



Lógica de Programação

Aldo de Moura Lima

Curso Técnico em Informática

Educação a Distância

2017



EXPEDIENTE

Professor Autor

Aldo de Moura Lima

Design Instrucional

Deyvid Souza Nascimento

Renata Marques de Otero

Terezinha Mônica Sinício Beltrão

Revisão de Língua Portuguesa

Letícia Garcia

Diagramação

Izabela Cavalcanti

Coordenação

Anderson Elias

Coordenação Executiva

George Bento Catunda

Coordenação Geral

Paulo Fernando de Vasconcelos Dutra

Produzido pela Secretaria Executiva de Educação Profissional de Pernambuco, em
convênio com o Ministério da Educação (Rede e-Tec Brasil).

Setembro, 2017

L732I

Lima, Aldo de Moura.

Lógica de Programação : Curso Técnico em
Informática:

Educação a distância / Aldo de Moura Lima. – Recife:
Secretaria Executiva de Educação Profissional de
Pernambuco, 2016.

78 p.: il.

Inclui referências bibliográficas.

1. Educação a distância. 2. Lógica de Programação.
3. Informática. I. Lima, Aldo de Moura. II. Título. III. Rede e-
Tec Brasil.

CDD – 005.1



Sumário

Introdução	6
1.Competência 01 Conhecer os Princípios de Lógica de Programação Algorítmica	7
1.1 Dado e Informação.....	7
1.1.1 Dado	7
1.1.2 Informação	7
1.2 Sistema Computacional	9
1.2.1 Elementos de um Sistema Computacional	9
1.2.1.1 Hardware.....	10
1.2.1.2 Software	10
1.3 Desenvolvimento de Software.....	10
1.4 Codificação	12
1.4.1 Processamento de Dados.....	12
1.4.2 Processamento Eletrônico de Dados	14
1.4.3 Programa.....	16
1.4.4 Programação	16
1.4.5 Programador	16
1.5 Lógica de Programação	16
1.5.1 Fluxograma.....	17
1.5.2 Algoritmo	19
1.6 Lógica de Programação Algorítmica.....	20
1.7 Representação e Armazenamento dos Dados.....	21
1.7.1 Variável	21
1.7.1.1 Tipo de Variável.....	22
1.7.1.2 Declaração de Variável.....	22
1.7.2 Constante	26
1.8 Estrutura de um Algoritmo	27



1.9 Entrada de Dados	30
1.10 Saída de Informações.....	30
1.11 Atribuição de Dados	31
1.12 Ferramenta para Edição e Teste de Algoritmo	32
1.12.1 Tela Principal.....	33
1.12.2 Digitando o Algoritmo.....	36
2.Competência 02 Desenvolver um Algoritmo para a Realização de Operações Matemáticas.....	41
2.1 Operadores Aritméticos Básicos	41
2.2 Prioridade dos Operadores	42
2.3 Outros Operadores Aritméticos.....	44
2.4 Funções Aritméticas.....	46
2.5 Tabela de Prioridades.....	47
3.Competência 03 Desenvolver um Algoritmo para Resolução de um Problema Utilizando Estrutura de Decisão	49
3.1 Operação Condicional.....	49
3.2 Operadores Lógicos.....	52
3.3 Prioridade dos Operadores Lógicos	54
3.4 Estrutura de Decisão	61
3.4.1 Estrutura de Decisão Simples.....	61
3.4.2 Estrutura de Decisão Composta.....	63
3.4.3 Estrutura de Decisão Encadeada.....	66
4.Competência 04 Desenvolver um Algoritmo para Resolução de um Problema Utilizando Estrutura de Repetição.....	68
4.1 Estrutura de Repetição Indefinida, com uma Repetição Obrigatória.....	68
4.2 Contador	70
4.3 Acumulador.....	71
4.4 Estrutura de Repetição Indefinida, com Repetição Opcional	72



4.5 Estrutura com Repetição Definida	74
Conclusão	76
Referências	77
Minicurrículo do Professor	78



Introdução

Gostaria de dar a vocês as boas-vindas à disciplina **Lógica de Programação**. Os assuntos que serão tratados nesta disciplina fazem parte de uma área da Informática chamada de Desenvolvimento de Software e envolvem a programação do computador.

A programação permite instruir o computador para que ele realize as tarefas que desejamos como, por exemplo: controlar o estoque de uma empresa, simular os cenários de um jogo, escrever e enviar uma mensagem em uma rede social ou disponibilizar imagens em uma tela. Os computadores precisam de programas para fazer com que seus componentes eletrônicos processem os dados e realizem os resultados que desejamos.

A programação de computadores ocorre de forma diferente da que um ser humano utiliza para instruir outro ser humano. Esta forma envolve a lógica de programação e poderá ser expressa de várias maneiras. Utilizaremos a forma algorítmica, onde as instruções que o computador deve executar são codificadas em forma de texto, com comandos escritos na língua portuguesa.

Nesta disciplina você aprenderá os princípios e as principais estruturas que regem a programação. Realizando os exercícios adequadamente, o seu raciocínio estará preparado para criar qualquer tipo de programa, quando em disciplina posterior aprender uma linguagem de programação específica para computação.

Tenho certeza de que você não vai querer ficar fora desta oportunidade. Como programação é uma área muito abrangente, você precisará complementar seus estudos através de outros materiais que estaremos propondo neste caderno e em outros que você, sem dúvida, irá pesquisar.

Vamos nessa?

Aldo Moura



1.Competência 01 | Conhecer os Princípios de Lógica de Programação Algorítmica

Iniciaremos nossos estudos conhecendo os princípios da Lógica de Programação, mas primeiramente precisamos compreender alguns conceitos da área da Informática.

1.1 Dado e Informação

São dois conceitos elementares da Informática.

1.1.1 Dado

Um dado representa um fato básico que pode sofrer manipulação.

Pode ser de diversos tipos, como:

- **Numérico:** composto apenas de números que normalmente utilizamos em operações aritméticas.

Exemplos: 2, 20.3, 25

- **Alfanumérico ou texto:** composto de letras, símbolos e/ou números que normalmente não serão usados em operações aritméticas.

Exemplos: Antonio, 2, maria@gmail.com, 2, F#, casa

- **Imagens:** composto de figuras ou fotos.
- **Áudio:** composto de sons ou ruídos.
- **Vídeo:** composto de imagens ou figuras em movimento.

1.1.2 Informação

Temos uma informação quando, através de processamento ou interpretação, de acordo com um contexto, um significado é atribuído a um ou vários dados.

Este significado dá um valor adicional ao dado, além do valor do fato representado.



Exemplos 1: nestes exemplos, para cada dado foi atribuído um significado, respectivamente: dias, anos, símbolo monetário do real e telefone.

- 53 dias
- 53 anos
- R\$ 53,00
- Telefone 99999-0101

Exemplos 2: nos exemplos abaixo temos três listas que possivelmente sofreram processamento, resultando em:

1ª lista: relação dos nomes de pessoas em ordem alfabética.

2ª lista: relação dos nomes de pessoas do sexo feminino em ordem alfabética.

3ª lista: relação dos nomes de pessoas do sexo masculino em ordem alfabética.

Afonso	Beatriz	Afonso
Aluísio	Betânia	Aluísio
Antonio	Marta	Antonio
Beatriz	Mércia	Cássio
Betânia	Rute	Celso
Cássio	Vânia	Cícero
Celso	Verônica	Paulo
Cícero	Zélia	Pedro
Marta		Renato
Mércia		
Paulo		
Pedro		
Renato		
Rute		
Vânia		
Verônica		
Zélia		

1.2 Sistema Computacional

Um sistema baseado em computador é um composto de infraestrutura tecnológica cujos elementos são organizados para coletar, manipular e processar dados em informações.

1.2.1 Elementos de um Sistema Computacional

Um sistema computacional é composto pelos elementos representados na figura 1.

