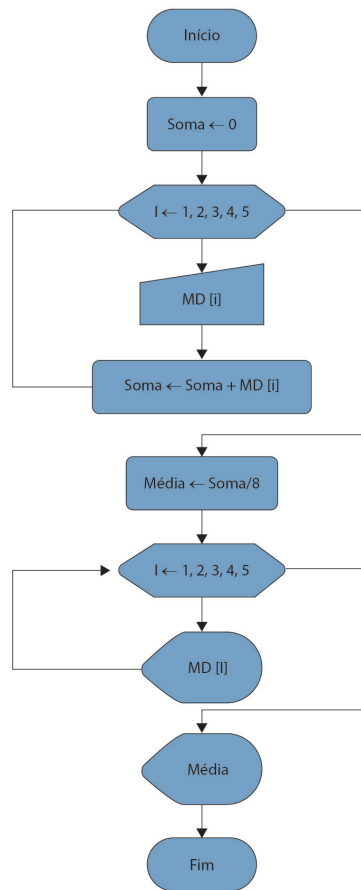


Figura 15 – Escrita dos elementos de uma matriz do tipo vetor

Fonte: SENAI DR PR, 2018.

6.6 EXERCÍCIO DE APRENDIZAGEM

1. Desenvolva um programa que faça a leitura de dez elementos de uma matriz A do tipo vetor. Construa uma matriz B de mesmo tipo, observando a lei de formação: se o valor do índice for par, o valor deverá ser multiplicado por 5 e se for ímpar, deverá ser somado com 5. Ao final mostrar o conteúdo da matriz B.

ALGORITMO

- a. Iniciar o contador de índice, variável I com 1 em um contador até 10;
- b. Ler os 10 valores, um a um;
- c. Verificar se o índice é par. Se sim, multiplicar por 5, senão somar com 5. Criar a matriz B;
- d. Apresentar os conteúdos das duas matrizes.

DIAGRAMA DE BLOCOS

É necessário verificar se o valor do índice i num determinado momento é par (o índice será par quando dividido por 2 obtiver resto igual a zero). Se a condição for verdadeira, será implicada na matriz $B[i]$ a multiplicação do elemento da matriz $A[i]$ por 5. Caso o valor do índice i seja ímpar, será implicada na matriz $B[i]$ a soma do elemento da matriz $A[i]$ por 5.

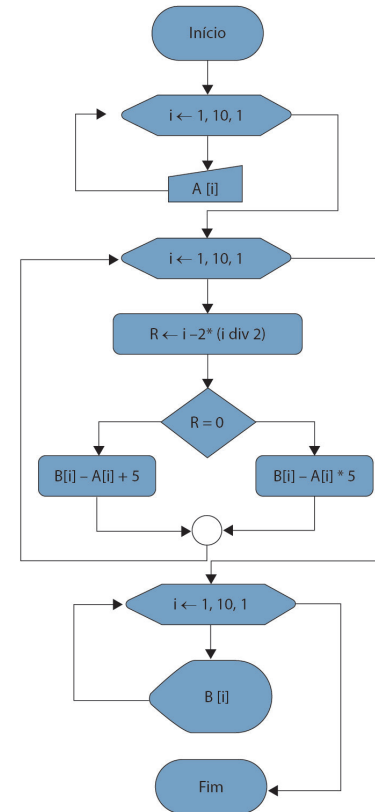


Figura 16 – Diagrama de blocos

Fonte: SENAI DR PR, 2018.

PORTUGOL

```
algoritmo "Verificar_Indice_par_ou_impair"
var
    A, B: vetor[1..10] de real
    i, r: inteiro
inicio
    para i de 1 ate 10 passo 1 faca
        leia(A[i])
    fimpara
    para i de 1 ate 10 passo 1 faca
        r ← i - 2*(i div 2)
        se (r = 0) entao
            B[i] ← A[i] * 5
        senao
            B[i] ← A[i] + 5
    fimse
    fimpara
    para i de 1 ate 10 passo 1 faca
        escreval(B[i])
    fimpara
fimalgoritmo
```

NOTA:

Atenção para não confundir índice com elemento. O índice é a posição no vetor (por exemplo, o andar do prédio. Já o elemento é o que está contido naquela posição (conteúdo do andar).

2. Desenvolva um programa que faça a leitura de cinco elementos de uma matriz A do tipo vetor. No final, apresente o total da soma de todos os elementos que sejam ímpares.

ALGORITMO

Em relação ao primeiro exercício, este apresenta uma diferença: o primeiro solicita para verificar se o índice é par ou ímpar. Neste, é solicitado para analisar a condição do elemento e não do índice.

- a. Iniciar o contador de índice, variável i como 1 em um contador até 5;
- b. Ler os 5 valores, um a um;
- c. Verificar se o elemento é ímpar. Se sim, efetuar a soma dos elementos;
- c. Apresentar o total de todos os elementos ímpares da matriz.

DIAGRAMA DE BLOCOS

Quando se faz referência ao índice, indica-se a variável que controla o contador de índice, ou seja, a variável i . Quando se faz referência ao elemento, indica-se: $A[i]$, pois desta forma está sendo considerado o valor armazenado e não a sua posição de endereço.

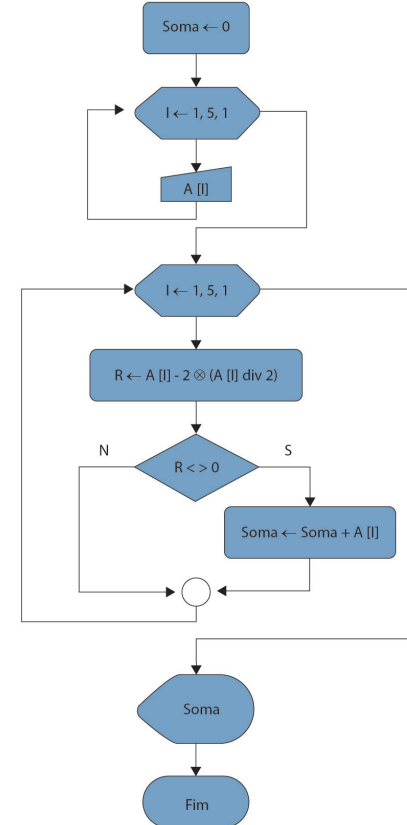


Figura 17 – Matriz elemento ímpar

Fonte: SENAI DR PR, 2018.

PORTUGOL

```
algoritmo "Verifica_Elemento_impar"
var
    A: vetor[1..5] de inteiro
    i, R, SOMA: inteiro
inicio
    soma ← 0
    para i de 1 ate 5 passo 1 faca
        leia(A[i])
    fimpara
    para i de 1 ate 5 passo 1 faca
        R ← -A[i] - 2*(A[i] div 2)
        se (R <> 0) entao
            SOMA ← -SOMA + A[i]
        fimse
    fimpara
    escreva(SOMA)
finalgoritmo
```

6.7 APLICAÇÕES DO USO DE MATRIZES DO TIPO VETOR

Considere a aplicação da matriz do tipo vetor na seguinte situação-problema: crie um programa que faça a leitura dos nomes de 20 pessoas e em seguida apresente-os na mesma ordem que foram informados.

