ETE JUSCELINO KUBITSCHEK

CURSO TÉCNICO DE INFORMÁTICA

LINGUAGEM DE PROGRAMAÇÃO I

2020 / 2021

(REVISÃO #1)

ETEJK > INFORMÁTICA > LTP1

CAPITULO I - INTRODUÇÃO

O DESENVOLVIMENTO DE UM SOFTWARE

Um programa de computador ou simplesmente software representado pelas instruções e dados que algum ser humano definiu e que ao serem executados por alguma máquina cumprem algum objetivo. A máquina é um computador digital. Os computadores digitais são máquinas eletrônicas contendo processadores e circuitos digitais adequados, operando, com sinais elétricos em dois níveis ou binários.

Os dados são organizados em um computador de acordo com sua representação binária, isto é, sequências de 0s e 1s. O objetivo de se utilizar um computador é extrair as informações resultantes de computações, isto é, o resultado das execuções das instruções de algum programa. Deve-se observar a diferença entre informação e dado: o dado por si só é um valor qualquer armazenado em um computador enquanto a informação representa a interpretação desse dado, ou seja, qual o seu significado.

Parte dos dados processados durante a execução de um software é fornecida pelo ser humano (ou outra máquina) e denominada dados de entrada. Por outro lado, os dados de saída são aqueles fornecidos ao ser humano (ou a outra máquina) após o processamento dos dados de entrada.

De qualquer forma, é importante notar que o objetivo do software é que motiva sua construção. Este pode ser definido como alguma necessidade humana, por exemplo, um programa para simular o funcionamento de um circuito digital. Um programa para comandar um robô em uma linha de montagem, um sistema de gerenciamento de informações em uma empresa, somente para citar algumas. Abaixo, apresenta-se uma figura que simplifica o processo de desenvolvimento de um software.

Na figura acima, o cliente espefificaextamente o que o software deve conter. Ele sabe o que o software deve conter e realizar, mas regra geral não sabe como. Ele indica o que o software deve contemplar e executar por meio de especificações chamadas requisitos.

Entendente-se por cliente a entidade que contrata os serviços para criação de um software, podendo ser uma empresa, pessoa ou ainda uma empresa que, por iniciativa própria, produza e venda seu software livremente (neste caso, um exemplo seria a Microsoft).

No desenvolvimento, os requisitos do cliente são traduzidos em especificações técnicas de software pelos analistas de sistemas ou engenheiros de software. O desenviolvimento de um software é tipicamente dividido nas seguintes etapas:

- Análise: criam-se especificações que detalham como o software vai funcionar;
- Projeto: criam-se especificações que detalham o resultam da análise em termos mais próximos da implementação do software;
- Implementação: utilizando uma linguagem de programação e as especificações de projeto, o software é construído;
- Testes: após a construção do software, são realizados testes para conferir sua conformidade com os requisitos funcionais iniciais. O software

deve satisfazer a todas especificações do cliente.

Por fim, após os testes o software é implantado na empresa. A implantação pode variar desde uma simples instalação, que dure alguns minutos, até a instalação e testes de integração de diversos softwares, que pode levar semanas. De qualquer forma, o fato de o software estar finalizado e testado não significa que esteja totalmente livre de erros, também, denominados bugs. Assim, deve-se voltar e tentar identificar a causa dos erros.

Pior que erros de programação, é o caso em que pode acontecer de o software funcionar corretamente, não apresentar erros, mas realizar o que o cliente esperava. Nesse caso, deve-se retornar à etapa inicial, verificando os requisitos e refazendo todo o ciclo de desenvolvimento novamente.

É um fato que grande parte do investimento feito em um software é gasta na correção de erros do que propriamente na sua elaboração. Daí, surge a necessidade de enxergar o software como o produto de um produto de um processo bem-definido e controlado, que atue sobre as suas etapas de desenvolvimento, em outras palavras, um processo de engenharia de software.

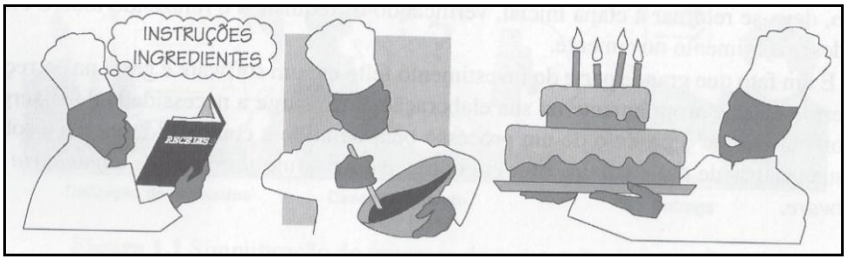
ALGORITMOS E LÓGICA DE PROGRAMAÇÃO

O software deve ser encarado como um produto de um processo bem-definido e controlado de engenharia. O estudo de algoritmos e de lógica de programação é essencial no contexto do processo de criação de um software. Ele está diretamente relacionado com a etapa de projeto de um software em que, mesmo sem saber qual será a linguagem de programação a ser utilizada, se especifica completamente o software a ponto de na implementação ser possível traduzir diretamente essas especificações em linhas de códigos em alguma linguagem de programação como Pascal, C, C++, Java, Python e outras.

Essta tarefa permite verificar, em um nível maior de abstração, se o software está correto ou não. Permite, inclusive, averiguar se o software atenderá às especificações originalmente propostas. Assim, evita-se partir diretamente para a etapa de implementação, o que poderá ocasionar mais erros no produto final.

O SIGNIFICADO DE UM ALGORITMO

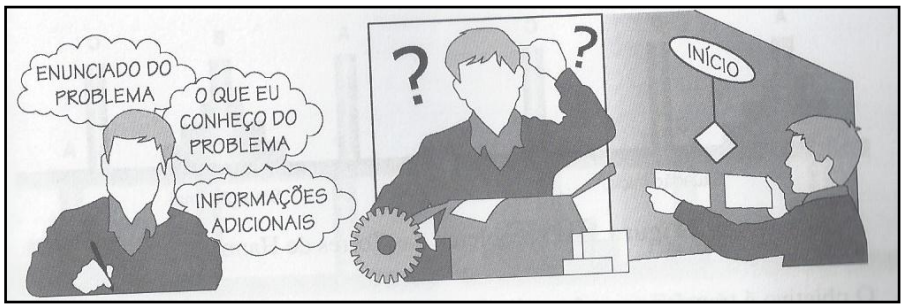
Um algoritmo representa um conjunto de regras para a solução de um problema. Essa é uma definição geral, podendo ser aplicada a qualquer circunstância que exija a descrição da solução. Dessa forma, uma receita de bolo é um exemplo de algoritmo, pois decreve as regras necessárias para a conclusão de seu objetivo: a preparação de um bolo.



Em um bolo, descrevem-se quais serão os ingredientes e as suas quantidades. Depois, quais são as regras para o seu preparo, como a sequencia de inclusão dos ingredientes para bater as gemas; cozimento e assim por diante.

A correta execução das instruções contidas na receita de bolo leva à sua preparação. No entanto, se essas instruções tiverem sua ordem trocada ou a quantidade dos ingredientes alterada, o resultado vai divergir do original. Existe, ainda, o perigo de o autor da receita não a ter testado previamente, o que poderá gerar, novamente, resultados indesejáveis.

Da mesma forma, em programação, o algoritmo especifica com clareza e deforma correta as insctruções que um software deverá conter para que, ao ser executado, forneça resultados esperados.



Em primeiro lugar, deve-se saber qual é o problema a ser resolvido pelo software – o seu objetivo. Daí, deve-se extrair todas as informações a respeito desse problema (dados e operações), realcioná-las com o conhecimento atual que se tem do assunto, buscando eventualmente informações de outras fontes. Essa fase representa a modelagem do problema em questão. A modelagem do problema é resultante de um processo mental de abstração.

Depois, sabendo como resolver o problema, a tarefa consiste em descrever claramente os passos para se chegar à sua solução. Os passos por si só não resolvem o problema; é necessário colocá-los em uma sequência lógica, que, ao ser seguida, de fato o solucionará.

Além disso, é importante que essa descrição possua algum tipo de convenção para que todas as pessoas envolvidas na definição do algoritmo possam entendê-lo. Chega-se, então, na especificação do algoritmo.
A FORMALIZAÇÃO DE UM ALGORITMO

A tarefa de especificar os algoritmos para representarum programa consiste em detalhar os dados que serão processados pelo programa e as instruções que vão operar sobre esses dados. Essa especificação pode ser feita livremente como visto anteriormente, mas é importante formalizar a descrição de algoritmos segundo alguma convenção, para que todas as pessoas envolvidas na sua criação possam entendê-lo da mesma forma.

Em primeiro lugar, é necessário definir um conjunto de regras que regulem a escrita do algoritmo, isto é, regras de sintaxe. Em segundo, é preciso estabelecer as regras que permitam interpretar um algoritmo, que são as regras semânticas.