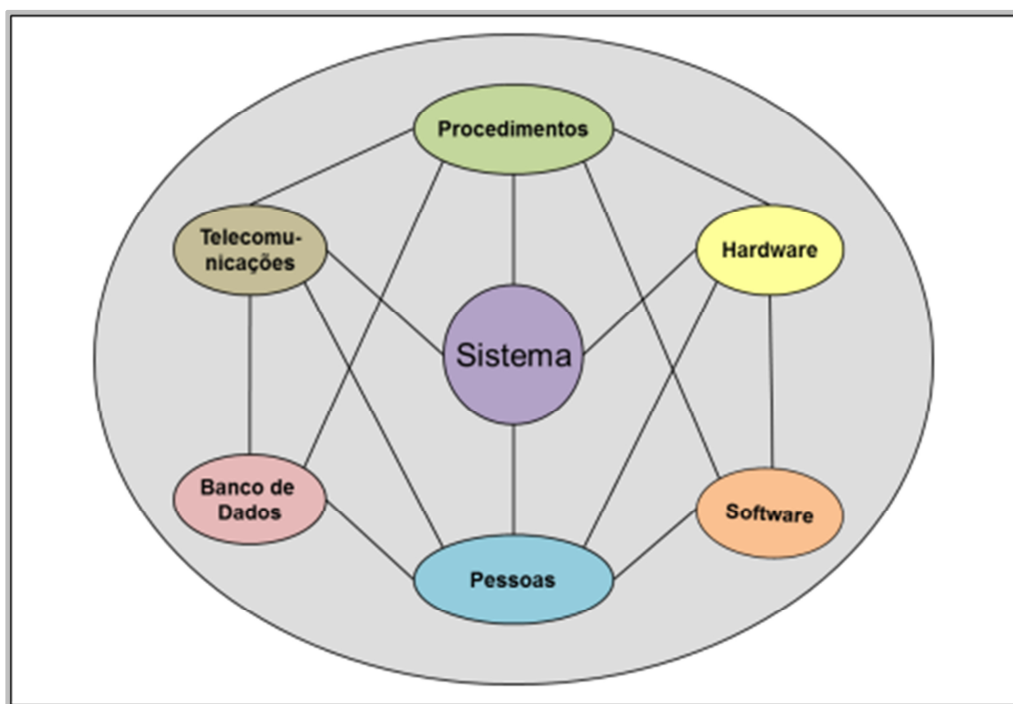


Figura 1- Representação da interação dos Elementos de um sistema computacional

Fonte: o autor (2013)

Descrição: a imagem mostra um círculo central preenchido na cor roxa com a palavra sistema no centro do círculo. Ao redor deste círculo estão dispostas seis elipses. Cada elipse representa um dos elementos de um sistema computacional. A primeira elipse, disposta acima do círculo central contém a palavra procedimento e está preenchida na cor verde. Considerando-se o sentido anti-horário, a segunda elipse contém a palavra hardware e está preenchida na cor amarela; a terceira elipse contém a palavra software e está preenchida na cor laranja; a quarta elipse contém a palavra pessoas e está preenchida na cor azul; a quinta elipse contém a palavra banco de dados e está preenchida na cor rosa e a última elipse contém a palavra telecomunicações e está preenchida na cor marrom. Existem linhas ligando o círculo a algumas elipses, bem como linhas ligando as elipses, para representar que os elementos interagem entre si para formar o sistema. Esta formação do sistema está representada por uma elipse maior que envolve todas as seis elipses. Essa elipse maior está preenchida na cor cinza.



Os outros elementos de um sistema computacional:

- **Pessoas:** todos que gerenciam, usam, programam e mantêm o sistema computacional.
- **Banco de dados:** coleção de dados e informações acessadas pelo software que integra o sistema.
- **Telecomunicações:** transmissão eletrônica de sinais para comunicações que permitem às organizações executarem seus processos e tarefas por meio de redes de computadores.
- **Procedimentos:** estratégias, políticas, métodos e regras que definem o uso dos elementos do sistema.

Os elementos que mais reconhecemos em um sistema computacional são: Hardware e Software.

1.2.1.1 Hardware

Os hardwares são os equipamentos eletrônicos como: computador, impressora e pen drive.

1.2.1.2 Software

Os softwares são os programas que determinam o que os hardwares devem fazer como, por exemplo: jogos, sites, editores de texto e controle de estoque.

Este módulo trata essencialmente sobre o desenvolvimento de software.

1.3 Desenvolvimento de Software

A ciência que trata do desenvolvimento de software é conhecida como Engenharia de Software. Ela prevê que para um software se desenvolver deve haver um processo de software definido.

O processo de software mais tradicional é conhecido como Modelo Cascata, representado na figura 2.

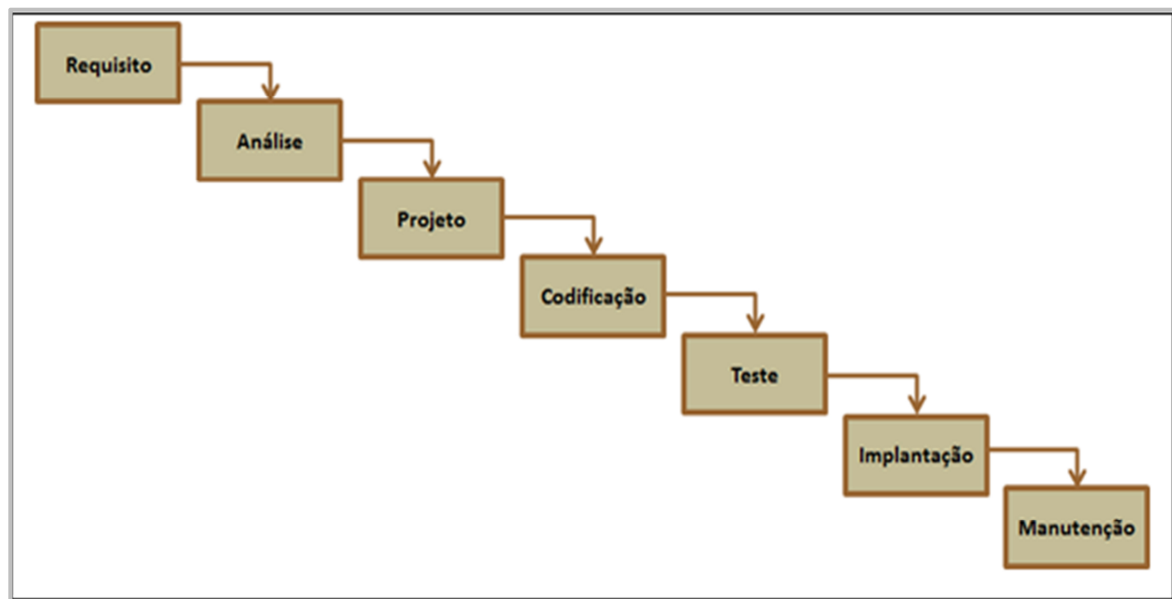


Figura 2- Representação do processo de Software - Modelo Cascata

Fonte: o autor (2013)

Descrição: a imagem mostra sete retângulos preenchidos na cor marrom, onde cada retângulo representa uma etapa do processo. Os retângulos estão dispostos um após o outro em degraus decrescentes. No centro da lateral direita de cada retângulo sai uma linha horizontal que faz uma curva de 90 graus e se liga ao centro da lateral superior do retângulo seguinte. No final de cada linha tem uma seta indicando que as etapas ocorrem uma após a outra. Em cada retângulo encontra-se o nome da etapa, sendo o primeiro retângulo a etapa de requisitos; o segundo retângulo a etapa de análise; o terceiro retângulo a etapa de projeto; o quarto retângulo a etapa de codificação; o quinto retângulo a etapa de teste; o sexto retângulo a etapa de implantação e o último retângulo a etapa de manutenção.



Processo de Software:

Conjunto de atividades relacionadas que levam à produção de um software.



Para conhecer outros modelos, pesquise sobre **Modelo de Processo de Software**. Além do modelo cascata, descrito acima, você deverá encontrar vários outros modelos: incremental, prototipagem e espiral.

Etapas do modelo cascata:

- **Requisito:** identificação e especificação do que o software deve fazer.
- **Análise:** definição dos dados necessários e como eles se relacionam.



- **Projeto:** determinações de como as funcionalidades do sistema serão implantadas.
- **Codificação:** construção do programa, traduzindo-se o projeto das funcionalidades em um conjunto de instruções.
- **Teste:** Verificação para saber se as funcionalidades implantadas atendem aos requisitos especificados.
- **Implantação:** o software é instalado e colocado em uso.
- **Manutenção:** o software sofre alterações para correção, adaptação ou acréscimo de funcionalidades.