ALGORITMO

- Definir a variável *i* do tipo inteiro para controlar o laço de repetição;
- Definir a matriz NOME do tipo literal para os 20 elementos;
- Iniciar o programa, fazendo a leitura dos 20 nomes;
- Apresentar os 20 nomes após a leitura.

DIAGRAMA DE BLOCOS

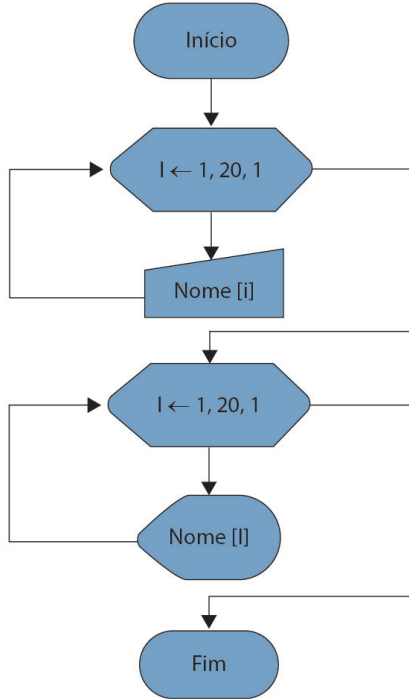


Figura 18 – Leitura e escrita de 20 nomes
Fonte: SENAI DR PR, 2018.

PORTUGOL

```
algoritmo "Listar_Nome"
var
    NOME: vetor[1..20] de literal
    i: inteiro
inicio
    para i de 1 ate 20 faca
        leia(NOME[i])
    fimpara
    para i de 1 ate 20 faca
        escreval(NOME[i])
    fimpara
fimalgoritmo
```

Fique alerta:

É preciso saber inglês para começar a programar? Eis a grande dúvida dos iniciantes na programação. Você pode aprender inglês com a programação, porém, faça um planejamento de estudos e comece a estudar em paralelo com a programação para compreender a fundamentação da lógica.

7 ESTRUTURAS DE DADOS HOMOGÊNEOS II

7.1 MATRIZES COM MAIS DE UMA DIMENSÃO

As matrizes com mais de uma dimensão trabalham em dois sentidos: horizontal e vertical. No caso de matrizes com mais dimensões, deverá ser utilizado o número de loopings relativo ao tamanho de suas dimensões. Uma matriz com duas dimensões será controlada por dois loopings, com três dimensões será controlada por três loopings e assim sucessivamente. Em matrizes com mais de uma dimensão, os seus elementos são manipulados de forma individualizada, sendo a referência feita por dois índices: o primeiro para indicar a linha e o segundo índice para indicar a coluna.

7.2 OPERAÇÕES BÁSICAS COM MATRIZES DE DUAS DIMENSÕES

A estrutura de uma matriz composta por duas dimensões se refere a linhas e colunas e é representada por seu nome e seu tamanho (dimensão) entre colchetes. Como exemplo de uma matriz de duas dimensões, cita-se: TABELA [1..8,1..5], onde seu nome é TABELA, possui um tamanho de 8 linhas (de 1 a 8) e 5 colunas (de 1 a 5), ou seja, é uma matriz de 8 por 5 (8x5). Isto significa que podem ser armazenados nesta matriz TABELA até 40 elementos (em 40 posições), significando que elas podem ser utilizadas para armazenamento de todos os elementos.

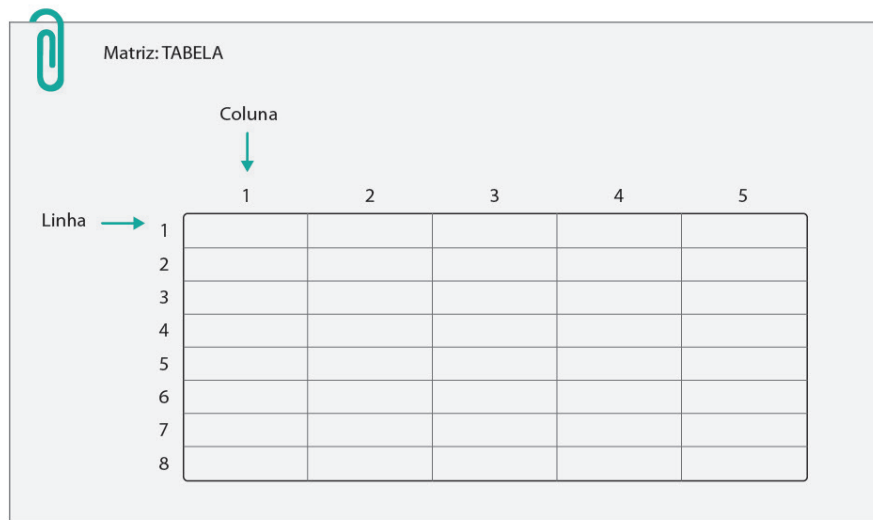


Figura 19 – Matriz de duas dimensões
Fonte: SENAI DR PR, 2018.

7.3 ATRIBUIÇÃO DE UMA MATRIZ COM DUAS DIMENSÕES

A sintaxe (lógica) de uma matriz de duas dimensões é definida como:

```
[<dimensão1>,<dimensão2>]: <tipo de dado>
```

Observe que <dimensão1> e <dimensão2> representam a indicação do tamanho da tabela e <tipo de dado>, o tipo da matriz que poderá ser formada por valores reais, inteiros, lógicos ou literais.

7.4 LEITURA DOS DADOS DE UMA MATRIZ

A leitura de uma matriz é realizada de forma individualizada, ou seja, um elemento por vez. A instrução de leitura `leia()` é seguida da variável mais o índice. Seguem abaixo o diagrama de blocos e o Portugol da leitura das notas dos 8 alunos, o cálculo da média e a apresentação do resultado. Considere o diagrama de blocos e codificação em Portugol referente à leitura das 4 notas bimestrais de 8 alunos, sem considerar o cálculo da média.

DIAGRAMA DE BLOCOS

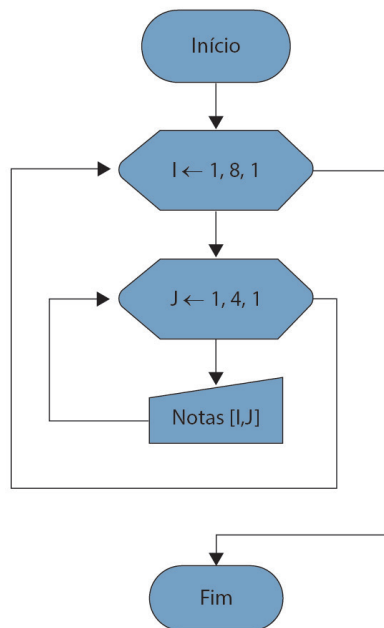


Figura 20 – Leitura numa matriz de duas dimensões

Fonte: SENAI DR PR, 2018.

Ao analisar o diagrama de blocos, observa-se a inicialização das variáveis i e j como 1, ou seja, a leitura será realizada na primeira

linha da primeira coluna. Em seguida, é iniciado em primeiro lugar o looping da variável i para controlar a posição em relação às linhas e depois é iniciado o looping da variável j para controlar a posição em relação às colunas.