A SINTAXE DE UM ALGORITMO

A sintaxe de um algoritmo resume-se nas regras para escrevê-lo corretamente. Em computação, essas regras indicam quais são os tipos de comandos que podem ser utilizados e também como eles escrever expressões. As expressões de um comando em um algoritmo realizam algum tipo de operação com os dados, isto é, operam com valores e resultam em outros valores que são usados pelo algoritmo.

Os tipos de comandos de um algoritmo são também denominados estruturas de programação. Existem apenas três tipos de estruturas que podem ser utilizadas para escrever qualquer programa: estruturas equenciais, de decisão e de repetição. Convencionou-se que os dados manipulados por um programa são categorizados em tipos de dados, que torna simples seu uso para o programador, mas que na realidade, para a máquina, são traduzidos em valores binários. Assim é possível manipular diversos tipos de dados em um algoritmo: números inteiros e reais, valores lógicos, textos, etc.

A manipulação desses dados é feita por meio de variáveis e valores constantes, que representam no texto do algoritmo os dados que serão armazenados na memória do computador. O significado de variável é similar àquele empregado na matemática: representar um valor, porém com um significado físico por trás; esse valor será armazenado na memória do computador. Um valor constante representa um valor que não pode ser alterado, como o número 25, o nome ‘Márcio’ e assim por diante.

Uma variável pode ser manipulada de muitas formas. Os valores constantes ou resultados de expressões envolvendo as variáveis podem ser atribuídos a estas.

Para escrever as expressões corretamente, é necessária também uma sintaxe. Essa sintaxe depende do tipo de variáveis envolvidas e determina quais são os operadores que podem ser aplicados. Além dos operadores propriamente ditos, especificam-se algumas funções e procedimentos predefinidos úteis, que simplificam algumas tarefas corriqueiras em computação, como ler os dados digitados por um usuário, escrever resultados na tela do computador, calcular o seno de um número, etc.

EXEMPLO DE SINTAXE DE UM ALGORITMO

Deseja-se especificar um algoritmo para calcular e exibir na tela a área de um triângulo de base b e altura h, em que os valores de b e h são fornecidos pelo usuário via teclado. Antes de mais nada, a solução desse problema é imediata, pois sabe-se que a área s de um triângulo de base b e altura h é dada por s=(bxh)/2.

Apresentamos abaixo, dois algoritmos para resolver esse problema: um de uma maneira informal, descrito em português e sem formatação; e outro, em portugês estruturado (ou portugol) utilizando-se uma representação em pseudocódigo.

Algoritmo em Português	Algoritmo em Portugol
Pedir para o usuário digitar os valores de b e de h; Calcular a área de s usando a fórmula s= (bxh)/2; Exibir o valor de s na tela.	Inicio Leia (b,h); s←(b*h)/2; Exiba (s); Fim

A SEMÂNTICA DE UM ALGORITMO

A semântica estabelece regras para sua interpretação. Os símbolos ou comandos de um algoritmo por si só não têm um significado, a menos que este seja bem-definido.

Dessa forma, a semântica de um algoritmo sempre acompanha a sua sintaxe, fornecendo um significado. A importância da formalização de um algoritmo, sua sintaxe e semântica podem ser resumidos assim:

- Evitar ambiguidades, pois definem regras sintáticas e semânticas que sempre são interpretadas da mesma forma;
- Impedir a criação de símbolos ou comandos desnecessários na criação de um algoritmo: representa um conjunto mínimo de regras que pode ser utilizado em qualquer algoritmo;
- Permitir uma aproximação com as regras de uma linguagem de programação, fazendo assim, uma fácil tradução de um algoritmo para sua implementação no computador.

COMO RESOLVER PROBLEMAS

A criação de um algoritmo é uma tarefa essencialmente intelectual. A partir do enunciado de um problema, deseja-se obter um algoritmo que o resolva. Pode-se afirmar que a tarefa de escrever algoritmos é, portanto, uma tarefa de resolver problemas.

A ANÁLISE E A SÍNTESE DE UM PROBLEMA

A resolução de um problema envolve duas grandes fases: a análise e a síntese da solução.

Na fase de análise, o problema é entendido de forma que se descubra o que deve ser solucionado, quais são os dados necessários e as condições para resolvê-lo se esses dados e essas condições são necessários, insuficientes ou redundantes ou ainda contraditórios e, então, parte-se para a sua modelagem, podendo ser enriquecida com o auxilio de equações, desenhos ou gráficos. Como resultado dessa fase, tem-se a elaboração deum plano de ação, no qual a experiência em problemas similares vistos anteriormente é utilizada e, também, pode ser necessária a utilização de problemas auxiliares. Nessa fase, faz-se uso direto de processos de abstração, o que significa

elaborar modelos mentais do problema em questão e do encaminhamento de sua solução.

Na etapa de síntese, executa-se o plano definido na fase de análise, representando os passos por meio de um algoritmo. Aqui, emprega-se uma representação formal. É importante que a solução seja verificada e comprovada corretamente, por meio da execução do algoritmo. Esta execução é feita percorrendo-se o algoritmo do seu início até o seu fim, e verificando, a cada passo, se o resultado esperado foi obtido. Caso tenha sido encontrada alguma discrepância, deve-se procurar saber qual foi a sua causa e eventualmente analisar novamente o problema, repetindo-se assim esse ciclo até que a solução tenha sido obtida.

MODELAGEM DE PROBLEMAS

A modelagem é a principal responsável pela facilidade ou dificuldade da resolução de um problema. Na Matemática e na Engenharia, por exemplo, o uso da linguagem matemática é fundamental, principalmente peleliminação de duplos sentidos que acontecem nessa prática. O mesmo ocorre na Computação, com o emprego de linguagens de descrição de algoritmos e de linguagens de programação.

O PAPEL DA LÓGICA EM PROGRAMAÇÃO

A Lógica é uma área da Mátemática cujo objetivo é investigar a veracidade de suas proposições. O papel da Lógica em programação de computadores está relacionado com a correta sequência de instruções que devem ser definidas para que o programa atinja seu objetivo. Serve como instrumento para verificação do programa escrito, provando se este está correto ou não

Em um algoritmo em execução, o valor das suas variáveis a cada instante representa o seu estado. Com a execução dessas instruções, esse estado vai sendo alterado. Um algoritmo correto é aquele que, a partir de um estado incial de suas variáveis,

consegue com a execução de suas instruções, chegar a um estado final, no qual os valores das variáveis estão de acordo com a solução esperada.

COMO ENFRENTAR UMA DISCIPLINA DE ALGORITMOS

O grande problema apresentado pelos estudantes em um curso de programação é a dificuldade em abstrair e descrever as soluções de problemas contando apenas com poucas e simples estruturas.

Um novo problema de computação pode ser gerado a partir de um já existente, alterando-se apenas poucos elementos de seu enunciado. Dessa forma, é um erro decorar as soluções em computação, pois podem não servir para outros entendimentos, que com certeza serão diferentes. O que deve ser procurado é o entendimento de como foi obtida uma solução, armazená-la na memória e então utilizar essa experiência adaptando-a a outras situações, por analogia, generalização ou especialização. A grande dica é que um problema pode ser diferente de outro, mas consegue-se aproveitar grande parte da experiência obtida em problemas passados em novos desafios

Não existe em Computação uma “fórmula mágica” para resolver problemas. De qualquer forma, apresenta-se a seguir um conjunto de dicas que podem ser utilizadas durante o processo de raciocínio empregado na resolução de problemas:

Dicas	Soluções
Ao se deparar com um problema novo, tente entendê-lo. Para auxiliar, pense no seguinte:	O que se deve descobrir ou calcular? Qual é o objetivo? Quais são os dados disponíveis? São suficientes? Faça um esboço informal de como ligar os dados com as condições. Se possível, modele o problema de forma matemática.
Crie um plano com a solução:	Consulte sua memória e verifique se você já resolveu algum problema similar. A sua solução pode ser

	aproveitada por anologia, quando o enunciado for diferente, mas a estrutura em si guarda similaridades; por generalização, quando se tem uma solução particular e deseja uma solução geral; por especialização, quando se conhece alguma solução geral que serve como base para uma em particular ou ainda uma mistura das três técnicas anteriores. Verifique se é necessário introduzir algum elemento novo no problema, como um problema auxiliar. Se o problema for muito complicado, tente quebra-lo em partes menores e solucionar essas partes. É possível enxergar o problema de outra forma, de modo que seu entendimento se torne mais simples?
Formalize a solução:	Crie um algoritmo informal com passos que resolvam o problema. Verifique se cada passo desse algoritmo está correto. Escreva um algoritmo formalizado por meio de um fluxograma ou outra técnica de representação.
Exame dos resultados:	Teste o algoritmo com diversos dados de entrada e verifique os resultados. Se o algoritmo não gerou resultado algum, o problema está na sua sintaxe e nos comandos utilizados. Volte e tente encontrar o erro. Se o algoritmo gerou resultados, estes estão corretos? Analise sua consistência. Se não estão corretos, alguma condição, operação ou ordem das operações está incorreta. Volte e tente encontrar o erro.

Otimização da solução:	É possível melhorar o algoritmo? É possível reduzir o número de passos ou dados? É possível conseguir uma solução ótima?