Normalmente a etapa de Codificação é executada por profissionais de nível técnico – um **programador**.

Na execução desta etapa, dois fatores são relevantes:

1. A codificação é realizada através da escrita de um programa, que é um conjunto de instruções codificado em uma linguagem específica, chamado de linguagem de programação, como por exemplo: Java, C# (lê-se C sharp) e PHP.
2. Como o programa instrui um hardware, que possui limitações em relação ao ser humano, o conjunto de instruções precisa atender ao modo próprio do hardware lidar com cada instrução. Este modo próprio deve obedecer a uma lógica específica denominado de **lógica de programação**. Entender esta lógica será o objetivo desta disciplina.

1.4 Codificação

Na etapa da codificação serão escritos os programas que permitirão ao computador realizar o processamento eletrônico de dados.

1.4.1 Processamento de Dados

Consiste em um conjunto de atividades ordenadas com o objetivo de manipular dados iniciais em informações desejáveis.

As etapas de um processamento de dados são:



Figura 3 – Representação das etapas do Processamento de dados

Fonte: o autor (2013)

Descrição: três retângulos preenchidos na cor cinza, onde cada retângulo representa uma etapa do processamento de dados. O primeiro retângulo contém a palavra entrada, o segundo retângulo contém a palavra processamento e o terceiro retângulo a palavra saída. Existe uma linha do primeiro para o segundo retângulo com uma seta apontando para o segundo retângulo e uma linha do segundo para o terceiro retângulo com uma seta apontando para o terceiro retângulo, indicando a ordem das etapas.

A entrada é responsável pela obtenção dos dados iniciais.

A manipulação dos dados iniciais é chamada de processamento.

O resultado do processamento é a saída.

A etapa da entrada é onde ocorre a inserção dos dados iniciais, que serão necessários para os resultados que desejamos obter.

Os dados inseridos, de acordo com a necessidade, podem ser manipulados de várias formas, tais como: usados em cálculos matemáticos ou organizados em ordem alfabética ou numérica. Esta manipulação é chamada de etapa de processamento.

A etapa da saída é onde as informações desejadas, obtidas com o processamento, são apresentadas aos interessados.

Exemplos de processamento de dados:

- **Exemplo 1:** ordenar uma lista de nomes

Entrada: nomes desordenados

Processamento: ordenação

Saída: lista ordenada de nomes

- **Exemplo 2:** calcular uma média de notas



Entrada: notas

Processamento: cálculo

Saída: média

Como os dados de saída são mais qualificados, esses dados são denominados de informações.

Estas três fases também estão presentes no mecanismo de um sistema, considerado como um conjunto de elementos que interagem para atingir objetivos.

Exemplos de processamento de dados:

- Sistema digestório
- Sistema sanguíneo
- Sistema de transporte
- Sistema hidráulico
- Sistema elétrico
- Sistema de saúde

Cada um destes sistemas é composto por diversos elementos que colaboram para que o objetivo desejado seja atingido.

1.4.2 Processamento Eletrônico de Dados

Quando um processamento de dados é realizado por equipamentos eletrônicos, por exemplo, microcomputadores, tablets e smartphones, temos o que denominamos de processamento eletrônico de dados.

Dependendo da quantidade de dados que desejamos processar e do processamento que queremos realizar, a quantidade de instruções de cada atividade pode variar de uma a centenas a milhares de instruções.

- **Exemplo 1:** para encontramos o maior número entre dois números iniciais, precisamos basicamente de cinco instruções:

1. Obter o 1º número;



2. Obter o 2º número;
3. Comparar o 1º número com o 2º número;
4. Memorizar o maior número comparado;
5. Exibir o número memorizado.

• **Exemplo 2:** Para encontramos o maior número entre três números iniciais, precisamos basicamente de oito instruções:

1. Obter o 1º número;
2. Obter o 2º número;
3. Obter o 3º número;
4. Comparar o 1º número com o 2º número;
5. Memorizar o maior número comparado;
6. Comparar o 3º número com o maior número memorizado;
7. Memorizar o maior número na nova comparação;
8. Exibir o número memorizado.

Aumentou apenas um número na comparação, mas a quantidade de instruções quase dobrou.

À medida que a quantidade de números a serem comparados aumenta, ou se consideramos que pode haver números iguais, a quantidade de instruções aumenta consideravelmente.

Imaginem a quantidade de instruções que deve ter um processamento para processar um pedido, envolvendo cálculo do valor total, cálculo dos impostos, cálculo dos descontos, fazer baixa do estoque, calcular as parcelas quando for vender a prazo, fazer os lançamentos da contabilidade, calcular a comissão do vendedor, etc.

No processamento eletrônico de dados, um roteiro com as instruções para processamento é denominado de programa.



1.4.3 Programa

Conjunto de instruções que um computador interpreta para executar uma tarefa específica.

Sendo uma máquina eletrônica, o computador está submetido às regras da eletrônica digital. Com isso, as diversas regras das linguagens naturais (português, inglês, japonês, etc.) não são entendidas pelo computador. Então, foram criadas linguagens de programação com regras próprias, indicando como as instruções deverão ser codificadas.

1.4.4 Programação

É a atividade de elaborar um programa para o computador.

1.4.5 Programador

O profissional responsável pela elaboração do programa.

1.5 Lógica de Programação

Como uma linguagem de programação tem regras próprias, as instruções de um programa deverão ser escritas em uma sequência lógica, para que a tarefa seja de fato executada. Esta sequência lógica é o que chamamos de **lógica de programação**.

Antes de escrever o programa propriamente dito, é comum primeiro fazermos uma representação abstrata, sem muitos detalhes, da ideia (lógica) que temos para resolver o problema para o qual o programa está sendo criado. Os arquitetos ou engenheiros civis normalmente fazem algo parecido. Quando um cliente solicita que eles elaborem um projeto para a construção de um imóvel, antes de desenhar a planta arquitetônica, eles primeiro fazem um rascunho (croqui) para entender melhor o desejo do cliente e verificar se a proposta desenhada no croqui atende ao que o cliente quer. Quando é verificado que realmente atende, a planta é finalmente elaborada com todos os detalhes necessários, de acordo com as regras da engenharia civil.

No caso da programação, as duas formas básicas para representar a lógica de programação são fluxograma e algoritmo.



Linguagem de programação:

conjunto de comandos (texto e símbolos) com sentido predeterminado e regras de sintaxe próprias que permitem a construção de instruções que um computador pode executar.

Codificação:

termo utilizado para substituir a programação, já que o ato de programar envolve escrever um programa nos códigos específicos de uma linguagem de programação.

1.5.1 Fluxograma

É uma forma de representação **gráfica** da lógica de programação, que utiliza **símbolos** como comandos para o computador. O programador deve conhecer os símbolos predefinidos que pode utilizar e o que cada símbolo representa, como, por exemplo, os símbolos relacionados na figura 4.

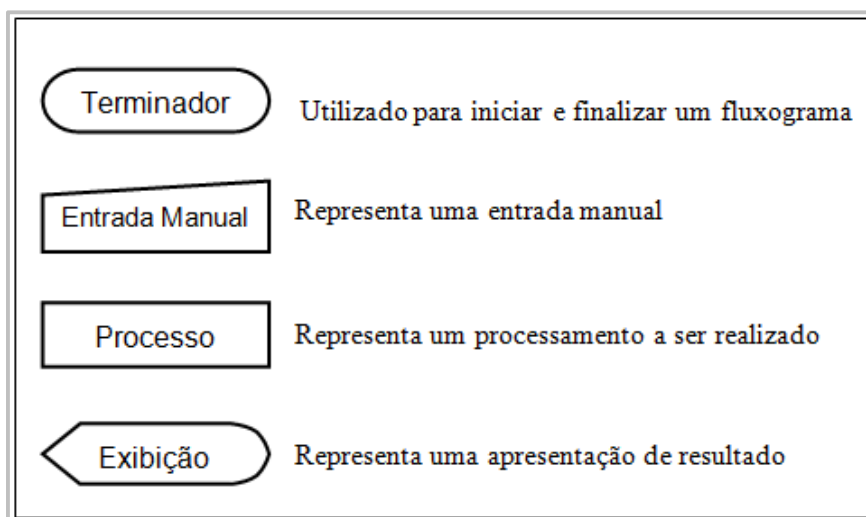


Figura 4-Exemplos de símbolos usados em fluxograma

Fonte: o autor (2013)