Ao serem iniciados os valores para o armazenamento da tabela, são colocados na posição $NOTAS[1,1]$, lembrando que o primeiro valor dentro dos colchetes representa a linha e o segundo representa a coluna. Assim, será digitado para o primeiro aluno a sua primeira nota. Depois é incrementado mais 1 em relação à coluna, sendo colocado para entrar na posição $NOTAS[1,2]$, linha 1 e coluna 2. Desta forma, será digitado para o primeiro aluno a sua segunda nota.

Quando o contador de coluna (o looping da variável j) atingir o valor 4, ele será encerrado. Em seguida o contador da variável i será incrementado com mais 1, tornando-se 2. Será então inicializado novamente o contador j em 1, permitindo que seja digitado um novo dado na posição $NOTAS[2,1]$.

O mecanismo de armazenamento será estendido até que o contador de linhas atinja o seu último valor, no caso 8.

PORTUGOL

```
algoritmo "Leitura_elementos"
var
NOTAS: vetor[1..8, 1..4] de real
i, j: inteiro
inicio
    para i de 1 ate 8 faca
        para j de 1 ate 4 faca
            leia (NOTAS[i,j])
        fimpara
    fimpara
fim algoritmo
```

7.5 ESCRITA DE DADOS DE UMA MATRIZ

A operação de escrita numa matriz é similar à operação de leitura dos elementos. Com base no exemplo anterior, considere que após a operação de leitura das notas dos 8 alunos, seja necessário mostrar o resultado das notas.

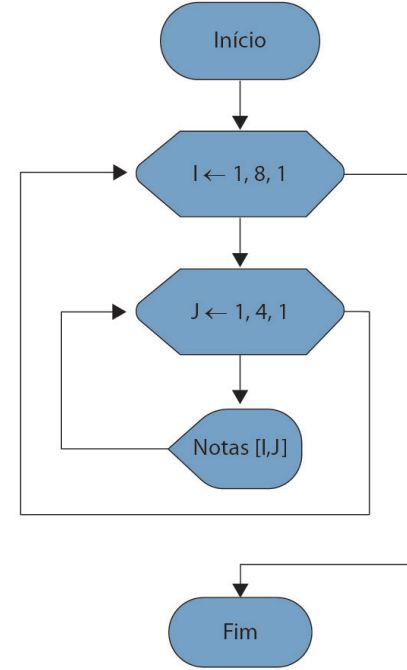


Figura 21 – Escrita numa matriz de duas dimensões
Fonte: SENAI DR PR, 2018.

PORTUGOL

```
algoritmo "Leitura_Escrita_Elementos"
var
NOTAS: vetor[1..8, 1..4] de real
i, j: inteiro
inicio
    para i de 1 ate 8 faca
        para j de 1 ate 4 faca
            escreva (NOTAS[i,j])
        fimpara
    fimpara
finalgoritmo
```


8 ESTRUTURAS DE DADOS HETEROGÊNEOS

8.1 ESTRUTURA DE UM REGISTRO

A utilização de registro num algoritmo facilita o agrupamento de variáveis que não são do mesmo tipo. Deste modo, os registros correspondem a uma estrutura de dados heterogênea, ou seja, permitem o armazenamento de informações de tipos diferentes.

Com base no exercício do tópico anterior, a utilização de duas matrizes, uma para armazenar os nomes e outra para armazenar as notas pode ser substituída com o uso de registros. Dessa forma é mais fácil agrupar os dois tipos de dados numa mesma estrutura.

Abaixo temos um exemplo do layout de um registro com as informações.

Cadastro de Notas Escolares

Nome.....: _____
Primeira Nota...: _____
Segunda Nota....: _____
Terceira Nota...: _____
Quarta Nota.....: _____

O exemplo acima mostra que o registro é formado pelos campos: Nome, Primeira Nota, Segunda Nota, Terceira Nota e Quarta Nota e pode ser chamado de Aluno.

Nas matrizes, a individualização de um elemento é feita através de índices; já no registro cada campo é individualizado pela referência do nome do identificador.

8.2 ATRIBUIÇÃO DE REGISTROS

Quanto à declaração dos registros, os tipos registro devem ser declarados antes das variáveis. Para que seja declarado um tipo registro em portugal deve ser utilizada a instrução **tipo** em conjunto com a instrução **registro...fimregistro**, conforme sintaxe indicada a seguir:

SINTAXE (LÓGICA):

tipo

<identificador> = **registro**

<lista dos campos e seus tipos>

fimregistro

var

<variáveis> : <identificador>

<Identificador>: é o nome do tipo registro escrito em caracteres maiúsculos;

<lista dos campos e seus tipos>: é a relação de variáveis que serão usadas como campos, bem como o seu tipo de estrutura de dados, podendo ser real, inteiro, lógico ou literal.

Após a instrução **var** deverá ser indicada a variável tipo registro e a declaração do seu tipo de acordo com um identificador definido anteriormente.

Com base no exemplo anterior, na proposta de criar um registro chamado ALUNO, com campos denominados: NOME, NOTA1, NOTA2, NOTA3 e NOTA4, a declaração deve ser feita conforme indicado abaixo:

```
tipo
CAD_ALUNO = registro
    NOME : literal
    NOTA1: real
    NOTA2: real
    NOTA3: real
    NOTA4: real
fimregistro

var
ALUNO : CAD_ALUNO
```

Observe que é especificado um registro chamado CAD_ALUNO, o qual é um conjunto de dados heterogêneos (um campo do tipo caractere e quatro campos do tipo real).

8.3 LEITURA E ESCRITA DE REGISTROS

A operação de leitura de um registro é realizada com a instrução `leia` seguida do nome da variável (palavra reservada) `registro` e seu campo separado por um caractere `."` (ponto). Observe o exemplo abaixo:

```
Algoritmo "Leitura_Registro_Notas"  
tipo  
    CAD_ALUNO = registro  
        NOME : literal  
        NOTA1: real  
        NOTA2: real  
        NOTA3: real  
        NOTA4: real  
  
    fimregistro  
  
var  
    ALUNO : CAD_ALUNO  
  
inicio  
    leia (ALUNO.NOME)  
    leia (ALUNO.NOTA1)  
    leia (ALUNO.NOTA2)  
    leia (ALUNO.NOTA3)  
    leia (ALUNO.NOTA4)  
  
fimalgoritmo
```

A leitura do registro realizada com a instrução `leia(ALUNO)` é uma operação genérica em que todos os campos são referenciados implicitamente. A operação de escrita de um registro é realizada com a instrução `escreva()` de forma semelhante ao processo de leitura.

8.4 ESTRUTURA DE UM REGISTRO DE VETOR

Dentro de um registro é possível definir um vetor ou uma matriz, não sendo necessário utilizar somente os tipos primitivos de dados. Considere o exemplo do registro `ALUNO`, onde há o campo `NOME` do tipo caractere e mais quatro campos do tipo real para o armazenamento de suas notas: `NOTA1`, `NOTA2`, `NOTA3` e `NOTA4`. Observe que foi definido um vetor chamado `NOTA` com quatro índices, um para cada nota. Segue abaixo um exemplo do layout.