REVISÃO DO CONCEITO DE ALGORITMO

Em outras palavras, o algoritmo representa o caminho de solução para um problema. A elaboração do algoritmo é de importância crucial para criação de um programa de computador e nas soluções de qualquer tipo de problema. Da definição exposta anteriormente, pode-se extrair as seguintes características evidentes:

- Um algoritmo representa uma sequência de regras;
- Essas regras devem ser executadas em uma ordem preestabelecida;
- Cada algoritmo possui um conjunto finito de regras.
- Essas devem possuir um siguinificado e ser formalizado segundo alguma convenção.

APLICABILIDADE DOS ALGORITMOS

Existe um algoritmo embutido em toda tarefa, independentemente de ela ser relacionada a um programa de computador. Em nosso cotidiano, executamos toda e qualquer tarefa utilizando algoritmos, mesmo não percebendo isso. Atos como comer, respirar, ir para a escola, dirigir um automóvel, resolver uma prova, estudar, cozinhar, fazer uma refeição, consertar o motor de um automóvel etc., são tarefas que podem ser descritas por meio de algoritmos.

Por outro lado, existem algoritmos que precisamos aprender para poder realizar certas tarefas especificas, como, por exemplo, aquelas ligadas à Engenharia e à Computação. Assim, para especificar um processo químico industrial e um programa eficiente de pesquisa de informações em um banco de dados,

entre tantos, são necessários conhecimentos e informações adicionais aos que já possuímos. Desse modo, conclui-se que: Algoritmos não servem apenas para programarcomputadores, eles são de uso geral!

PROPRIEDADES DE UM ALGORITMO

Todo algoritmo possui uma série de propriedade que serão descritas a seguir:

PROPRIEDADE	DESCRIÇÃO
Valores de entrada	Todo algoritmo deve possuir zero, uma ou mais entradas de dados.
Valores de saída	Todo algoritmo possui uma ou mais saídas, que simboliza(m) seu(s) resultado(s).
Finitude	Costuma-se dizer que toda tarefa a ser realizada possui um início, meio e fim. Como os algoritmos representam os passos de solução de um problema – executando assim uma tarefa -, também possuem um início, meio e fim. Portanto, uma primeira propriedade do algoritmo é a finitude. Todo algoritmo deve ser finito, isto é, deve possuir um início e um conjunto de passos que, ao serem executados, levarão sempre ao seu término ou fim, executanto a tarefa a que se propõe. Deve-se uma atenção especial a essa propriedade. Muitas vezes, por desatenção, pode-se criar um algoritmo que nunca chegará a um resultado, tornando-se infinito.
Passos elementares	Um algoritmo computacional deve ser explicitado por meio de operações elementares, sem que possam haver diferenças de interpretação, de forma tal que possa ser executado até por máquinas bastante limitadas.
Correção	Um algoritmo deve ser correto, isto é, deve permitir que, com sua execução, se chegue à(s) saída(s) com resultados

	coerentes com a(s) entrada(s). Para saber se um algoritmo está correto ou não, deve-se realizar testes com diversos valores de entrada (simulação), cujos valores a serem produzidos já se conhece a priori e, então, comprar esses resultados com os valores produzidos pelo algoritmo em questão.

ETEJK > INFORMÁTICA > LTP1

CAPITULO II - ITENS FUNDAMENTAIS

INSTRUÇÕES BÁSICAS

As instruções são representadas pelo conjunto de palavras-chave (vocabulário) de uma determinada linguagem de programação, que tem por finalidade comandar em um computador, o seu funcionamento e a forma como os dados armazenados deverão ser tratados. Deve-se considerar que existem várias linguagens de programação como: Pascal, C, C++, Python, Java, entre outras, sendo que uma determina instrução para se fazer uma tarefa em um computador piderá ser escrita de forma diferente, dependendo da linguagem utilizada. Por exemplo, em português se diz rua, em inglês se diz street e em espanhol se diz calle. São termos diferentes para representar a mesma coisa.

ENTRADA, PROCESSAMENTO E SAÍDA

Para se fazer um programa é necessário ter em mente trê pontos: a entrada, o processamento, e a saída de dados. Se os dados forem encaminhados (entrada) de forma errada, serão consequentemente processados de forma errada e resultarão em respostas erradas. Se houve algum erro, é porque foi causado por falha humana.

Uma entrada e uma saída poderão ocorrer de um computador de diversas formas. Por exemplo, uma entrada poderá ser feita via teclado, modem, leitores óticos, disco, entre outras. Uma saída poderá ser feita em vídeo, impressora, disco, entre outras formas. Devido a esta grande variedade, nossos programas escritos em português estruturado farão menção às instruções leia e escreva.

Português Estruturado

LEIA<lista de dados> (...)
ESCREVA<lista de dados>

REPRESENTAÇÃO

Há várias formas de se representar o problema para o qual iremos desenvolver um programa de computador. Estas formas de representação têm como papel fundamental auxiliar o programador, ou o analista de sistemas, a entender e resolver o problema, para depois buscar a sua solução dentro de um computador. Na verdade, o que vamor fazer é ensinar o computador a resolver o problema por meio de um programa. Assim, o segredo de uma boa lógica de programação está no conhecimento aprofundado do problema a ser solucionado.

O Português Estruturado é uma pseudolinguagem utilizada para desenvolver a lógica de programação simulando uma linguagem de computador. Essa primeira codificação em uma pseudolinguagem, utiliza palavras comuns ao nosso vocabulário como: leia, escreva, início e fim. Assim, fica mais fácil entender a construção da estrutura do programa.

Como exemplo, apresentamos o algoritmo para multiplicar dois números reais.

```
ALGORITMO multiplicação
VARIAVEIS
m: REAL
n: REAL
x: REAL
INICIO
LEIA m
LEIA n
X ← m*n
ESCREVA x
FIM
```

A declaração das variáveis deve ser feita antes da execução das instruções do programa, ou seja, antes do INICIO do programa. Após a declaração das variáveis começa-se a montar a lógica do algoritmo.

Visando melhorar a legibilidade do código de um programa, asinstruções são organizadas de modo a ser visível a qual bloco ou estrutura de controle o comando pertence. Esta técnica recebe o nome de endentação.

COMANDO DE ATRIBUIÇÃO

A partir dos conceitos introduzidos no início, pode-se definir comando como sendo a descrição de uma ação a ser executada em um dado momento. Da presente seção ao final deste capítulo, discute-se como descrever as ações básicas contidas em um algoritmo, ou seja, os comandos e as estruturas disponíveis para o desenvolvimento de algoritmos e o conjunto de regras e convenções adotadas para a representação dos mesmos.

O primeiro dos comandos considerado denomina-se comando de atribuição. Este comando permite que se forneça um valor a uma certa variável, onde a natureza deste valor tem de ser compatível com o tipo da variável na qual está sendo armazenado. O comando de atribuição tem a forma geral apresentada a seguir:

identificador ← expressão

onde:

identificador é o nome da variável à qual está sendo atribuído o valor;

← é o símbolo de atribuição;

A expressão pode ser uma expressão aritmética, expressão lógica ou expressão literal de cuja avaliação é obtido o valor a ser atribuídoà variável.

Exemplo

a) K4 ← 1;
b) COR ← “VERDE”;
c) TESTE 4 ← falso;
d) A4 ← B;

e) MÉDIA 4 ← SOMA/N;
f) COD ← N2 + 1 5;
g) SIM ← X = 0 e Y <> 2

CONSTANTES

Uma constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa. Uma constante pode ser um número (como se conhece na Matemática), um valor lógico ou uma seqüência de caracteres quaisquer com algum significado para o problema em estudo. Conforme o seu tipo, a constante é classificada como sendo numérica, lógica ou literal.

CONSTANTE NUMÉRICA

A representação de uma constante numérica nos algoritmos é feita no sistema decimal, podendo ser um número com ou sem parte fracionária.

Exemplo

a) 25;
b) 3,14.

Na Matemática é comum a existência de constantes com uma parte exponencial, isto é, um fator 10 elevado a um expoente inteiro. Neste caso, é usada a notação já conhecida.

Exemplo

$7,8 \times 10^3$

A constante numérica pode ser positiva ou negativa, de acordo com o sinal que precede os algarismos formadores do número. Caso não exista um sinal, a constante é considerada positiva. Além disso, também o expoente da parte exponencial possui um sinal indicando deslocamento da vírgula para a direita ou para a esquerda, conforme o sinal seja positivo ou negativo, respectivamente.

A seguir, são apresentados alguns exemplos de constantes numéricas.

Exemplo

a) -15;
b) +15;
c) 0,314;
d) —2,5 x 10^3;
e) 10;
f) 2,5 x 10^3.
g) 1000000000;

CONSTANTE LÓGICA

É um valor lógico, isto é, que só pode ser falso ou verdadeiro, usado em proposições lógicas, conforme será visto mais adiante. Só existem duas constantes deste tipo, sendo representadas pelas palavras falso e verdadeiro.

CONSTANTE LITERAL

Uma constante deste tipo pode ser qualquer seqüência de caracteres (letras, dígitos ou símbolos especiais) que forme um literal com algum significado para o problema em estudo. Toda constante literal que aparece no algoritmo será colocada entre aspas para que não seja confundida com outro item qualquer.