Descrição: quatro símbolos utilizados para elaboração de fluxogramas. O primeiro símbolo se parece com um retângulo, sendo que a lateral esquerda e a lateral direita são arredondadas como um meio círculo, com as curvas para os lados de fora do retângulo. Neste símbolo, encontra-se a palavra terminador e ao lado direito um texto explicando o objetivo deste símbolo, que é iniciar e finalizar um fluxograma. O segundo símbolo é um trapézio com a palavra entrada manual e o texto explicativo ao lado direito informa que representa uma entrada de dado via digitação. O terceiro símbolo é um retângulo que em seu interior tem o texto processo e o texto explicativo em seu lado direito informa que representa um processamento a ser realizado. O quarto símbolo se parece com um retângulo, sendo que a lateral esquerda é pontuda como um sinal de maior que, e a lateral direita é arredondada como um meio círculo. No seu interior há o texto exibição e do lado direito um texto explicando o objetivo do símbolo, que é indicar a exibição de resultado.

A partir dos exemplos de símbolos da figura 4 já podemos representar a lógica de programação por fluxograma para somar dois números (veja Figura 5).

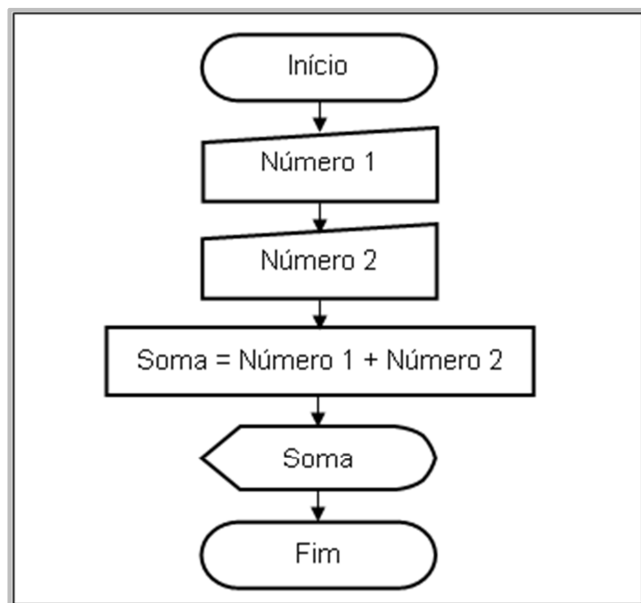


Figura 5 - Fluxograma para somar dois números

Fonte: o autor (2013)

Descrição: a imagem mostra um fluxograma que recebe dois números e exibe a soma deles. Além de dois terminadores, um iniciado e outro finalizando o fluxograma, encontramos dois símbolos de entrada manual representando a entrada dos dois números, representados por número 1 e número 2. Em seguida, o símbolo de processo que executa a soma dos valores representados por número 1 e número 2, armazenando o resultado em soma. Finalizando, foi colocado um símbolo de exibição para mostrar o valor de soma.

E, então, considerando o significado dos símbolos da figura 4, você consegue descrever o que o fluxograma da figura 5 faz?

Escreva em uma folha de papel e depois compare com a descrição abaixo.

- O fluxograma tem início com o terminador nomeado de **início**;
- Um número dará entrada no sistema, e, conforme o símbolo, a entrada será manual (pode ser um usuário digitando via teclado);
- Outro número dará entrada no sistema, também de forma manual;
- Haverá um processamento onde os dois números que deram entrada no sistema serão somados e o resultado representado pela palavra Soma;
- O resultado é apresentado;
- O fluxograma encerra com o terminador nomeado de **fim**.



Muito bem, espero que tenha acertado. Pode não ter ficado igual ao que fiz, mas o importante é que você tenha entendido.

1.5.2 Algoritmo

É uma forma de representação **textual** da lógica de programação, que utiliza **palavras** como comandos para o computador. O programador deve conhecer as palavras predeterminadas da linguagem algorítmica que pode utilizar e o que cada palavra representa. Dizemos que as palavras da linguagem algorítmica são **palavras reservadas**, como as palavras relacionadas na figura 6.

<u>início</u>	Usada para iniciar um algoritmo
<u>fim</u>	Usada para finalizar um algoritmo
<u>leia</u>	Usada para entrada de dados
<u>escreva</u>	Usada para exibir um resultado
:=	Usada para atribuir um valor (processamento)

Figura 6 - Exemplos de palavras reservadas usadas em algoritmo

Fonte: O autor (2013)

Descrição: a imagem mostra cinco palavras reservadas utilizadas para elaboração de algoritmo e ao lado de cada palavra um texto explicando para que cada palavra é usada. A primeira é a palavra **início**, escrita sem acentuação, usada para iniciar um algoritmo. A segunda é a palavra **fim**, usada para finalizar um algoritmo. A terceira é a palavra **leia**, usada para entrada de dados. A quarta é a palavra **escreva**, usada para exibir um resultado. A última palavra reservada é formada pelo caractere dois pontos e o caractere igual. Eles são escritos juntos e servem para atribuir um valor no processamento.

A maioria das palavras reservadas são palavras utilizadas na língua portuguesa (**início**, **fim**, **leia**, **escreva**). No entanto, por se tratar de algoritmo, as palavras não são acentuadas. Também há um grupo de palavras (conjunto de caracteres) que não tem correspondente na língua portuguesa (:=, <').

As palavras reservadas só devem ser utilizadas no algoritmo para o que foram determinadas. A palavra **leia**, por exemplo, foi reservada para significar entrada de dados. Caso você tenha algum item do algoritmo que queira chamar de **leia**, utilize um sinônimo. No caso de **leia** pode ser **estude**.

A partir dos exemplos de palavras reservadas da figura 6 já podemos representar a lógica de



programação algorítmica para somar dois números (veja Figura 7).

```
inicio  
leia (numero1)  
leia (numero2)  
soma := numero1 + numero2  
escreva (soma)  
fim
```

Figura 7 - Exemplo de um algoritmo

Fonte: o autor (2013)

Descrição: a imagem mostra um algoritmo que recebe dois números e exibe a soma deles. Após iniciar o algoritmo com a palavra reservada **inicio**, encontramos duas entradas manuais para receber dois números, representados por **numero1** e **numero2**. Segue-se um processamento onde **numero1** é somado ao **numero2** e o resultado armazenando em **soma**, que em seguida é exibido. Encerra-se o algoritmo com a palavra reservada **fim**.

Creio que você é um bom observador e deve ter percebido alguns detalhes. Algumas palavras reservadas estão sublinhadas, os dados a serem lidos nos comandos **leia** e o dado a ser exibido no comando **escreva** estão entre parênteses. Estes e alguns outros detalhes de um algoritmo você aprenderá no decorrer da disciplina.

1.6 Lógica de Programação Algorítmica

Conforme já explicado, a lógica de programação algorítmica é a maneira de representar a lógica (ideia) de programação que temos para resolver um determinado problema em forma de algoritmo.

Esta forma utiliza uma linguagem própria – linguagem algorítmica, com regras próprias.

Não existe apenas uma maneira de resolver um problema. Cada programador pode desenvolver ideias diferentes e ambos conseguem resolver o mesmo problema. O que vai caracterizar que o algoritmo está correto é a conformidade do resultado que ele gera em relação aos dados de entrada.

Uma linguagem de programação algorítmica é uma linguagem didática, ou seja, utilizada para ensinar lógica de programação, não sendo usada em programação real de computadores. Por ser uma linguagem apenas didática, cada escola pode desenvolver suas próprias palavras reservadas e suas próprias regras de sintaxe. Após o algoritmo ser testado, e comprovado que a lógica está



correta, o programador precisa codificar o programa em uma linguagem de programação.

1.7 Representação e Armazenamento dos Dados

Na programação podemos representar e armazenar os dados de duas maneiras: variável ou constante, conforme veremos a seguir.