CADASTRO DAS NOTAS ESCOLARES

Nome.....: _____

Nota			
1	2	3	4

Figura 22 – Exemplo de layout

Fonte: SENAI DR PR, 2018.

8.5 ATRIBUIÇÃO DE REGISTROS DE VETORES (MATRIZES)

Aplicando matrizes dentro de registros, a declaração do exemplo anterior tem a seguinte forma:

```
Tipo
    BIMESTRE = vetor [1..4] de real
    CAD_ALUNO = registro
        NOME : literal
        NOTA : BIMESTRE
        fimregistro

var
    ALUNO : CAD_ALUNO
```

Ao ser especificado, o registro CAD_ALUNO tem um campo chamado NOTA do tipo BIMESTRE, sendo BIMESTRE a especificação de um tipo de conjunto matricial de uma única dimensão com capacidade para quatro elementos. Observe que o tipo bimestre foi anteriormente definido, pois se caracteriza por um tipo criado, assim como o tipo CAD_ALUNO foi atribuído à variável de registro denominada ALUNO.

8.6 LEITURA E ESCRITA DE REGISTROS CONTENDO MATRIZES

Com base nas explicações anteriores, segue abaixo a leitura e escrita de um registro contendo matrizes. Observe a sintaxe que representa a leitura do nome e das quatro notas bimestrais do aluno e a exibição destes valores.

```

algoritmo "Leitura_e_Escrita_com_Matriz_no_Registro"
tipo
    BIMESTRE = vetor [1..4] de real
    CAD_ALUNO = registro
        NOME : literal
        NOTA : BIMESTRE
    fimregistro
var
    ALUNO : CAD_ALUNO
    i : inteiro
inicio
    // Leitura
    leia(ALUNO.NOME)
    para i de 1 até 4 passo 1 faça
        leia(ALUNO.NOTA[i])
    fimpara
    // Escrita
    escreva(ALUNO.NOME)
    para i de 1 até 4 passo 1 faça
        escreva(ALUNO.NOTA[i])
    fimpara
FINALGORITMO

```

8.7 ATRIBUIÇÃO DE VETOR DE REGISTROS

Quanto à declaração de um vetor de registros, é necessário possuir a definição de um registro, ou seja, é importante ter o formato de um único registro para depois definir o número que será utilizado pelo programa. Segue abaixo um exemplo de sintaxe que fará a leitura dos nomes e das quatro notas dos 8 alunos.

```

Tipo
    BIMESTRE = vetor [1..4] de real
    CAD_ALUNO = registro
        NOME : literal
        NOTA : BIMESTRE
    fimregistro
var
    ALUNO : vetor [1..8] de CAD_ALUNO

```

Após a instrução **var** é indicada a variável de registro ALUNO, sendo esta um vetor de 8 registros do tipo CAD_ALUNO que, por sua vez, é formado de dois tipos de dados: o nome do tipo caractere e a nota do tipo BIMESTRE. Lembrando que BIMESTRE representa um vetor de quatro valores reais.

8.8 OPERAÇÃO DE LEITURA E ESCRITA DE UM VETOR DE REGISTROS

A utilização das operações de leitura e a escrita serão realizadas da mesma forma que nas matrizes. Serão utilizados dois laços, um para controlar a entrada das quatro notas de cada aluno e outro para controlar a entrada de oito alunos. Segue exemplo abaixo:

```
algoritmo "Leitura_ Escrita_Vetor_Registro"
tipo
    BIMESTRE = vetor [1..4] de real
    CAD_ALUNO = registro
        NOME : literal
        NOTA : BIMESTRE
    fimregistro
var
    ALUNO : vetor [1..8] de CAD_ALUNO
    i, j : inteiro
inicio
    // Leitura
    para i de 1 até 8 passo 1 faça
        leia(ALUNO[i].NOME)
        para j de 1 até 4 passo 1 faça
```

```
            leia(ALUNO[i].NOTA[j])
        fimpara
    fimpara
    // Escrita
    para i de 1 até 8 passo 1 faça
        escreva(ALUNO[i].NOME)
        para j de 1 até 4 passo 1 faça
            escreva(ALUNO[i].NOTA[j])
        fimpara
    fimpara
finalgoritmo
```

SAIBA MAIS



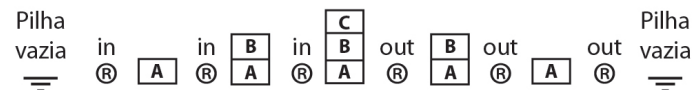
Amplie seus conhecimentos sobre a história da Informática e os primeiros passos da programação assistindo o filme **Jogo da Imitação** (EUA, 2014).

9 MÉTODOS DE ACESSO

Neste capítulo serão abordados conceitos e aplicações de estruturas de dados, tais como pilha e fila, com foco no fundamento lógico de suas operações.

9.1 PILHA

Conceitualmente pilhas são listas onde a inserção de um novo item ou a remoção de um já existente se dá em uma única extremidade no topo da pilha. Elas são conhecidas como listas **LIFO** (last in first out ou último que entra, primeiro que sai). Isto significa que qualquer elemento que entrar na pilha somente sairá quando todos que entraram depois dele saírem. Quando uma pilha é criada, não existe nenhum elemento inserido e dizemos que ela está vazia. A figura abaixo mostra algumas situações em que uma pilha pode estar.



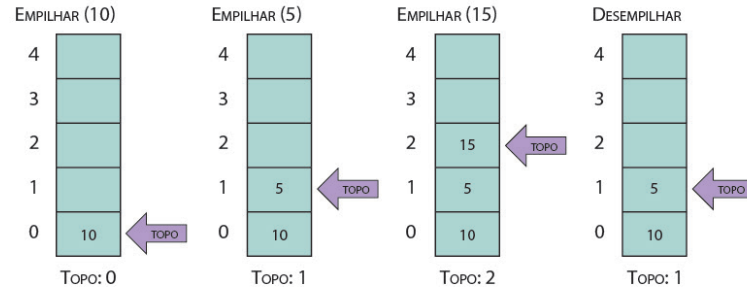
Um exemplo comum que pode representar esse conceito é o de uma pilha de pratos que estão guardados no armário: quando a pessoa vai utilizar um deles pega sempre o que se encontra no topo da pilha, assim como, quando um novo prato vai ser guardado, é colocado no topo. Isso acontece porque apenas uma das extremidades da pilha está acessível.

Operações Associadas:

1. Criar uma pilha P vazia;
2. Testar se P está vazia;

3. Obter o elemento do topo da pilha (sem eliminar);
4. Inserir um novo elemento no topo de P (empilhar);
5. Remover o elemento do topo de P (desempilhar).

EXEMPLO DE OPERAÇÃO NUMA PILHA



LEMBRETE:

A operação de inserção de elementos é denominada empilhamento e a de exclusão, desempilhamento.