Exemplo

a) “JOSE DA SILVA”;
b) “xpto”;
c) “MENSAGEM”;
d) “12345”;
e) “23/09/55”.

Note que um numeral entre aspas é considerado como urna seqüência de dígitos (literal), e não como urna constante numérica. Também não se deve confundir uma constante lógica (por exemplo, falso) com urna

seqüência de caracteres (literal), que apareça entre aspas (por exemplo, “FALSO”).

VARIÁVEL

O termo variável tem como definição aquilo que é sujeito aIa,q é incerto, instável ou inconstante. Em programação a quantidade de informações que são processadas é muito grande e diversificada, com isso, os dados se tornam bastante variáveis.

A memória de um computador pode ser comparada com um grande arquivo com várias gavetas, que seriam os locais físicos responsáveis por armazenar os dados. Cada gaveta pode armazenar um único valor de um tipo específico por vez. Considerando que a memória de um computador seria um arquivo muito grande, é necessário identificar cada gaveta com uma etiqueta para facilitar a localização da informação.

As variáveis são como as gavetas do arquivo. Elas correspondem a uma posição de memória onde será armazenada uma informação que pode variar no decorrer da execução do algoritmo. Apesar de urna variável poder assumir diferentes valores, ela só pode armazenar um valor de cada vez.

Toda variável é identificada por um nome ou identificador para sua localização e posterior uso dentro de um programa. Veja a seguir as regras para a utilização de nomes para variáveis:

- O nome pode ter um ou mais caracteres;
- Primeiro caractere do nome de urna variável deve ser sempre urna letra;
- Pode utilizar apenas letras e números;
- Não pode conter caracteres especiais;
- Não pode possuir espaços em branco;
- Não pode ser palavra reservada, ou seja, que faça parte da linguagem de programação utilizada.

São exemplos de nomes válidos: NOME, NUMERO1, A, N2 entre outros. São nomes inválidos para variáveis:

NOME USUARIO, 1NUM, EMAIL@, VAR (palavra reservada), entre outras.

Dentro de um programa a variável pode exercer dois papéis: um de ação, quando é modificada ao longo de um programa para apresentar um determinado resultado, e outro de controle, onde esta poderá ser controlada durante a execução de um programa.

COMENTÁRIOS

A esta altura o leitor já percebeu a preocupação existente com a clareza do algoritmo, ou seja, o grau de facilidade que as pessoas terão em compreender o que nele está descrito.

Um instrumento de grande valia usado para esta finalidade denomina-se comentário. Ele é um texto, ou simplesmente uma frase, que aparece sempre delimitado por chaves ({comentário}). Os comentários podem ser colocados em qualquer ponto do algorilmo onde se façam necessários.

No exemplo a seguir é mostrado um conjunto de declarações onde foram introduzidos comentários com o intuito de explicar o significado de cada uma das variáveis.

Exemplo

declare
MAT, {número de matrícula de aluno}
NOTA, {total de pontos obtidos no semestre letivo}
COD {código do curso}
numérico

declare
NOME, {nome completo do aluno}
END, {endereço do aluno}
C {conceito final}
literal

Importante: Todo algoritmo deve conter comentários, a

fim de que as pessoas possam entendê-los mais facilmente.

OPERADORES

Os operadores são meios pelo qual se incrementa, decrementa, compara e avalia os dados dentro do computador. Tendo as variáveis como base da informação de uma linguagem, elas podem ser modificadas e comparadas com outras por meio dos chamados operadores. Temos três tipos de operadores:

- Operadores Aritméticos;
- Operadores Relacionais;
- Operadores Lógicos.

OPERADORES ARITMÉTICOS

Os operadores aritméticos são utilizados para realizar operações numéricas com os dados utilizados pelo programa. Além da adição, subtração, multiplicação e divisão, pode-se utilizar também o operador para exponenciação e radiciação.

Há dois operadores não muito convencionais, porém bastante úteis para resolver diversas situações na construção de um algoritmo: o mod e o div. O mod retorna o resto, enquanto que o div retorna o quociente da divisão inteira.

Os simbolos para os operadores aritméticos são os seguintes:

Operação	Símbolo
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Exponenciação	^ ou **
Radiciação	rad
Resto da Divisão	mod
Quociente da Divisão	div

Os símbolos mostrados na tabela acima são os únicos simbolos que podem ser usados para representar as operações aritméticas em um algoritmo. As expressões aritméticas devem ser escritas no formato linear para facilitar a digitação dos programas e também porque alguns simbolos usados em matemática não existem nos teclados. O exemplo mais comum deste formato é a operação de divisão que deve ser escrita na forma a/b.

Para resolver as operações aritméticas há uma hierarquia a ser seguida:

Prioridade	Operadores
1.º	( )
2.º	** rad
3.º	* / div mod
4.º	+ -

Nota: Quando duas operações de mesmo nível de prioridade têm de ser avaliadas, a operação mais à esquerda será resolvida primeiro.

Os parênteses têm um papel importante nas expressões e permitem que a ordem das operações seja alterada. Expressões entre pares de parênteses são calculadas em primeiro lugar, portanto eles conferem o maior grau de prioridade as expressões que eles envolvem. Podemos ter pares de parênteses envolvendo outros pares. Dizemos que os parênteses estão aninhados. Neste caso, as expressões dentro dos parênteses mais internos são avaliadas primeiro.

Um ponto importante que deve ser sempre levado em consideração, quando uma expressão for calculada, são os tipos das variáveis, porque eles alteram radicalmente os resultados das expressões. Se uma variável foi declarada como inteiro, a divisão entre inteiros trunca qualquer parte decimal que ocorra. Veja a seguir alguns exemplos de como resolver as operações matemáticas seguindo a ordem de prioridade:

OPERADORES RELACIONAIS

Durante o desenvolvimento de um algoritmo, vamos sempre encontrar situações em que será necessário comparar informações para que o programa possa tomar uma decisão. Para isso, é necessário testar uma condição, que em geral consiste em verificar se uma determinada variável tem valor verdadeiro ou falso. Essa comparação é por exemplo: "se A é maior que B", ou se "A é igual 15' se "A é diferente de 50".

Os operadores relacionais são utilizados para comparar duas expressões de qualquer tipo. Quando efetuamos uma comparação o resultado é sempre do tipo lógico (booleano), isto é resulta sempre em um valor Verdadeiro (True) ou Falso (False). Para comparar elementos utilizamos os operadores relacionais que se encontram discriminados na tabela seguinte:

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

Tendo duas variáveis A = 7 e B = 4, os resultados das expressões seriam:

A = B	Falso
A <> B	Verdadeiro
A > B	Verdadeiro
A < B	Falso
A >= B	Verdadeiro
A <= B	Falso

Lembre-se: Para estabelecer prioridades no que diz respeito a qual operação executar primeiro, utilize os parênteses. As operações lógicas só podem ser efetuadas com relação a valores do mesmo tipo.

OPERADORES LÓGICOS

Quando há a necessidade de trabalhar com duas ou mais condições ao mesmo tempo são utilizados os operadores lógicos que são responsáveis para a ftrmação de novas proposições lógicas compostas a partir de outras proposições lógicas simples. Os operadores lógicos são utilizados para comparar mais de uma condição em urna mesma expressão, ou seja pode-se fazer mais de uma comparação (teste) ao mesmo tempo, retornando se o resultado da nova proposição é verdadeiro ou falso.

Os operadores lógicos são:

Operador	Operação
não / not	Negação
e / and	Conjunção
ou / or	Disjunção

Operações lógicas são utilizadas quando é necessário tomar uma decisão. Como no exemplo do algoritmo "CHUPAR UMA BALA": Imagine que algumas pessoas não gostem de chupar bala de caramelo, neste caso teremos que modificar o algoritmo para:

- Pegar a bala
- A bala é de caramelo?
- Se sim, não chupe a bala.
- Se não, continue com o algoritmo.
- Retirar o papel.
- Chupar a bala.
- Jogar o papel no lixo.

A prioridade entre os operadores lógico são:

Prioridade	Operador
1ª	não / not
2ª	e / and
3ª	ou / or

OPERADOR E/AND

O operador lógico E é utilizado quando todas as proposições lógicas de uma condição necessitam ser verdadeiras. Veja a seguir a tabela verdade para esse tipo de operador:

1ª Condição	Operador	2ª Condição	Resultado
V	E	V	V
V	E	F	F
F	E	V	F
F	E	F	F

Observe como esse operador é utilizado.

Em portugol:

```
PROGRAMA Exemplo_E
VARIÁVEL
  numero: INTEIRO
INÍCIO
  LEIA numero
  SE (numero >= 1) E (numero <= 10) ENTÃO
    ESCREVA "O número está entre 1 e 10"
  SENÃO
    ESCREVA "O número não está entre 1 e 10"
  FIM SE
FIM
```

No exemplo, a ação de exibir na tela "O número está entre 1 e 10" só õcorre se o número digitado pelo usuário for maior ou igual a 1 e menor ou igual a 10. Caso contrário é exibido na tela "O número não está entre 1 e 10". O operador E só executa uma determinada operação se todas as condições forem verdadeiras.