1.7.1 Variável

Um programa pode ser executado diversas vezes, com a vantagem de que em cada execução forneçamos dados de entrada diferentes. Na codificação do programa, o dado precisa ser representado, mas o valor dele só será fornecido quando o programa for executado. Como pode ser fornecido qualquer valor, esta representação do dado é chamada de variável. No algoritmo da figura 8 podemos observar em destaque as variáveis `numero1`, `numero2` e `soma`. Elas representam qualquer valor. Se em uma execução do programa forem fornecidos os valores 10 e 20 respectivamente para as variáveis `numero 1` e `numero 2`, o valor da variável `soma` será 30. Se em outra execução forem fornecidos os valores 5 e 7, o valor da variável `soma` será 12. Não é fácil?

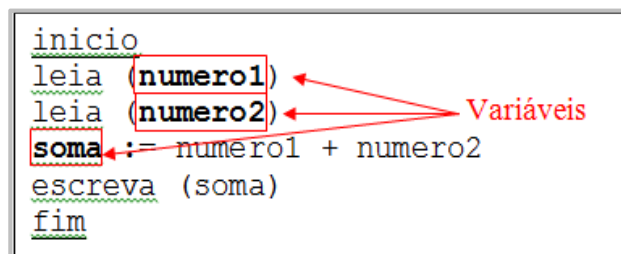


Figura 8 - Algoritmo indicando as variáveis utilizadas no processamento

Fonte: o autor (2013)

Descrição: algoritmo que recebe dois números e exibe a soma deles. Após iniciar o algoritmo com a palavra reservada `inicio`, encontramos duas entradas manuais para receber dois números, representados por `numero1` e `numero2`. Segue-se um processamento onde `numero1` é somado ao `numero2` e o resultado armazenando em `soma`, que em seguida é exibido e se encerra o algoritmo com a palavra reservada `fim`. No lado direito do algoritmo há um texto com a palavra `variável` e três setas apontam do texto para os elementos `numero1`, `numero2` e `soma`, indicando que esses elementos representam variáveis no algoritmo.

O programador deve ter o cuidado de **declarar** antecipadamente todas as variáveis que vai utilizar em seu algoritmo, de acordo com o tipo de dado que a variável vai representar. Os dados ocupam espaço na memória do computador e a declaração de variáveis é a maneira de reservar os espaços



de que os dados necessitam.

1.7.1.1 Tipo de Variável

Os tipos de variáveis que iremos utilizar são:

- **Caractere ou literal:** armazena dado textual, que normalmente não é usado em operações matemáticas. Os dados literais são descritos entre aspas.

Exemplos: “Carlos”, “jose@gmail.com”, “Sala B2”

- **Inteiro:** armazena dado numérico inteiro. Exemplos: 2, 150
- **Real ou numérico:** armazena dado numérico com decimais. Exemplos: 2.5, 124.55
- **Lógico:** armazena apenas os valores lógicos. Os dados lógicos são descritos sem aspas e podem ser VERDADEIRO ou FALSO.

A partir de agora iremos apresentar formalmente os comandos da linguagem de programação algorítmica. Sempre que um novo comando for apresentado, informaremos a sintaxe (forma correta) de sua utilização.



O formato da sintaxe é: `nome_comando parâmetro`

Onde: `nome_comando`: é o nome do comando em estudo.

`parâmetro`: complemento que pode ser acrescentado ao comando para que a instrução possa ser executada pelo computador.

Obs.: Se o parâmetro estiver entre parênteses angulares “<>”, o parâmetro é obrigatório. Se o parâmetro estiver entre colchetes “[]”, o parâmetro é opcional.

1.7.1.2 Declaração de Variável

O espaço reservado pela declaração depende do tipo de dado declarado. O programador apenas faz a declaração e o sistema operacional, que gerencia os recursos do computador, encarrega-se de reservar o espaço específico.



Sintaxe para declaração de variáveis:

var

<nome_variável>: <tipo>

Onde:

nome_variável = nome da variável. **Não pode ter espaço, símbolos (exceto sublinhado “_”) e nem iniciar por número.** Não pode ser uma palavra reservada utilizada na linguagem de programação, como algoritmo, var, inicio, etc. Deve ser um nome sugestivo em relação ao dado que vai armazenar.

tipo = indica o tipo de dados que a variável pode armazenar.

Exemplo:

```
var
    numerol: real
```

Note que o nome da variável e o tipo são parâmetros obrigatórios, pois estão entre parênteses angulares “<>”.

Em um comando de declaração var deverão ser declaradas todas as variáveis do algoritmo, como no exemplo a seguir:

```
var
    numerol: real
    numero2: real
    nome: literal
```



As variáveis do mesmo tipo podem ser agrupadas em uma linha, separando-as por vírgula, com apenas uma especificação do tipo, conforme o seguinte exemplo:

```
var
    numero1, numero2: real
    nome: literal
```

Veja como fica o algoritmo da figura 7 com o comando de declaração das variáveis na figura 9 a seguir:

```
var
    numero1: inteiro
    numero2: inteiro
    soma: inteiro
inicio
    leia (numero1)
    leia (numero2)
    soma := numero1 + numero2
    escreva (soma)
fim
```

Figura 9 - Algoritmo com declaração de variáveis

Fonte: O autor (2013)

Descrição: algoritmo que começa declarando as variáveis que serão utilizadas no processamento. A declaração ocorre através da palavra reservada var. Em seguida, linha após linha as variáveis são descritas deslocadas da margem esquerda de três posições. Após a declaração das variáveis, inicia-se o algoritmo propriamente dito com a palavra reservada inicio, depois temos duas entradas manuais para receber dois números, representados por numero1 e numero2. Segue-se um processamento onde numero1 é somado ao numero2 e o resultado armazenando em soma, que em seguida é exibido e se encerra o algoritmo com a palavra reservada fim.

Você também deve ter observado que as variáveis foram colocadas na linha seguinte ao comando var e não se encontram alinhadas na mesma margem esquerda que o comando. Isso é para permitir uma melhor organização do código. Nada impede que todas as variáveis estejam na mesma margem esquerda do comando var, ou mesmo que a primeira variável esteja na mesma linha do comando var, como no exemplo da figura 10.



```
var numero1: inteiro
numero2: inteiro
soma: inteiro
inicio
leia (numero1)
leia (numero2)
soma := numero1 + numero2
escreva (soma)
fim
```

Figura 10 - Algoritmo com declaração de variáveis sem indentação

Fonte: o autor (2013)

Descrição: algoritmo que começa declarando as variáveis que serão utilizadas no processamento. A declaração ocorre através da palavra reservada var. Em seguida, linha após linha as variáveis são descritas sem o deslocamento da margem esquerda, já que o deslocamento é opcional e serve para organizar o algoritmo. Após a declaração das variáveis, inicia-se o algoritmo propriamente dito com a palavra reservada inicio, depois temos duas entradas manuais para receber dois números, representados por numero1 e numero2. Segue-se um processamento onde numero1 é somado ao numero2 e o resultado armazenando em soma, que em seguida é exibido e se encerra o algoritmo com a palavra reservada fim.

Não são todas as linguagens de programação que exigem o comando de declaração de variáveis e em algumas delas a declaração pode ser feita em qualquer parte do código, como nas linguagens Java e Python.

O recurso de deslocar os parâmetros subordinados a um comando ou também o de deslocar os comandos subordinados a outros comandos, caso dos comandos que estão entre os comandos inicio e fim, (como na figura 11 a seguir) é conhecido como indentação. Normalmente, três posições são deslocadas em relação à margem esquerda do comando superior. É um recurso profissionalmente exigido para melhorar a compreensão do código, principalmente em futuras modificações do algoritmo. Veja como o código da figura 11 é mais legível que o código da figura 10.



```
var
    numero1: inteiro
    numero2: inteiro
    soma: inteiro
inicio
    leia (numero1)
    leia (numero2)
    soma := numero1 + numero2
    escreva (soma)
fim
```

Figura 11- Algoritmo demonstrando o uso da indentação

Fonte: o autor (2013)

Descrição: algoritmo que começa declarando as variáveis que serão utilizadas no processamento. A declaração ocorre através da palavra reservada var. Em seguida, linha após linha, as variáveis são descritas deslocadas três posições da margem esquerda. Após a declaração das variáveis, inicia-se o algoritmo propriamente dito com a palavra reservada início, depois temos duas entradas manuais para receber dois números, representados por numero1 e numero2. Segue-se um processamento onde numero1 é somado ao numero2 e o resultado armazenando em soma, que em seguida é exibido e encerra-se o algoritmo com a palavra reservada fim. Neste algoritmo, as duas entradas manuais, o processamento e a exibição do resultado também estão deslocados três posições em relação à margem esquerda.