9.2 IMPLEMENTAÇÃO ESTÁTICA NUMA PILHA

Para ilustrar a implementação de forma estática numa pilha foi utilizada uma estrutura de vetor para representar a pilha. Como a declaração de um vetor requer um tamanho máximo da estrutura, não é permitido inserir elementos quando a estrutura já estiver cheia. Portanto, para realizar o controle da estrutura, foi utilizada uma variável “topo” para indicar a posição do último elemento da pilha. Quando o valor do topo for igual ao máximo de elementos contidos na pilha, isso indicará que se chegou ao fim da mesma.

1. Algoritmo Pilha

```
2. var
3.     Tipopilha_reg = registro
4.         topo: inteiro
5.         elemento: vetor [1.50] de inteiros
6.     fim
7. pilha: pilha_reg
8. início
9. pilha.topo ← * 1
10. Função vazia ( ): lógica
11.     início
12.         Se(pilha.topo <= 0) então
13.             retorne.v.
14.         Senão
15.             retorne.f.
16.     fim*se
17. fim
18. Função cheia ( ): lógica
19. início
20.     Se(pilha.topo >= 50) então
21.         retorne.v.
22.     Senão
23.         retorne.f.
24.     fim*se
25. fim
26. Procedimento enfileirar (elem:inteiro)
27. início
28. Se (cheia ( )=.f..)então
```

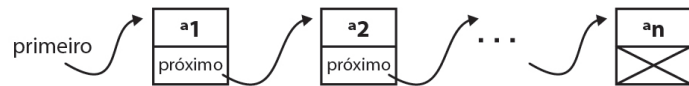
```
29.     pilha.topo ← pilha.topo + 1
30.     elemento.topo ← elem
31. Senão
32.     Mostre ("Pilha Cheia!")
33.     fim*se
34. Função desempilhar ( ):inteiro
35.     var
36.         valorDesempilhado: inteiro
37. início
38.     Se (vazia ( )=.f.)então
39.         Mostre ("Pilha vazia!")
40.         valorDesempilhado ← nulo
41.     Senão
42.         valorDesempilhado ← pilha.vetor [topo]
43.         pilha.topo ← pilha.topo*1
44.         retorne (valorDesempilhado)
45.     fim*se
46. Procedimento exhibePilha( )
47.     var
48.         i:inteiro
49. início
50.     Para(i ← 0 até topo)faça
51.         Mostre ("elemento", elemento [i], "posição", i)
52.     fim*para
53.     fim
54. fim
```

9.3 FILA

A fila é um método de acesso estruturado no qual os dados são inseridos num extremo e retirados noutra. Podemos fazer analogia às filas de supermercado, pois o funcionamento é similar. A fila se trata de uma estrutura de dados que se modifica dinamicamente. São chamadas de FIFO (“First In First Out” = “Primeiro a Entrar e o Primeiro a Sair”) ou LIFO (“Last In Last Out” = “Último a Entrar e o Último a Sair”).

Cada elemento (ou nó) da lista tem a seguinte estrutura:

- um atributo com o valor do elemento;
- um atributo com uma referência para o próximo elemento da lista (será nula se for o último elemento).
- a ordem dos elementos na lista é relevante.
- os elementos de uma lista são todos do mesmo tipo.



EXEMPLOS:

- Escalonamento de "Jobs": fila de processos aguardando os recursos do sistema operacional;
- Fila de pacotes a serem transmitidos numa rede;
- Simulação: fila de caixa em banco;
- Fila de impressão, em que os arquivos a serem impressos são organizados em uma lista e serão impressos na ordem de chegada, à medida que a impressora estiver disponível.

OPERAÇÕES ASSOCIADAS:

1. **Criar** – cria uma fila vazia;
2. **Vazia** – testa se uma fila está vazia;
3. **Primeiro** – obtém o elemento do início de uma fila;
4. **Inserir** – insere um elemento no fim de uma fila;
5. **Remover** – remove o elemento do início de uma fila, retornando o elemento removido.

Para definir a estrutura de uma fila por vetor é necessário construir um registro que contenha as informações da fila, tais como o início, o

final e o array de elementos (vetor); nesse caso, cada um dos elementos representado por uma posição no vetor. Segue abaixo exemplo de uma fila implementada com vetor:

```
Algoritmo Fila_Vetor
var
    Tipo fila_reg = registro
        inicio: inteiro
        fim: inteiro
        elemento: vetor [1.50] de inteiro
    fim
    total: inteiro
    fila: fila_reg
inicio
    fila.inicio ← 0
    fila.fim ← 0
    total ← 0
Função vazia ( ): lógica
    inicio
        Se(total = 0) então
            return .v.
        Senão
            return .f.
    fim-se
```

```
    fim
Função cheia ( ): lógica
    inicio
        Se(total >=50) então
            return .v.
        Senão
            return .f.
    fim-se
fim
Procedimento enfileirar (elem: inteiro)
    inicio
        Se(cheia ( ) = .f..) então
            fila.elemento[inicio] <- elem
            fila.fim <- fila.fim + 1
            total <- total + 1
            Se(fila.fim >= 50) então
                fila.fim = 0
            fim-se
        Senão
            Mostre("Fila cheia")
        fim-se
    fim
Função desenfileirar ( ): inteiro
    var
        excluido: inteiro
```

```

    inicio
    Se (vazia ( ) = .f.) então
        excluido <- fila.elemento[inicio]
        fila.inicio <- fila.inicio + 1
        Se (fila.inicio > = tamanho) então
            fila.inicio <- 0
        fim-se
        total <- total - 1
        retorne excluido
    Senão
        excluido <- nulo
        retorne excluido
    fim-se
fim

Procedimento exibefila ( )
    var
        i: inteiro
    inicio
        Para (i<- 0 até total) faça
            Mostre ("Posição ", i, " valor ", elemento
[i] )
        fim-para
    fim
fim

```

Observe que a variável elemento é do tipo vetor e comporta até 50 números inteiros.

OPERAÇÕES COM FILA

Todas as operações numa fila podem ser imaginadas como as que ocorrem numa fila de pessoas num supermercado, exceto os elementos que não se movem na fila, conforme o primeiro elemento é retirado. No âmbito computacional isto exigiria muito para o computador, pois o que se faz é indicar quem é o primeiro elemento.

Supondo uma fila com capacidade para 5 elementos.

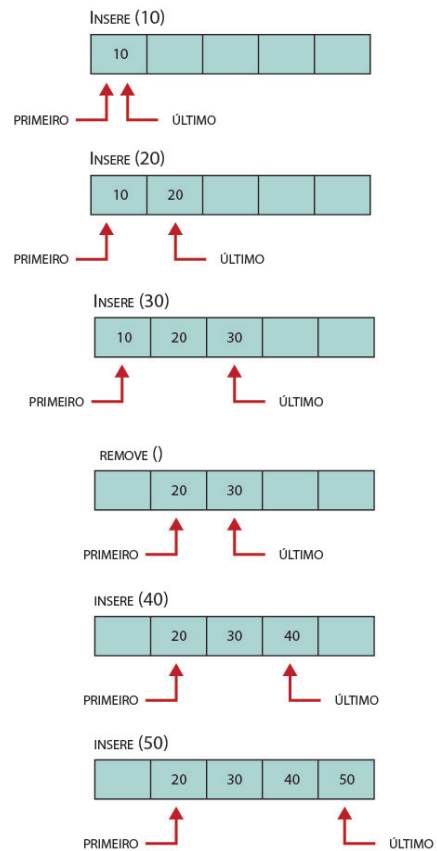


Figura 23 – Elementos de uma fila
Fonte: SENAI DR PR, 2018.