OPERADOR OU/OR

O operador lógico OU é utilizado quando pelo menos uma das proposições lógicas de uma condição necessitam ser verdadeiras. Veja a seguir a tabela verdade para esse tipo de operador:

1ª Condição	Operador	2ª Condição	Resultado

V	OU	v	v
V	OU	F	V
F	OU	V	V
F	OU	F	F

O operador OU só executa uma determinada operação se pelo menos uma das condições for verdadeira.

Veja como esse operador é utilizado.

Em portugol:

```
PROGRAMA Exemplo_OU
VARIÁVEL
  periodo: CARACTER
INÍCIO
  LEIA período
  SE (periodo = "manhã") OU (periodo = "tarde") ENTÃO
    ESCREVA "Período válido!"
  SENÃO
    ESCREVA "Período inválido!"
  FIM SE
FIM
```

Como podemos ver no exemplo anterior, só será exibido na tela "Período válido!" se o período digitado pelo usuário for manhã ou tarde. Se o usuário digitar um período que não seja um desses dois, aparecerá na tela a mensagem "Período inválido!"

OPERAÇÃO NÃO/NOT

O operador lógico NÃO é utilizado quando há a necessidade de inverter o valor lógico de uma determinada condição. Se a condição for verdadeira assumirá o valor falso, e se a condição for falsa assumirá o valor verdadeiro. Veja a seguir a tabela verdade para esse tipo de operador:

1ª Condição	Operador	Resultado
V	NÃO	F
F	NÃO	V

O operador NÃO executa uma determinada operação se a condição não for verdadeira, ou seja, se for falsa ou vice-versa.

Observe como esse operador pode ser utilizado.

Em portugol:

```
PROGRAMA Exemplo_NAO
VARIÁVEL
  M, N, P, resultado: INTEIRO
INÍCIO
  LEIA M, N, P
  SE NÃO (P < 10) ENTÃO
    resultado ← P * (M – N)
  SENÃO
    resultado ← P * (N — M)
  FIM SE
FIM
```

No exemplo anterior o cálculo resultado ← P * (M - N) só é executado se P for maior que 10. Para qualquer valor abaixo de 10, será efetuado o cálculo resultado ← P * (N - M).

OPERAÇÕES RELACIONAIS

Operador	Significado	Resultado
=	Igualdade	Booleano (true ou false). Sendo true, se o 1° operando for igual ao 2° e false, se diferentes.
<	Menor	Booleano (true ou false). Sendo true, se o 1° operando for nenor que o 2° e false, caso contrário.
>	Maior	Booleano (true ou false). Sendo true, se o 1° operando for maior que o 2° e false, caso contrário.
<=	Menor ou igual	Booleano (true ou false). Sendo true, se o 1° for menor ou igual ao 2° e false, caso

		contrário.
>=	Maior ou igual	Booleano (true ou false). Sendo true, se o 1° operando for maior ou igual ao 2° e false, caso contrário.
<>	Diferente	Booleano (true ou false). Sendo true, se o 1° operando for diferente do 2° e false, caso sejam iguais.

OPERAÇÕES LÓGICAS

Operador lógico	1° operando (A)	2° operando (B)	Resultado (C)	Simbologia
NOT	True		False	C ← not A
	False		True	
AND	True	True	True	C ← A and B
	True	False	False	
	False	True	False	
	False	False	False	
OR	True	True	True	C ← A or B
	True	False	True	
	False	True	True	
	False	False	False	

EXPRESSÕES

Se prestarmos atenção, todas as Operações discutidas são levadas a efeito Com um ou dois operandos de cada vez. Uma expressão conteiido diversos operandos deve ser avaliada de acordo com a precedência dos operadores envolvidos. A precedência dos operadores indica, em uma expressão, qual operação será realizada antes das outras.

Os operadores aritméticos possuem a seguinte precedência, do maior para o menor:

Troca de sinal (“-“ > unário);
*, /, mod, div (na ordem em que aparecem);
3 + e -.

De forma diferente da Matemática, na qual as expressões podem ser agrupadas cot chaves, colchetes e parênteses, aqui o iinico elemento de agrupamento de expressõt válido serão os parênteses. Assim, em expressões contendo parênteses, a precedência ligeiramente alterada:
1. Resolver o nível mais interno de parênteses, usando a precedência de operações...

(a) Troca de sinal (“—“ unário);
(b) *, /, mod, div (na ordem em que aparecerem);
(e) + e -.

2. Passar para o próximo nível, sempre do mais interno para o mais externo, utilIzando a precedência de operações.

No caso de operadores lógicos, a precedência é (do maior para o menor): NOT, AND e por fim OR. Esta última também pode ser alterada, da mesma forma, pelo uso de parênteses. Finalmente, em um último nível da hierarquia dos operadores, têm-se os operadores relacionais.

PRECEDÊNCIA DOS OPERADORES

Hierarquia	Operadores
1	not
2	*, /, div, mod, and
3	+, —, or
4	>, >=, <, <=, =, <>

ESTRUTURAS DE CONTROLE

Toda linguagem de programação possui instruções que controlam o fluxo de execução de um programa. As estruturas de controle são inseridas em um código-fonte com o objetivo de direcionar o fluxo de execução, fazendo com que algumas linhas de código só

sejam executadas caso seja necessário. Estas estruturas permitem que um código seja executado um número determinado de vezes ou quando obedecerem a condições lógicas.

Para desenvolver programas eficientes é necessário, na maioria das vezes, tomar decisões durante a execução do algoritmo, como por exemplo, decidir se será exibido para o usuário se o aluno foi aprovado ou reprovado, de acordo com os dados informados. Essas decisões interferem diretamente no andamento do programa, e é através delas que resolvemos os problemas propostos.

ESTRUTURAS DE DECISÃO

Os comandos de decisão, ou desvio, são técnicas de programação que conduzem as estruturas de programas que não são totalmente seqüenciais. As estruturas de decisão determinam se um bloco de ações será ou não executado de acordo com as decisões tomadas por condições lógicas ou relacionais. As principais estruturas de decisão são:

- Se... Então;
- Se... Então... Senão;
- Selecione... Caso.

SE... ENTÃO

A estrutura de decisão SE ENTÃO é utilizada quando há a necessidade de testar uma condição antes de executar uma ação.

A estrutura SE ENTÃO é a seguinte...

Portugol:

SE <condição> ENTÃO
<comandos>
FIM SE

O código é analisado da seguinte forma: se a condição for verdadeira, então os comandos que estão entre o "SE... ENTÃO" e o "FIM SE" serão executados. Caso a condição seja falsa, serão executados os comandos que estão após o "FIM SE".

Exemplo:

Imagine um algoritmo que deve verificar se um número é positivo e exibir na tela "Número positivo" Veja como ficaria o algoritmo em portugol:

```
PROGRAMA Exemplo_SE
VARIÁVEL
  num: INTEIRO
INÍCIO
  LEIA num
  SE (num > 0) ENTÃO
    ESCREVA "O número é positivo"
  FIM SE
FIM
```

SE... ENTÃO... SENÃO

A estrutura de decisão "SE... ENTÃO... SENÃO" é utilizada quando há duas alternativas que dependem de uma mesma condição, onde uma será executada caso a condição seja verdadeira e a outra caso a condição seja falsa. A estrutura "SE... ENTÃO... SENÃO" é a seguinte...

Portugol:

```
SE <condição> ENTÃO
  <comandos 1>
SENÃO
  <comandos 2>
FIM_SE
```

Essa estrutura é executada da seguinte forma: se a condição for verdadeira então os comandos 1 que estão entre o SE ENTÃO e o SENÃO serão executados, caso a condição seja falsa serão executados os comandos 2 que estão entre o SENÃO e o FIM SE.

Exemplo:

Como exemplo vamos mostrar um algoritmo de um programa que exiba na tela se o número digitado pelo usuário é par ou ímpar.

Portugol:

```
PROGRAMA Exemplo_SE_SENAO
VARIÁVEL
num, par: INTEIRO
INÍCIO
  LEIA num
  par ← num mod 2
  SE (par = 0) ENTÃO
    ESCREVA "O número é par"
  SENÃO
    ESCREVA "O número é ímpar"
  FIM SE
FIM
```

SELECIONE... CASO

A estrutura de decisão "SELECIONE CASO" é utilizada para testar, na condiçao, uma unica expressão ou variável que produz um resultado diferente para cada valor que assumir. Compara-se, então, o resultado obtido no teste com os valores fornecidos em cada cláusula CASO e executa somente os comandos do CASO cujo valor é igual ao declarado no SELECIONE. A estrutura do "SELECIONE CASO" é a seguinte...

Portugol:

```
SELECIONE <condição>
  CASO V1: <comandos>
  CASO V2: <comandos>
  CASO V3: <comandos>
FIM SELECIONE
```

Observe como essa estrutura é executada de ãcprdo com os dados fornecidos pelo usuário para uma variável ou expressão. Em cada condição da estrutura há um possível valor que é comparado ao valor da variável ou expressão que foi informado pelo usuário, se os

valores forem iguais então é executado os comandos do CASO correspondente saindo dessa estrutura de controle. Se nenhuma das condições forem verdadeiras, ou é executado um comando predeterminado para essa situação, ou a estrutura é encerrada. Nessa estrutura de decisão somente um dos casos é executado.