A indentação será exigida em nossos exercícios.

1.7.2 Constante

Quando em um algoritmo temos a necessidade de representar um dado que não sofrerá alteração, ao invés de variáveis temos uma constante. Por exemplo, se o problema é obter a média aritmética entre duas notas, teremos que somar as duas notas e dividir a soma por 2. Logo, o divisor sempre será 2. Não precisa declarar uma variável para que o valor 2 seja fornecido na execução do programa. Já durante a programação podemos definir o valor do divisor como 2. Em nossa linguagem algorítmica as constantes não são declaradas, basta escrevê-las no algoritmo. Observe no exemplo da figura 12 o uso de uma constante. Não se preocupe agora com a forma que a expressão matemática foi codificada, pois este será assunto da nossa próxima competência. Observe apenas que na linha com a expressão que calcula a média aritmética, o número 2 sempre terá o mesmo valor. Logo, é um dado constante.



```
var
    nota1, nota2: inteiro
    media: real
inicio
    leia (nota1)
    leia (nota2)
    media := (nota1 + nota2) / 2
    escreva (media)
fim
```

Constante

Figura 12 - Algoritmo para calcular a média aritmética, a partir de duas notas

Fonte: o autor (2013)

Descrição: a imagem mostra um algoritmo que começa declarando as variáveis que serão utilizadas no processamento, que são nota1, nota2 e media. Após o início do algoritmo são colocadas as entradas das nota1 e nota2, seguindo-se do processamento da média, que é a soma da nota1 e da nota2 e o resultado dividido por 2. Em seguida, a média calculada é exibida. No lado esquerdo do algoritmo há a palavra constante e uma seta apontando para o número 2 na expressão que calcula a média.

Os tipos de constantes também seguem os mesmos tipos de variáveis: literal ou texto; inteiro; real ou numérico e lógico.



Algumas linguagens de programação algorítmica utilizam declaração de constantes da seguinte forma:

CONSTANTE

PI = 3.14

Onde PI é o nome da constante e 3.14 é o valor atribuído à mesma.

1.8 Estrutura de um Algoritmo

Até o momento você aprendeu como fazer parte de um algoritmo. Ainda estão faltando alguns detalhes para que o algoritmo esteja completo e possa ser testado.

A linguagem de programação algorítmica que estudamos utiliza a seguinte estrutura para um algoritmo:

```
algoritmo "nome_algoritmo"
// Seção de Declarações
var
inicio
```



```
// Seção de Comandos  
fimalgoritmo
```

Um algoritmo inicia com o comando algoritmo e finaliza com o comando fimalgoritmo.

A sintaxe do comando algoritmo é:

algoritmo <"nome_algoritmo">

Onde:

nome_algoritmo = é um nome, entre aspas, que obrigatoriamente deve ser dado ao algoritmo.

Exemplos:

```
algoritmo "soma"  
algoritmo "media"  
algoritmo "calcular média de 2 notas"  
algoritmo "## calcular área ##"
```

O nome de algoritmo pode conter espaço, número ou caracteres especiais. O ideal é colocar um nome sugestivo, de acordo com o problema que o algoritmo resolve.

A sintaxe do comando fimalgoritmo é simplesmente só o nome do comando, sem nenhum parâmetro adicional.

Um algoritmo tem duas seções:

A primeira é a seção de declaração, onde deve ser colocado o comando `var`, já estudado nesta competência.

A segunda é a seção de comando, que inicia com o comando `inicio`. É onde as instruções que orientam o computador são codificadas.

Com base nesta estrutura vamos codificar totalmente um algoritmo que soma dois números (veja a figura 13).



```
algoritmo "Somar 2 números"
// Seção de Declarações
var
    numero1, numero2, soma: inteiro
// seção de comandos
inicio
    leia (numero1)
    leia (numero2)
    soma := numero1 + numero2
    escreva (soma)
finalgoritmo
```

Figura 13 - Algoritmo com comentários

Fonte: o autor (2013)

Descrição: algoritmo para receber dois números e mostrar a soma deles. As duas seções de um algoritmo são indicadas por linhas iniciadas por duas barras juntas, apontando que as linhas são apenas comentários, utilizados apenas para melhorar o entendimento do algoritmo. Logo após a palavra algoritmo, que define o início do algoritmo, foi colocado uma linha com barra dupla e o texto seção de declarações. Na linha anterior à palavra reservada, inicio, em que começa a sequência lógica dos comandos, foi colocada uma linha iniciada com barra dupla e o texto seção de comandos.

As linhas que descrevem as seções não precisam ser codificadas. Linhas iniciadas com os caracteres // (barras duplas) indicam comentários para melhor entendimento e podem ser acrescentadas livremente pelo programador, de acordo com a necessidade. No exemplo da figura 14 foram suprimidos os comentários que descrevem as seções e foram acrescentados outros comentários.

```
algoritmo "Somar 2 números"
var
    numero1, numero2, soma: inteiro
inicio
    // entrada de dados
    leia (numero1)
    leia (numero2)
    // processamento
    soma := numero1 + numero2
    // saída
    escreva (soma)
finalgoritmo
```

Figura 14 - Algoritmo com comentários

Fonte: o autor (2013)

Descrição: a imagem descreve um algoritmo para receber dois números e mostrar a soma deles. Neste algoritmo foram colocadas linhas de comentários indicando onde inicia cada etapa do processamento. Antes das instruções de entrada dos números, foi colocada uma linha de comentário com o texto entrada de dados. Antes da instrução de calcular a soma dos números, foi colocada uma linha de comentário com o texto processamento. Antes da instrução de exibe o resultado da soma, foi colocada uma linha de comentário com o texto saída.



Os **comentários** são utilizados em algoritmos e também em programas. O objetivo é servir de orientação para o programador, não tendo nenhuma influência na execução do programa. Em pequenos programas, comentários podem ser desnecessários, mas em grandes programas, que podem conter milhares de linhas, eles são muito importantes e até indispensáveis.

Um comentário pode ser utilizado para qualquer propósito como: justificar porque determinada instrução foi utilizada, para informar sobre uma modificação (quem solicitou, a data da solicitação ou se está baseada em alguma legislação).

1.9 Entrada de Dados

Para que o processamento de dados ocorra é necessário que o computador tenha conhecimento dos dados que deverá processar. O conhecimento dos dados é possível pela entrada de dados na memória, ou seja, armazenamento do dado na variável. A entrada de dado ocorre através do comando **leia**.

A sintaxe do comando **leia** é: **leia (<nome_variável>)**

Onde **nome_variável** é o nome da variável que armazenará o dado a ser recebido.

Exemplos:

```
leia (nome)
leia (numero)
```

Este comando pressupõe que o dado virá de alguma fonte, que em nosso caso será do teclado. Quando o programa for executado ocorrerá uma pausa para que o usuário digite o **dado** que ele deseja introduzir. O dado introduzido deverá ser do mesmo tipo declarado para a variável que receberá o dado.

1.10 Saída de Informações

Após o processamento, normalmente, informações são exibidas. Dizemos que esta exibição é a saída do processamento. A saída de informação ocorre através do comando **escreva**.



Tudo que foi escrito no algoritmo, como a declaração das variáveis, por exemplo, não é visualizado pelo usuário durante a sua execução. Apenas o que tiver determinado pelo comando **escreva**. Este comando é a forma que o algoritmo tem para se comunicar com o usuário do algoritmo. Ele pressupõe que a informação será exibida na tela.

A sintaxe do comando escreva é:

escreva ([texto ,] [nome_variável])

Onde:

texto = texto opcional, pois se encontra entre colchetes. Pode ser um texto para descrever a informação que será exibida. Caso tenha optado em colocar esse texto, deve-se colocar uma vírgula para separar do nome da variável.

nome_variável = variável que contém o dado que se deseja exibir.