10 ALGORITMOS DE BUSCA

Os algoritmos de busca representam um conjunto de algoritmos de elevada importância na computação. Em diversas situações há um algoritmo que é empregado para buscar o termo (um nome de pessoa, um endereço, um CPF, etc.) que foi requisitado pelo usuário. Seguem abaixo dois tipos de algoritmo de busca.

10.1 PESQUISA SEQUENCIAL

É o método de pesquisa mais simples e consiste numa varredura da tabela (vetor, matriz ou arquivo), durante a qual o argumento de pesquisa é comparado com a chave de cada entrada até ser encontrada uma igual ou ser atingido o final da tabela, caso a chave procurada não exista.

Como exemplo é apresentado o algoritmo de pesquisa sequencial em uma tabela não ordenada. No caso de uma busca sem êxito, serão necessárias n comparações. Será adotado para a implementação abaixo uma função que recebe como parâmetro um vetor de elementos inteiros (v), a quantidade de elementos do vetor (n) e a chave a ser pesquisada (k), retornando o endereço (índice) do elemento encontrado, ou -1 caso contrário.

```

algoritmo "Busca Sequencial"
var
    vet : vetor [1..20] de inteiro
    x, busca : inteiro
    achou : logico
inicio
    // preencher o vetor
    para x de 1 ate 20 faca
        vet[x] <- randi(10) + 1 // gera de 0 a 9, e
soma 1
    fimpara
    escreval("Digite um valor entre 1 e 10: ")
    leia(busca)
    // busca no vetor
    para x de 1 ate 20 faca
        se (busca = vet[x]) entao
            achou <- verdadeiro
            escreval("Valor encontrado na posição ", x)
        fimse
    fimpara
    se (achou = falso) entao
        escreval("Valor não encontrado!")
    fimse
fimalgoritmo

```

10.2 PESQUISA BINÁRIA

A pesquisa binária é um método que pode ser aplicado a tabelas ordenadas, armazenadas em dispositivos de acesso direto. Este método verifica o elemento central e se analisa se este elemento é maior que a chave. O método testa o elemento central da primeira metade, caso contrário, ele testa o elemento central da segunda metade. O método consiste na comparação do argumento de pesquisa (k) com a chave localizada no meio da tabela. Se for igual, a pesquisa termina com sucesso. Se k for maior, o processo é repetido para a metade superior da tabela e se for menor, para a metade inferior. Assim, a cada comparação, a área de pesquisa é reduzida à metade do número de elementos.

Como exemplo, para encontrar o número 4 na matriz 1 2 3 4 5 6 7 8 9, uma pesquisa binária primeiramente testa o elemento médio, nesse caso 5. Visto que é maior que 4, a pesquisa continua com a primeira metade ou 1 2 3 4 5. O elemento central agora é 3, que é menor que 4, então a primeira metade é descartada. A pesquisa continua com 4 5. Nesse momento o elemento é encontrado.

Segue abaixo um algoritmo de pesquisa binária:


```

//Efetuar a pesquisa binária
  inicial <- 1
  final <- 10
  dado_encontrado <- falso
  enquanto (inicial <= final) e nao dado_encontrado
faca
  meio <- (inicial + final) DIV 2
  se VET[meio] = busca entao
    dado_encontrado <- verdadeiro
  fimse
  se VET[meio] > busca entao
    final <- meio - 1
  senao
    inicial <- meio + 1
  fimse
fimenquanto
//Exibir Resultados da busca
se dado_encontrado = verdadeiro entao
  escreva ("Dado encontrado na posição", meio)
senao
  escreva ("Informação não encontrada no vetor")
fimse

```

10.3 AVALIAÇÃO DOS MÉTODOS DE BUSCA APRESENTADOS

Um dos principais critérios na avaliação dos métodos de busca é o número médio de comparações, calculando-se o número médio de comparações efetuadas com algum elemento do vetor pesquisado. A tabela abaixo compara os métodos de busca apresentados anteriormente (a variável N, que aparece nas fórmulas, refere-se à quantidade de itens do vetor pesquisado).

CRITÉRIO ANALISADO	SEQUENCIAL	BINÁRIA
Número médio de comparações para pesquisa bem-sucedida.	$N/2$	$2 * \log_2 N$
Número médio de comparações para pesquisa incerta.	$N - (N * P/2)$	$2 * \log_2 N$
Número máximo de comparações para pesquisa sem sucesso	N	$2 * ((\log_2 N) + 1)$

Tabela 6 – Tabela comparativa

Fonte: SENAI DR PR, 2018.

11 ALGORITMOS DE ORDENAÇÃO

Os algoritmos de ordenação são os que direcionam para a ordenação ou reordenação de valores apresentados em uma determinada sequência, para que os dados possam ser acessados posteriormente de forma mais eficiente. Uma das principais finalidades desse tipo de algoritmo é a ordenação de vetores, uma vez que, em uma única variável pode-se ter inúmeras posições. Exemplo: a organização de uma lista de chamada escolar para que a lista fique em ordem alfabética.

11.1 BUBBLE SORT

O algoritmo Bubble Sort tem o objetivo de realizar a ordenação de arrays e listas dinâmicas. Se o objetivo for ordenar os valores em forma decrescente, então, a posição atual do array é comparada com a próxima posição e se a posição atual for maior que a posição seguinte é realizada

a troca dos valores nessa posição. Se a condição não for atendida, a troca não é realizada, passando-se assim para o próximo elemento e o processo de comparação é novamente realizado e assim sucessivamente.

Se o objetivo for ordenar os valores do array de forma crescente, a posição atual é comparada com a próxima e, se a posição atual for menor que a posição posterior, é realizada a troca. Se a condição não for atendida, a troca não é realizada, passando para o elemento seguinte para a realização da comparação.

Um array ou lista pode estar ordenado no momento em que sua ordenação é solicitada, então, esta situação deve ser prevista na implementação do algoritmo.

Segue abaixo exemplo da ordenação em ordem crescente:

algoritmo "Bubble Sort"

```
var
  a, b : inteiro
  temp : real
  x : vetor[1..10] de real
inicio
  // Leitura dos dados
  aleatorio 0,100,3
  para a de 1 ate 10 faca
    leia(x[a])
  fimpara
  // Ordenação
  para a de 1 ate 10 faca
    para b de 1 ate 9 faca
      se x[b] > x[b+1] entao
        temp <- x[b]
        x[b] <- x[b+1]
        x[b+1] <- temp
      fimse
    fimpara
  fimpara
  // Impressão dos dados ordenados
  para a de 1 ate 10 faca
    escreval(a:3," - ", x[a] : 10 : 3)
  fimpara
fimalgoritmo
```

11.2 INSERTION SORT

O algoritmo Insertion Sort é de fácil implementação, similar ao Bubble Sort, e seu funcionamento é feito por comparação e inserção direta. À medida que o algoritmo percorre a lista de elementos, o mesmo os organiza um a um em sua posição mais correta, onde o elemento a ser alocado (k) terá à sua esquerda um valor menor (k-1) e de maneira similar, à sua direita um valor maior (k+1).