Exemplo:

Para exemplificar vamos montar um algoritmo que leia dois números e execute com eles uma das quatros operações básicas da matemática: adição, subtração, multiplicação e divisão.

Portugol:

```
PROGRAMA Exenplo_CASO
VARIÁVEIS
  nunl, num 2, total: INTEIRO
  operador: CARACTER
INÍCIO
  LEIA nunl e num2
  LEIA operador
  SELECIONE operador
    CASO "+"
      total ← num1 + num2
      ESCREVA (total)
    CASO "-"
      total ← numl — num2
      ESCREVA (total)
    CASO "*"
      total ← numi * nun2
      ESCREVA (total)
    CASO "/"
      total ← numl / nun2
      ESCREVA (total)
    CASO CONTRÁRIO:
      ESCREVA "Opção errada!"
  FIM SELECIONE
FIM
```

O comando CASO SELECIONE executa o mesmo papel do comando SE ENTÃO, mas de uma forma mais compacta nas operações de seleção Veja como ficaria esse mesmo exemplo com o comando SE ENTÃO:

```
PROGRAMA Exemplo SE ENTAO
```

```
VARIÁVEIS
  num1, num 2, total: INTEIRO
  operador: CARACTER
INÍCIO
  LEIA num1 e num2
  LEIA operador
  SE (operador = "+") ENTÃO
    total <— num1 + num2
    ESCREVA (total)
  SENÃO
    SE (operador = "-") ENTÃO
      total ← num1 - num2
      ESCREVA (total)
    SENÃO
      SE (operador = "*") ENTÃO
      total <— num1 * num2
      ESCREVA (total)
      SENÃO
        SE (operador = "/") ENTÃO
          total <— num1 / num2
          ESCREVA (total)
        SENÃO
          ESCREVA "Opção errada!"
        FIM SE
      FIM SE
    FIM SE
  FIM SE
FIM
```

Estruturas de Decisão Encadeadas

Dependendo do contexto do programa é necessário verificar condições sucessivas, colocando uma condição dentro de outra condição. As estruturas que possuem vários níveis de condição são chamadas de aninhamentos ou encadeamentos.
Nesse tipo de estrutura uma ação só é realizada se um conjunto de comandos ou condições for executado. Assim a realização de uma ação depende do resultado de outras condições ou instruções.

```
SE <condição> ENTÃO
<comandos 1>
SENÃO
SE <condição 2> ENTÃO
<comandos 2>
SENÃO
<comandos 3>
```

FIM SE
FIMSE

Veja que os <comandos 1> só serão executados se a <condição 1> for verdadeira, caso ela seja falsa será verificado a <condição 2>. Nesse momento a estrutura entra em um outro nível de condição, onde se a <condição 2> for verdadeira os <comandos 2> serão executados, porém se o seu resultado for falso a execução será dos <comandos 3>.

Exemplo:

Para exemplificar essa estrutura vamos elaborar um programa que lê o valor dos três lados de um triângulo e verifica se ele é eqüilátero (os 3 lados iguais), isósceles (2 lados iguais) ou escaleno (os 3 lados diferentes).

```
PROGRAMA Exemplo_Desvios_Condicionais_Encadeados
VARIÁVEIS
  A, B, C: INTEIRO
INÍCIO
  LEIA A, B, C
  SE (A<B+C) E (B<A+C) E (C<A+B) ENTÃO
    SE (A=B) E (B=C) ENTÃO
      ESCREVA "Triângulo Equilátero"
    SENÃO
      SE (A=B) OU (A=C) OU (C=B) ENTÃO
        ESCREVA "Triângulo Isósceles"
      SENÃO
        ESCREVA "Triângulo Escaleno"
      FIM SE
    FIM SE
  SENÃO
    ESCREVA "Estas medidas não formam um triângulo"
  FIM SE
FIM
```

Obervações importantes: Num dado triângulo, qualquer que seja o seu tipo, a medida ou comprimento de um dos seus lados SEMPRE será menor do que a soma da medida ou comprimento dos outros dois lados. No caso do triângulo equilátero, todos os seus três lados

possuem a mesma medida ou comprimento; no isóscele, dois lados possuem a mesma medida ou comprimento e o terceiro lado é diferente; já no triângulo escaleno seus três lados possuem medidas ou comprimentos diferentes.

ESTRUTURA DE REPETIÇÃO

Durante o desenvolvimento de um programa muitas vezes encontramos situações onde é necessário repetir um trecho de código várias vezes, como, por exemplo, quando temos que calcular a média aritmética das notas dos alunos de uma escola. Se fossemos utilizar a estrutura SE ENTÃO teríamos que escrever um trecho de cálculo para cada aluno da escola. Para resolver situações como esta, utilizamos os comandos de repetição, que possibilitam que um trecho de código seja repetido por quantas vezes forem necessárias.

Os comandos de repetição também são conhecidos por: loops ou looping, que significa voltas, laçadas. São utilizados quando desejamos que um determinado conjunto de instruções ou comandos sejam executados um número definido ou indefinido de vezes, ou enquanto um determinado estado de coisas prevalecer ou até que seja alcançado.

Veremos agora os seguintes comandos de repetição:

- ENQUANTO... FAÇA
- REPITA... ATË QUE
- PARA... DE... ATË... PASSO... FAÇA

A principal vantagem desses comandos é que eles possibilitam a redução das linhas de códigos e a melhora no desempenho do processamento.

ENQUANTO... FAÇA

A estrutura de repetição “ENQUANTO... FAÇA” permite que um trecho de código possa ser executado diversas vezes dependendo do resultado de um teste lógico. Essa estrutura possui um teste lógico no início da

repetição, então as instruções apenas serão executadas se a condição for verdadeira, caso contrário é executado os comandos posteriores a essa estrutura. A estrutura do comando “ENQUANTO... FAÇA” é a seguinte...

Portugol:

ENQUANTO <condição> FAÇA
<comandos>
FIM ENQUANTO

Nota: Perceba que nessa estrutura existe um retorno à condição após a execução dos comandos, até que a condição seja falsa.

Neste caso, antes de executar os comandos é feita a verificação da condição, se a condição for verdadeira, então os comandos são executados e o fluxo de execução retorna novamente para verificar a condição, até que ela seja falsa. Quando a resposta da condição for falsa, o fluxo de execução sai da estrutura de repetição e executa as instruções que estão depois do “FIM ENQUANTO”.

O teste da condição será sempre realizado antes de qualquer operação e enquanto a condição for verdadeira o processo se repete. No entanto, se na primeira consulta a condição já retornar como resultado o valor falso, o comando de repetição é abandonado e os comandos não são executados nenhuma vez. Essa é a sua principal característica.

Exemplo:

Como eempIo vamos construir uni algoritmo para o cálculo do fatorial do número 3, 3!. Lembre-se que o fatorial de 3! = 3 * 2 * 1 ou 3! = 1 * 2 * 3 é igual a 6.

```
PROGRAMA Exemplo_Enquanto
VARIÁVEIS
  contador, fatorial: INTEIRO
INICÍCIO
```

```
  fatorial <— 1
  contador <— 1
  ENQUANTO (contador <= 3) FAÇA
    fatorial <— fatorial * contador
    contador <— contador + 1
  FIM ENQUANTO
  ESCREVA "O fatorial de 3 é = ", fatorial
FIM
```

ENQUANTO... FAÇA

A estrutura de repetição "ENQUANTO... FAÇA" permite que um trecho de código possa ser executado diversas vezes dependendo do resultado de um teste lógico. Essa estrutura possui um teste lógico no início da repetição, então as instruções apenas serão executadas se a condição for verdadeira, caso contrário é executado os comandos posteriores a essa estrutura. A estrutura do comando "ENQUANTO... FAÇA" é a seguinte...

Portugol:

ENQUANTO <condição> FAÇA
<comandos>
FIM ENQUANTO

Nota: Perceba que nessa estrutura existe um retorno à condição após a execução dos comandos, até que a condição seja falsa.

Neste caso, antes de executar os comandos é feita a verificação da condição, se a condição for verdadeira, então os comandos são executados e o fluxo de execução retorna novamente para verificar a condição, até que ela seja falsa. Quando a resposta da condição for falsa, o fluxo de execução sai da estrutura de repetição e executa as instruções que estão depois do "FIM ENQUANTO".

O teste da condição será sempre realizado antes de qualquer operação e enquanto a condição for verdadeira o processo se repete. No entanto, se na primeira consulta a condição já retornar como resultado o valor falso, o comando de repetição é

abandonado e os comandos não são executados nenhuma vez. Essa é a sua principal característica.