Exemplos:

```
escreva (numero)
escreva ("Nome do aluno:", nome)
```

1.11 Atribuição de Dados

Um dos processamentos mais básicos é o de atribuir o resultado de uma operação aritmética a uma variável. Na próxima competência estudaremos vários aspectos das operações aritméticas. Por enquanto vamos apenas nos preocupar como o resultado que é guardado para ser posteriormente utilizado em uma saída de informações, por exemplo.

A atribuição é a ação de guardar um dado em uma variável. Já sabemos que o comando **leia** recebe um dado e o guarda em uma variável, mas na atribuição o dado não é recebido da digitação de um usuário via teclado, mas sim conforme a instrução fornecida pelo programador.



A atribuição ocorre através do comando `:=` (dois pontos e igual sem espaço entre eles) ou `<-` (maior que e hífen sem espaço entre os símbolos). Quando encontramos este comando lemos “Recebe”.

Os comandos `:=` / `<-` têm a seguinte sintaxe:

<nome_variável> := / <- <expressão>

Onde:

nome_variável = variável que armazenará o dado resultante da expressão.

expressão = constante, variável ou expressão cujo resultado será armazenado na variável que recebe a atribuição.

Você deve usar um dos comandos por instrução, ou seja, usar `:=` ou `<-` por cada instrução de atribuição. No algoritmo, você pode usar os dois comandos, desde que em linhas diferentes.

Exemplos:

```
numero := 124
nome := "jose"
resposta := numero1 + numero2
```

No primeiro exemplo o programador determinou que a variável `numero` receba por atribuição a constante inteira 124.

No segundo exemplo o programador determinou que a variável `nome` recebesse por atribuição a constante literal `jose`.

No terceiro exemplo o programador determinou que a variável `resposta` recebesse por atribuição o resultado da operação aritmética de adição dos valores armazenados nas variáveis `numero1` e `numero2`.

1.12 Ferramenta para Edição e Teste de Algoritmo

Como o algoritmo é um pseudocódigo, não há como executá-lo para testar se ele está correto.



Tradicionalmente o teste é simulado seguindo as instruções e verificando se há erros. Esse tipo de teste é chamado **teste de mesa**.

Atualmente já existem diversas ferramentas que permitem escrever e testar o algoritmo. Usaremos a ferramenta chamada Visualg. Você irá precisar desta ferramenta para realizar exercícios e testá-los. Você deverá baixá-lo no endereço (<http://ultradownloads.com.br/download/Visualg/>) e fazer sua instalação.

Iniciaremos mostrando a tela do Visualg e algumas de suas funcionalidades. Posteriormente, nas próximas competências, mostraremos outras funcionalidades.

1.12.1 Tela Principal

Sempre que executamos o Visualg a tela abaixo aparece. É um ambiente que segue os padrões do sistema operacional Windows e outros programas da Microsoft (Veja a figura 15).

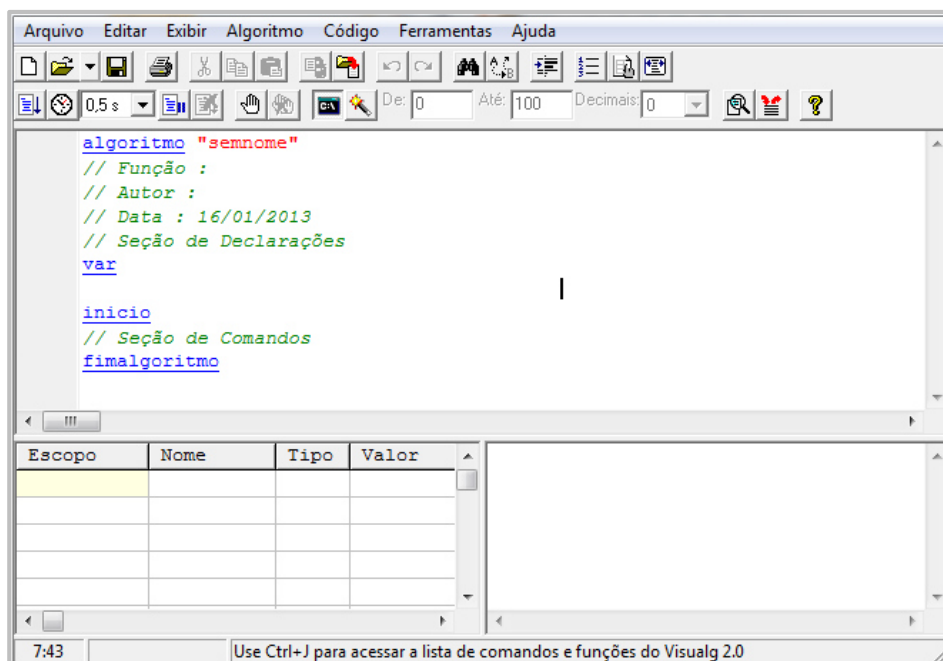


Figura 15 - Tela principal do aplicativo Visualg utilizado para praticar exercícios

Fonte: aplicativo Visualg versão 2.0 (2013)

Descrição: a imagem mostra a tela principal do Visualg versão 2.0, aplicativo que será utilizado para praticar os algoritmos propostos. A tela tem 5 divisões. A primeira divisão fica no topo da tela. A segunda divisão fica abaixo da primeira divisão. A terceira divisão fica abaixo da segunda divisão. A quarta divisão fica abaixo da terceira divisão e à esquerda da tela e a quinta divisão fica abaixo da terceira divisão e à direita da tela. A finalidade de cada divisão será explicada nas imagens a seguir.



Para uma melhor compreensão do conteúdo deste item a partir deste ponto, sugerimos que você assista primeiro ao vídeo postado no link <http://youtu.be/6N-EUkZxJIM>.

Na parte superior temos a barra de menu que dá acesso a todas as funcionalidades disponíveis. As funcionalidades são agrupadas por assunto: Arquivo, Editar, Exibir, Algoritmo, Código, Ferramentas e Ajuda, conforme a figura 16.

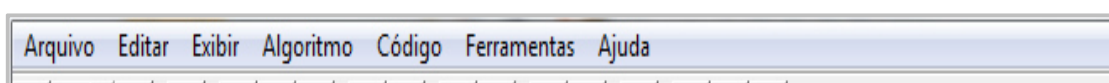


Figura 16 - Barra de menu do Visualg

Fonte: aplicativo Visualg versão 2.0 (2013)

Descrição: a imagem apresenta a primeira divisão do Visualg, denominada de barra de menu, composta de um menu com os grupos de funcionalidades do aplicativo. Cada opção do menu reúne um conjunto de funcionalidade, agrupados por afinidade.

Abaixo da barra de menu temos as barras de ferramentas (figura 17). Nesta barra cada ícone representa uma funcionalidade que também pode ser acessada pela barra de menu, mas são funcionalidades que são muito utilizadas, então são colocadas nesta barra para fornecer um acesso mais rápido. Dependendo da forma como a sua tela esteja configurada, esta barra poderá ser exibida em uma ou em mais linhas.



Figura 17- Barra de menu do aplicativo Visualg

Fonte: aplicativo Visualg versão 2.0 (2013)

Descrição: a imagem apresenta a segunda divisão do Visualg, denominada de barra de ferramentas, composta de botões com as funcionalidades mais utilizadas no aplicativo.

Na parte central do Visualg, encontramos a janela de edição onde devemos digitar o algoritmo (figura 18). A estrutura básica de um algoritmo já vem codificada.

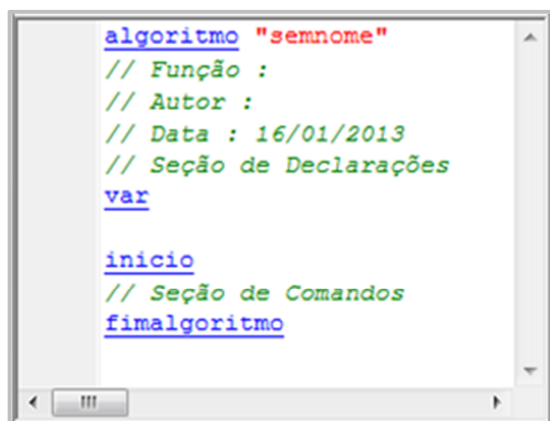


Figura 18 - Janela de edição do Visualg

Fonte: aplicativo Visualg versão 2.0 (2013)

Descrição: terceira divisão do Visualg, denominada de janela de edição. É uma janela onde um algoritmo pode ser criado e modificado.