O funcionamento do algoritmo ocorre da mesma forma que as pessoas usam para ordenar um jogo de cartas.

```
para j ← 2 até comprimento do vetor faça
  elemento ← vetor[j]
  i ← j - 1
  enquanto i > 0 e vetor[i] > elemento, faça
    vetor[i + 1] ← vetor[i]
    i ← i - 1
  fimenquanto
  vetor[i + 1] ← elemento
fimpara
```

11.3 SELECTION SORT

A ideia do algoritmo Selection Sort é buscar o elemento que contém o menor valor do vetor e colocá-lo no início. É buscado o menor valor do vetor e este é colocado na posição [1] e assim sucessivamente até todos elementos do vetor estarem ordenados.

Sempre a partir do último elemento reordenado (a partir do i), o programa procura o menor elemento no vetor e o substitui pelo elemento i atual.

```
para i ← 1 até tamanho-1, faça
    minimo ← i
    para j ← i+1 até tamanho, faça
        se vetor[j] < vetor[minimo], então
            minimo ← j
        fimse
    fimpara
    temp ← vetor[i]
    vetor[i] ← vetor[minimo]
    vetor[minimo] ← temp
fimpara
```

11.4 MERGE SORT

Este algoritmo realiza a reordenação de uma estrutura através de uma quebra, intercalação e união dos vários elementos existentes. Para isto, o algoritmo divide o problema em partes menores, resolve cada parte e depois junta (merge) os resultados para obter o resultado final. Inicialmente o vetor é dividido em duas partes iguais e estas partes serão divididas em mais duas partes e assim sucessivamente até ficarem um ou dois elementos cuja ordenação é trivial.

Para juntar (realizar o merge), as partes ordenadas dos dois elementos de cada parte são separadas e o menor deles é selecionado e retirado de sua parte. Feito isso, as subestruturas menores (agora ordenadas) serão unidas, sendo seus elementos ordenados por meio de intercalação.

A imagem a seguir ilustra o funcionamento do algoritmo:

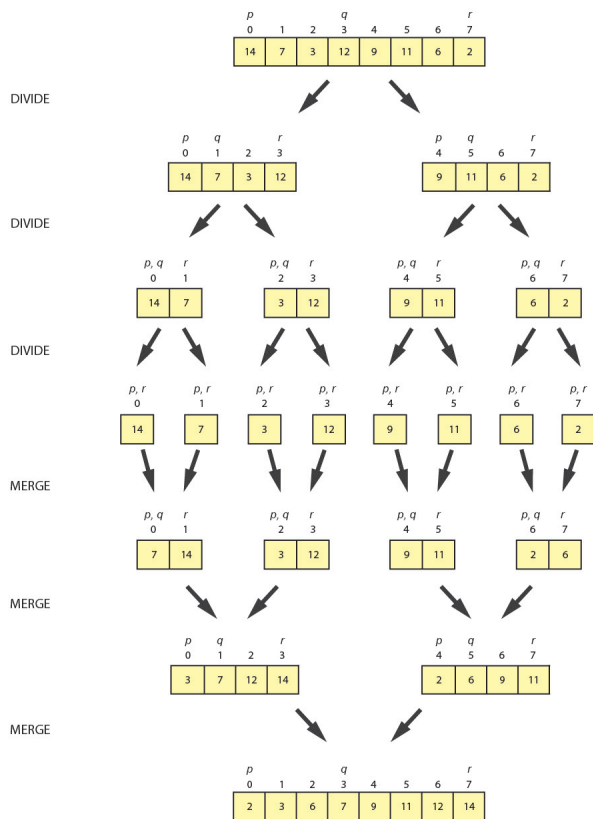


Figura 24 – Ordenação por Merge Sort

Fonte: <https://pt.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/overview-of-merge-sort>

Conforme mencionado, o Merge Sort possui duas etapas, como ilustrado na imagem.

1. divisão do vetor em subvetores até atingirmos o caso base;
2. fusão dos subvetores (merge).

A primeira etapa consiste no seguinte algoritmo:

```

01.mergesort(A[0...n - 1], inicio, fim)
02.|   se(inicio < fim)
03.|   |   meio ← (inicio + fim) / 2 //calcula o meio
04.|   |   mergesort(A, inicio, meio) //ordena o
      subvetor esquerdo
05.|   |   mergesort(A, meio + 1, fim) //ordena o
      subvetor direito
06.|   |   merge(A, inicio, meio, fim) //funde os
      subvetores esquerdo e direito
07.|   fim_se
08.fim_mergesort

```

A segunda etapa é a de fusão:

```

01. merge(A[0...n - 1], inicio, meio, fim)
02. |   tamEsq ← meio - inicio + 1 //tamanho do
subvetor esquerdo
03. |   tamDir ← fim - meio //tamanho do subvetor
direito
04. |   inicializar vetor Esq[0...tamEsq - 1]
05. |   inicializar vetor Dir[0...tamDir - 1]
06. |   para i ← 0 até tamEsq - 1
07. |   |   Esq[i] ← A[inicio + i] //elementos do
subvetor esquerdo
08. |   fim_para
09. |   para j ← 0 até tamDir - 1
10. |   |   Dir[j] ← A[meio + 1 + j] //elementos do
subvetor direito
11. |   fim_para
12. |   idxEsq ← 0 //índice do subvetor auxiliar
esquerdo
13. |   idxDir ← 0 //índice do subvetor auxiliar
direito
14. |   para k ← inicio até fim
15. |   |   se(idxEsq < tamEsq)
16. |   |   |   se(idxDir < tamDir)
17. |   |   |   |   se(Esq[idxEsq] < Dir[idxDir])
18. |   |   |   |   |   A[k] ← Esq[idxEsq]

```

```

19. |   |   |   |   |   idxEsq ← idxEsq + 1
20. |   |   |   |   |   senão
21. |   |   |   |   |   A[k] ← Dir[idxDir]
22. |   |   |   |   |   idxDir ← idxDir + 1
23. |   |   |   |   |   fim_se
24. |   |   |   |   |   senão
25. |   |   |   |   |   A[k] ← Esq[idxEsq]
26. |   |   |   |   |   idxEsq ← idxEsq + 1
27. |   |   |   |   |   fim_se
28. |   |   |   |   |   senão
29. |   |   |   |   |   A[k] ← Dir[idxDir]
30. |   |   |   |   |   idxDir ← idxDir + 1
31. |   |   |   |   |   fim_se
32. |   |   |   |   |   fim_para
33. fim_merge

```

11.5 QUICK SORT

O Algoritmo Quicksort criado por C. A. R. Hoare em 1960, é um dos métodos de ordenação mais rápidos que se conhece para uma ampla variedade de situações.

É um algoritmo de comparação que utiliza a estratégia de “divisão e conquista”. A ideia consiste em dividir o problema de ordenação de um conjunto com n itens em dois problemas de menor complexidade. Os problemas menores são ordenados e os resultados são combinados para produzir a solução final.