Exemplo:

Como exemplo, vamos construir um algoritmo para o cálculo do fatorial do número 3, representado por "3!". Lembre-se que o fatorial de 3! = 3 * 2 * 1 ou 3! = 1 * 2 * 3 é iguala 6.

```
PROGRAMA Exemplo_Enquanto
VARIÁVEIS
  contador, fatorial: INTEIRO
INÍCIO
  fatorial <— 1
  contador <— 1
  ENQUANTO (contador < 3) FAÇA
    fatorial <— fatorial * contador
    contador <— contador + 1
  FIM ENQUANTO
  ESCREVA "O fatorial de 3 é = ", fatorial
FIM
```

Como podemos ver é a variável CONTADOR que controla a quantidade de vezes que as instruções do comando ENQUANTO serão repetidas. Esse recurso é muito utilizado nas estruturas de controle, pois a cada looping a variável que armazena as contagens é incrementada em 1. Assim pode-se determinar quantas vezes os comandos foram executados.

Nota: Incrementar é somar um valor constante, normalmente o 1, a um valor já existente.

A estrutura básica de um contador é a seguinte:

Declaração do contador: contador: INTEIRO
Inicialização do contador: contador ← 0
Incrementar o contador em 1: contador ← contador + i
Já a variável FATORIAL é um contador que funciona como acumulador, pois o valor adicionado pode variar não mantendo uma seqüência que possa ser utilizada para controlar a quantidade de voltas da repetição, ao contrário da contagem, que é incrementado de 1 em

1.

Porém o contador não é a única maneira de controlar as voltas do comando de repetição ENQUANTO FAÇA. A condição de repetição pode ser uma decisão do usuário. Veja a seguir um exemplo com esse tipo de decisão:

```
PROGRAMA Exemplo Enquanto#2
VARIÁVEIS
  dias, salario: INTEIRO
  resposta: CARACTER
INÍCIO
  resposta <— "sim"
  ENQUANTO (resposta = "sim") FAÇA
    LEIA dias
    salario <— dias * 10
    ESCREVA salario
    ESCREVA "Deseja continuar?"
    LEIA resposta
  FIM ENQUANTO
FIM
```

REPITA... ATÉ QUE

Nesta estrutura de repetição, o teste lógico é feito no final do bloco de instruções, ao contrário do "ENQUANTO... FAÇA" que possui o teste no início. Devido a isso os comandos dessa estrutura são executados pelo menos uma vez, já que o teste é realizado após a execução dos comandos. Veja a estrutura do comando "REPITA... ATË QUE"...

Portugol:

REPITA
<comandos>
ATÉ QUE <condição>

Quando o código é executado pela primeira vez, o fluxo de execução das instruções é mantido na mesma seqüência até a verificação da validade da condição. Se a condição for verdadeira, o fluxo do programa continua normalmente, mas se a resposta da condição for falsa, os comandos que estão entre o "REPITA" e o

“ATÉ QUE” são processados novamente.

Observe que o funcionamento do REPITA é o contrário do ENQUANTO, já que sempre irá processar o bloco de instruções pelo menos uma vez até que a condição seja verdadeira, ou seja, ela mantém a repetição dos comandos enquanto a condição for falsa, abortando quando for ela verdadeira. Já a estrutura ENQUANTO só executa seu bloco de instruções se a condição for verdadeira e aborta quando for falsa.

Exemplo:

Para analisar melhor esta estrutura de repetição, vamos desenvolver um algoritmo que exibe os números inteiros de 1 a 100.

Portugol:

```
PROGRAMA Exemplo Repita#1
VARIÁVEIS
  num: INTEIRO
INÍCIO
  num <— 1
  REPITA
    ESCREVA num
    num <— num + 1
  ATÉ QUE (num > 100)
FIM
```

A condição de repetição da estrutura REPITA também pode ser determinada por uma resposta do usuário. Veja a seguir um exemplo dessa situação onde o programa lê e calcula a média aritmética de dois números e pergunta para o usuário se ele deseja continuar.

Portugol:

```
PROGRAMA Exemplo_Repita#2
VARIÁVEIS
  media, num1, num2: INTEIRO
  resposta: CARACTER
INÍCIO
  REPITA
    ESCREVA “Digite os números:”
```

```
    LEIA numl e num2
    media <— (numi + num2) / 2
    ESCREVA media
    ESCREVA "Deseja continuar (sim/não)?"
    LEIA resposta
  ATÉ QUE (resposta "não")
FIM
```

(...)

PARA... DE... ATÉ... PASSO... FAÇA

Até agora vimos duas estruturas de repetição, ENQUANTO e REPITA, onde um bloco de instruções é executado enquanto ou até que uma condição seja satisfeita. Essas duas estruturas controlam a quantidade de vezes que um grupo de operações é executado utilizando uma variável de controle como contador. Nesse caso, o número de repetições já é determinado, porém a quantidade de voltas dentro da estrutura pode ser indeterminado, tendo como condição uma resposta do usuário.

Já a estrutura PARA é utilizada somente quando já se sabe a quantidade de vezes que a execução de um bloco de instruções deve ser repetida, pois não verifica uma condição, mas sim uma variável denominada contador. Assim, quando é conhecido o número de vezes que uma determinada seqüência de instruções deverá ser executada utiliza-se a estrutura "PARA... DE... ATÉ... PASSO... FAÇA", deixando para as estruturas ENQUANTO e REPITA as situações onde o número de repetições não é conhecido. A estrutura do comando "PARA... DE... ATÉ... PASSO... FAÇA" é a seguinte:

```
PARA <variável> DE <valor inicial> ATÉ <valor final>
PASSO <incremnto> FAÇA
<comandos>
FIM PARA
```

Na estrutura de repetição PARA a variável de controle recebe o valor inicial e verifica se o valor de variável de controle ultrapassa o valor final. Se não ultrapassa, o bloco de comandos que está dentro da estrutura PARA será executado. A seguir, a variável

de controle será incrementada com o valor determinado e verifica-se novamente se o seu valor ultrapassa o valor final. Se não ultrapassar, o bloco de comandos será executado, caso contrário o fluxo de execução do programa vai para a próxima linha depois do FIM PARÁ. Na representação em forma de fluxograma a estrutura do PARA é indicada pela figura de um hexágono denominado Preparação ou Processamento predefinido.

Dentro da figura é indicada a variável a ser controlada com a implicação dos valores iniciais, fInais e de incrementos, separados por vírgula.
Um aspecto importante à ser lembrado sobre essa estrutura de repetição é que ela sempre será executada pelo menos uma vez, pois a variável de controle receberá o valor inicial como primeiro valor, estando então dentro da faixa de valores que possibilitam a execução dos comandos. Outro aspecto à ser destacado é o fato de nem sempre a variável de controle atingir o valor final estabelecido, como por exemplo quando o valor do incremento é maior que 1. Nesse caso, a repetição termina quando a variável de controle assume um valor maior que o valor final.

Exemplo:

Para ilustrar vamos desenvolver um algoritmo de um programa que calcula e exibe a tabuada do número 9.

```
PROGRAIO Exemplo_PARA
VARIÁVEIS
  contador, resultado: INTEIRO
INÍCIO
  PARA contador DE 1 ATÉ 10 PASSO 1 FAÇA
    resultado <— contador * 9
    ESCREVA resultado
  FIM PARA
FIM
```

ESTRUTURAS DE REPETIÇÃO ENCADEADAS

Assim como as estruturas de decisão, as estruturas de repetição também podem ser encadeadas de acordo com o problema a ser resolvido, O encadeamento pode ser entre estruturas do mesmo tipo e também com

estruturas de tipos diferentes.

Dica: entenda bem as estruturas de decisão e repetição para que você possa saber utilizar a estrutura mais conveniente para a resolução de um problema.

Para facilitar o entendimento das combinações das estruturas de repetição, vamos explicar cada uma utilizando a representação em portugol e em fluxograma.

ENQUANTO com ENQUANTO

ENQUANTO <condição 1> FAÇA
ENQUANTO <condição 2> FAÇA
<comandos>
FIM ENQUANTO
FIM ENQUANTO

Exemplo:

Programa que calcula a tabuada de qualquer número quantas vezes o usuário desejar, utilizando a estrutura ENQUANTO com ENQUANTO.

```
PROGRAMA Exemplo_Enquanto_com_Enquanto
VARIÁVEIS
  contador, num: INTEIRO
  resposta: CARACTER
INÍCIO
  resposta <— sim
  ENQUANTO (resposta = sim) FAÇA
    LEIA num
    contador <— 1
    ENQUANTO (contador <= 10) FAÇA
      ESCREVA contador, "X", num, "=", contador * num
      contador <- contador + 1
    FIM ENQUANTO
    ESCREVA "Deseja continuar? (sim/nao)"
    LEIA resposta
  FIM ENQUANTO
FIM
```

ENQUANTO com REPITA

ENQUANTO <condição 1> FAÇA
REPITA
<instruções>
ATÉ QUE <condição 2>
FIM ENQUANTO

Exemplo:
Programa que calcula a tabuada de qualquer número quantas vezes o usuário desejar, utilizando a estrutura ENQUANTO com REPITA.

```
PROGRAMA Exemplo_Enquanto_com_Repita
VARIÁVEIS
  contador, num: INTEIRO
  resposta: CARACTER
INÍCIO
  resposta <— sim
  ENQUANTO (resposta = sim) FAÇA
    LEIA num
    contador <— 1
    REPITA
      ESCREVA contador, "X", num, "=", contador * num
      contador <- contador + 1
    ATÉ QUE (contador > 10)
    ESCREVA "Deseja continuar? (sim/nao)"
    LEIA resposta
  FIM ENQUANTO
FIM
```