Resumidamente, o funcionamento do algoritmo funciona da seguinte forma:

O algoritmo divide seu vetor de entrada em dois sub-vetores a partir de um pivô para em seguida realizar o mesmo procedimento nas duas listas menores até uma lista unitária.

Funcionamento do algoritmo:

- É escolhido um elemento da lista que será denominado pivô;
- Reorganiza o vetor de forma que os elementos menores que o pivô fiquem de um lado e os maiores fiquem de outro. Esta operação é chamada de “particionamento”;
- Recursivamente ordena o sub-vetor abaixo e acima do pivô.

```
01. QuickSort (vet:INTEIRO[], inicio:INTEIRO,
fim:INTEIRO):NEUTRO
02. INICIO
03.   var i,j,pivo,aux: INTEIRO
04.   i=inicio
05.   j=fim
06.   pivo=(inicio+fim) DIV 2
07.   ENQUANTO (i<j) FACA
08.     ENQUANTO (vet[i]<vet[pivo]) FACA
09.       i++
10.     FIMENQUANTO
11.     ENQUANTO (vet[j]>vet[pivo]) FACA
12.       j--
13.     FIMENQUANTO
14.     SE (i<=j) ENTAO
15.       aux = vet[i]
16.       vet[i] = vet[j]
17.       vet[j] = aux
18.       i++
19.       j--
20.     FIMSE
21.   FIMENQUANTO
22.   SE (j>inicio) ENTAO
23.     QuickSort (vet, inicio, j)
```



```
24.     FIMSE
25.     SE (i<fim) ENTAO
26.         QuickSort (vet,i,fim)
27.     FIMSE
28. FIM
29.
30. PRINCIPAL():NEUTRO
31. INICIO
32. var vet = {55, 76, 26, 64, 26, 80, 71, 46}
33. QuickSort(vet,0,vet.tamanho-1)
34. FIM
```


12 INDENTAÇÃO E COMENTÁRIOS DE CÓDIGO

De maneira geral um código legível é direto e fácil de compreender, não possui duplicidades e faz apenas o que é proposto. Para auxiliar na busca por este resultado, a seguir serão descritas boas práticas que visam a criação de um código fonte organizado.

12.1 INDENTAÇÃO

Indentação é a forma como o código é organizado, ou seja, é uma forma de hierarquizar todas as linhas do programa a fim de tornar o código mais legível. Inicialmente é importante decidir se serão aplicadas tabulações ou espaços e, após isso, definir o tamanho da indentação.

A indentação não é obrigatória, porém, é uma prática entre os programadores e entregar um código não indentado pode ser considerado desorganização.

Em relação ao comprimento das linhas, é importante definir um limite. Linhas muito longas tendem a ser confusas e os limites normalmente utilizados são 60, 80 ou 100 caracteres. A quebra da linha pode ser feita manualmente de forma a não prejudicar o entendimento do conteúdo.

EXEMPLO:

```
#include <stdio.h>

// =====
// ===== www.eXcript.com =====
// =====

//comentário de uma única linha

/*
comentário de várias linhas
*/

void main(){
    printf("ddd");
    return 0;
}
```

12.2 COMENTÁRIOS

Uma prática que facilita a compreensão de um código é colocar comentários nas partes que precisam de explicação. Isto torna a análise

mais otimizada e favorece a continuidade do desenvolvimento da codificação.

Para criar um comentário de uma linha, basta colocar `/**` seguido do texto que é o seu comentário. Caso você precise escrever muito texto, aí use o comentário multilinhas. Ele é feito com a seguinte sintaxe: `/*` seguido do texto que pode estar em várias linhas seguido de `*/`.

EXEMPLO:

```
1  /*
2  * Esse é o primeiro programa.
3  * O resultado dele é mostrar o texto "Olá, mundo!"
   na tela!
4  */
   programa
5  { // início do programa
6  funcao inicio()
7      {
8          escreva("Olá, mundo!");
9      }
10 } /* fim do programa */
```

REFERÊNCIAS

PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados com aplicações em Java**. 2. ed. São Paulo: Pearson Prentice Hall, 2009.

_____. **Lógica de programação e estrutura de dados, com aplicações em Java**. 3. ed. São Paulo: Pearson Education Brasil, 2016.

EDELWEISS, N.; GALANTE, R. **Estruturas de dados**. Porto Alegre: Bookman, 2009.

WIRTH, N. **Algoritmo e estrutura de dados**. Rio de Janeiro: Editora LTC, 1986.

MANZANO, J.A.N.G. **Estudo dirigido: algoritmos**. 5. ed. São Paulo: Editora Érica, 2000.

FORBELLONE, A.L.V.; EBERSPACHER, H.F. **Lógica de programação: a construção de algoritmos e estruturas de dados**. 3. ed. São Paulo, SP: Pearson Prentice Hall, 2005.

SZWARCFITER, J. L.; Markezon, L. **Estruturas de dados e seus algoritmos**. 3. ed. Rio de Janeiro: Editora LTC, 2015.

SCHIMIGUEL, J. **Lógica: uma ferramenta indispensável na programação de computadores**. Devmedia, 2013. Disponível em: <<https://www.devmedia.com.br/logica-uma-ferramenta-indispensavel-na-programacao-de-computadores/28386>>. Acesso em: 23 mai 2018.

MINICURRÍCULO

Hilosi José Nunes Miamoto é graduado em Engenharia da Computação pela UNOPAR - Universidade Norte do Paraná (2003); possui pós-graduação Lato Sensu em Administração de Empresas pela FGV – Fundação Getúlio Vargas (2005); pós-graduação Lato Sensu em Docência da Educação Profissional e Tecnológica pelo SENAI CETIQT (2015); Atualmente cursa pós-graduação em Robótica com Ênfase em Tecnologia da Educação pelo SENAI CIC (2018-2019). Também possui graduação pelo Programa Especial de Formação Pedagógica em Matemática pelo Centro Universitário Claretiano (2012). Atualmente, é técnico de ensino e coordenador técnico na unidade SENAI Campus da Indústria.

<http://lattes.cnpq.br/9852886537285671>

CRÉDITOS

SENAI – DEPARTAMENTO REGIONAL DO PARANÁ

José Antônio Fares

Diretor Regional

Giovana Chimentao Punhagui

Gerente Executiva de Educação

Vanessa Sorda Frason

Gerente de Educação Profissional

Jacielle Feltrin Vila Verde Ribeiro

Sandra Cristina Brasil Toloto

Coordenação Pedagógica

Hilosi José Nunes Miamoto

Elaboração

Anderson Paulo Avila Santos

Revisão Técnica

Alexandre Luis Kloch

Coordenação Técnica

Erica Luz de Souza

Orientação Pedagógica

Iris Jerusa D'Amico Burger

Karem Morigi

Revisão Ortográfica e Gramatical

Daniel Gustavo Hella

Estela Pereira

Michelle Kesselring Ferreira da Costa

Coordenação de Projeto

Anderson Calixto de Carvalho

André Dias

Aline Sentone

Ilustrações e Tratamento de Imagens

Ricardo Luiz Freire de Menezes Junior

Projeto Gráfico

Ricardo Luiz Freire de Menezes Junior

Diagramação, Revisão de Arte e Fechamento de Arquivo