ENQUANTO com PARA

Portugol:

ENQUANTO <condição> FAÇA
PARA <variável> DE <valor inicial> ATÉ <valor final>
PASSO FAÇA
<comandos>
FIM PARA
FIM ENQUANTO

Exemplo:

Programa que calcula a tabuada de qualquer número quantas vezes o usuário desejar, utilizando a estrutura ENQUANTO com PARA.

```
PROGRAMA Exemplo_Enquanto_com_Para
VARIÁVEIS
  contador, num: INTEIRO
  resposta: CARACTER
INÍCIO
  resposta <- sim
  ENQUANTO (resposta sim) FAÇA
    LEIA num
    PARA contador DE 1 ATÉ 10 PASSO 1 FAÇA
      ESCREVA contador, "X", num, "=", contador * num
    FIM PARA
    ESCREVA "Deseja continuar? (sim/nao)"
    LEIA resposta
  FIM ENQUANTO
FIM
```

REPITA com REPITA

Portugol:

REPITA
REPITA
<instruções>
ATÉ QUE <condição 2>
ATÉ QUE <condição 1>

Exemplo:

Programa que calcula a tabuada de qualquer número quantas vezes o usuário desejar, utilizando a estrutura REPITA com REPITA.

```
PROGRAMA Exemplo_Repita_com_Repita
VARIÁVEIS
  contador, num: INTEIRO
  resposta: CARACTER
INÍCIO
  REPITA
    LEIA num
    contador <- 1
    REPITA
      ESCREVA contador, "X", num, "=", contador * num
      contador <- contador + 1
    ATÉ QUE (contador > 10)
    ESCREVA "Deseja continuar? (sim/não)"
    LEIA resposta
  ATÉ QUE (resposta = "não")
```

```
FIM
```

REPITA com ENQUANTO

Portugol:

REPITA
ENQUANTO <condição 2> FAÇA
<comandos>
FIM ENQUANTO
ATÉ QUE <condição 1>

Exemplo:

Programa que calcula a tabuada de qualquer número quantas vezes o usuário desejar» utilizando a estrutura REPITA com ENQUANTO.

```
PROGRAMA Exemplo_Repita_com_Enquanto
VARIÁVEIS
  contador, num: INTEIRO
  resposta: CARACTER
INÍCIO
  REPITA
    LEIA num
    contador <- 1
    ENQUANTO (contador < 10) FAÇA
      ESCREVA contador, "X", num, "", contador * num
      contador <— contador + 1
    FIM ENQUANTO
    ESCREVA "Deseja continuar? (sim/nao)"
    LEIA resposta
  ATÉ QUE (resposta = "nao")
FIM
```

REPITA com PARA

Portugol:
REPITA
PARA <variável> DE <valor inicial> É <valor final>
PASSO <incremento> FAÇA
<comandos>
FIM PARA
ATÉ QUE <condição>

Exemplo:

Programa que calcula a tabuada de qualquer número quantas vezes o usuário desejar, utilizando a estrutura REPITA com PARA.

```
PROGRAMA Exemplo Repita_com_Para
VARIÁVEIS
  contador, num: INTEIRO
  resposta: CARACTER
INÍCIO
  REPITA
    LEIA num
    PARA contador DE 1 ATÉ 10 PASSO 1 FAÇA
      ESCREVA contador, "X", num, "=", contador * num
    FIM PARA
    ESCREVA "Deseja continuar? (sim/nao)"
    LEIA resposta
  ATÉ QUE (resposta = "nao")
FIM
```

PARA COM PARA

Portugol:

PARA <variável> DE <valor inicial> ATÉ <valor final>
PASSO <incremento> FAÇA
PARA <variável> DE <valor inicial> ATÉ <valor final>
PASSO <incremento> FAÇA
<comandos>
FIM PARA
FIM PARA

Exemplo:

Programa que calcula três vezes a tabuada de qualquer número, utilizando a estrutura PARA com PARA.

```
PROGRAMA Exemplo_Para_com_Para
VARIÁVEIS
  contador, num, i: INTEIRO
  resposta: CARACTER
INÍCIO
  LEIA num
  PARA i DE 1 ATÉ 3 PASSO 1 FAÇA
    PARA contador DE 1 ATÉ 10 PASSO 1 FAÇA
      ESCREVA contador, "x", num, "=", contador * num
    FIM PARA
```

```
    ESCREVA "Digite outro n1Imero"
    LEIA num
  FIM PARA
FIM
```

PARA com ENQUANTO

Portugol:

PARA <variável> DE <valor inicial> ATÉ <valor tina!>
PAESO <incremento> FAÇA
ENQUANTO <condição> FAÇA
<comandos>
FIM ENQUANTO
FIM PARA

Exemplo: Programa que calcula três vezes a tabuada de qualquer número, utilizando a estrutura PARA com ENQUANTO.

```
PROGRAMA Exemplo_Para_com_Enquanto
VARIÁVEIS
  contador, num, i: INTEIRO
  resposta: CARACTER
INÍCIO
  LEIA num
  PARA i DE 1 ATÉ 3 PASSO 1 FAÇA
    contador <— 1
    ENQUANTO (contador < 10) FAÇA
      ESCREVA contador, "X", num, "=", contador * num
      contador <- contador + 1
    FIM ENQUANTO
    ESCREVA "Digite outro número"
    LEIA num
  FIM PARA
FIM
```

PARA com REPITA

Portugol:

PARA <variável> DE <valor inicial> ATÉ <valor final>
PASSO <incremento> FAÇA
REPITA
<comandos>
ATÉ QUE <condição>
FIM PARA

REPITA com PARA

Portugol:

REPITA
PARA <variável> DE <valor inicial> ATÉ <valor final>
PASSO <incremento> FAÇA
<comandos>
FIM PARA
ATÉ QUE <condiçâo>

Exemplo:

Programa que calcula a tabuada de qualquer número quantas vezes o usuário desejar, utilizando a estrutura REPITA com PARA.

```
PROGRAMA Exemplo Repita_com_Para
VARIÁVEIS
  contador, num: INTEIRO
  resposta: CARACTER
INÍCIO
  REPITA
    LEIA num
    PARA contador DE 1 ATÉ 10 PASSO 1 FAÇA
      ESCREVA contador, "X", num, "=", contador * num
    FIM PARA
    ESCREVA "Deseja Continuar? (sim/nao)"
    LEIA resposta
  ATÉ QUE (resposta = "nao")
FIM
```

PARA com PARA

Portugol:

PARA <variável> DE <valor inicial> ATÉ <valor final>
PASSO <incremento> FAÇA
PARA <variável> DE <valor inicial> ATÉ <valor final>
PASSO <incremento> FAÇA
<comandos>
FIM PARA
FIM PARA

Exemplo:

Programa que calcula três vezes a tabuada de qualquer número, utilizando a estrutura PARA com PARA.

```
PROGRAMA Exemplo Para_com_Para
VARIÁVEIS
  contador, num, i: INTEIRO
  resposta: CARACTER
INÍCIO
  LEIA num
  PARA i DE 1 ATÉ 3 PASSO 1 FAÇA
    PARA contador DE 1 ATÉ 10 PASSO 1 FAÇA
      ESCREVA contador, "X", num, "=", contador * num
    FIM PARA
    ESCREVA "Digite Outro número"
    LEIA num
  FIM PARA
FIM
```

Exemplo:

Programa que calcula três vezes a tabuada de qualquer número, utilizando a estrutura PARA com REPITA.

```
PROGRAMA Exemplo Para_com_Repita
VARIÁVEIS
  contador, num, i: INTEIRO
  resposta: CARACTER
INÍCIO
  LEIA num
  PARA i DE 1 ATÉ 3 PASSO 1 FAÇA
    contador <— 1
    REPITA
      ESCREVA contador, "X", num, "=", contador * num
      contador <- contador + 1
    ATÉ QUE (contador > 10)
    ESCREVA "Digite outro número"
    LEIA num
  FIM PARA
FIM
```

COMPARAÇÃO DAS ESTRUTURAS DE REPETIÇÃO

As três estruturas de repetição apresentadas neste livro possuem algumas características que as diferenciam uma das outras, como, por exemplo, a forma de verificar a condição da repetição. Nesse

caso, a estrutura ENQUANTO faz a verificação do teste lógico no início e a repetição da estrutura só ocorre se o resultado da condição for verdadeira, enquanto que a estrutura REPITA verifica a condição no final e o looping só continua se a resposta da condição for falsa. Já a estrutura PARA, não possui condição a ser verificada, pois a quantidade de repetições é predeterminada. A estrutura PARA é a menos versátil, porque ela não consegue substituir a estrutura ENQUANTO ou REPITA, quando essas estruturas utilizam como condição de repetição uma resposta do usuário. Por outro lado, as estruturas ENQUANTO e REPITA podem ser substituídas uma pela outra, além de substituir a estrutura PARA.