Na parte inferior à esquerda temos a janela de variáveis para exibir as variáveis com seus respectivos tipos e valores (figura 19).

Escopo	Nome	Tipo	Valor

Figura 19 - Janela de variáveis do Visualg

Fonte: aplicativo Visualg versão 2.0 (2013)

Descrição: quarta divisão do Visualg, denominada de janela de variáveis. Nesta janela são apresentadas as variáveis declaradas no algoritmo, com seus respectivos tipos e, à medida em que o algoritmo é executado, também são exibidos os valores armazenados em cada variável.

Na parte inferior à direita temos a janela em branco, que é a janela de simulação, onde é simulada a execução do algoritmo.

Finalmente temos a barra de status com informações sobre o número da linha e da coluna em que o cursor está posicionado, o estado do algoritmo (em branco ou modificado, se houve alguma modificação no código do algoritmo depois da última vez em que ele foi salvo) e informação para



acessar os comandos predefinidos do visualg (figura 20).

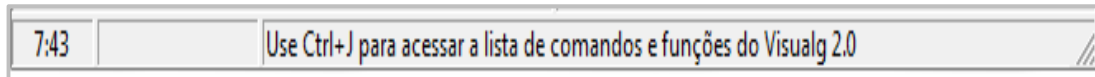


Figura 20 - Barra de status do Visualg

Fonte: aplicativo Visualg versão 2.0 (2013)

Descrição: última linha do Visualg, denominada de barra de status do Visualg. Ela apresenta informações sobre o número da linha e da coluna em que o cursor está posicionado e o estado do algoritmo.



Para dominar melhor as funcionalidades do **Visualg**, recomendamos que você leia a apostila encontrada no endereço: www.slideshare.net/regispires/apostila-sobre-o-visualg-presentation.



Assista ao vídeo postado no link <http://youtu.be/zQtTpt76lqI> para que você possa compreender e praticar o que está sendo ensinado no item a seguir.

1.12.2 Digitando o Algoritmo

Execute o Visualg e vamos digitar nosso primeiro algoritmo com base na figura 13, está preparado? Primeiro, vamos digitar o nome do algoritmo. Na primeira linha, substitua o texto “**semnome**” pelo texto “**Somar 2 números**”. Depois, abaixo do comando var digite a declaração das variáveis numero1, numero2 e soma. Não se esqueça de fazer a indentação (deslocar três posições da margem esquerda).

Agora, na seção de comandos, também endentado, vamos digitar os demais comandos do algoritmo. Para abrir linhas em branco para digitação coloque o cursor no início da linha finalgoritmo e digite várias vezes a tecla ENTER.

O algoritmo digitado deverá ser o seguinte:

```
algoritmo "Somar 2 números"  
// Seção de Declarações
```



```
var
    numero1, numero2, soma: inteiro
// seção de comandos
inicio
    leia (numero1)
    leia (numero2)
    soma := numero1 + numero2
    escreva (soma)
fimalgoritmo
```

Revise com atenção para ter certeza de ter digitado tudo certinho, pois agora iremos mostrar como testar o algoritmo.

Na barra de ferramenta clique no ícone executar, em destaque na figura 21.



Figura 21 - Barra de ferramenta do aplicativo Visualg destacando o botão Executar

Fonte: aplicativo Visualg versão 2.0 (2013)

Descrição: a imagem apresenta a barra de ferramentas do visual, onde o botão executar está sinalizado com um retângulo vermelho. Na configuração da tela apresentada o botão está localizado no início da na segunda linha de botões.

A execução inicia e, como você pode conferir na figura 22, uma janela com fundo preto aparecerá sobre a janela do Visualg. Para entender melhor a execução do algoritmo, arraste esta janela de modo que você *possa enxergar a janela de variáveis e a janela de simulação*.



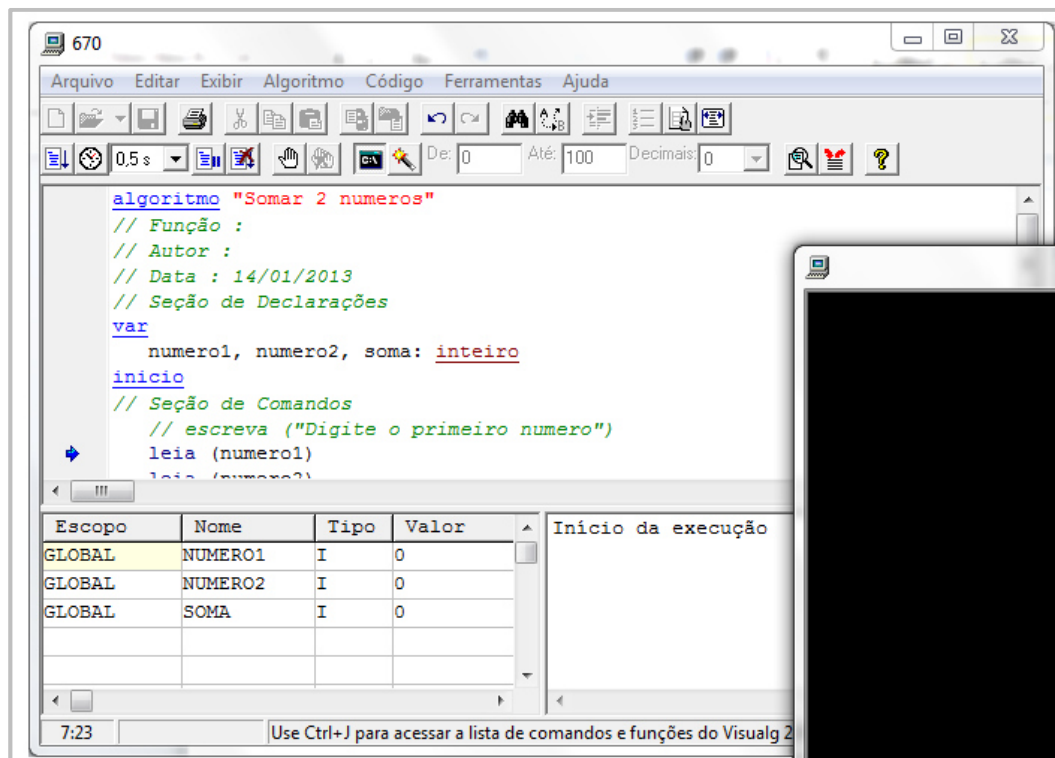



Figura 22 - Tela principal do Visualg mostrando um algoritmo em execução

Fonte: aplicativo Visualg versão 2.0 (2013)

Descrição: tela principal do Visualg onde na janela de edição foi digitado um algoritmo para receber dois números, somá-los e exibir o resultado. Na janela de variáveis aparecem as três variáveis declaradas no algoritmo, que são: numero1, numero2 e soma, com seus respectivos tipos e valores. Como a execução do algoritmo ainda está no início, o valor das variáveis é zero.



Existem dois modos de executar um algoritmo no Visualg.

- O primeiro é baseado no modo texto do sistema operacional DOS. Em uma tela de fundo preto ocorre a execução das instruções em modo texto, sem recurso gráfico.
 - O segundo é baseado no modo gráfico do sistema operacional Windows. A entrada de dados ocorre em uma janela com uma caixa de texto, onde o usuário digita o dado.
- Você pode alternar para qualquer um dos modos clicando no ícone Executar o algoritmo como DOS , na barra de ferramenta.

Observe que a janela de variáveis exibe as três variáveis declaradas, seu tipo (I indicando inteiro) e seus respectivos valores, que até o presente momento é 0 (zero).

A janela de simulação informa que houve o início da execução do algoritmo.

Neste momento a execução está parada aguardando a digitação do primeiro número para ser



armazenado na variável 1, já que a primeira instrução da seção de comandos possui a função de ler um dado de entrada. O computador está aguardando este dado de entrada. Digite um número, por exemplo: 2, e tecla ENTER. Observe que na janela de variáveis o valor da variável **número1** mudou e na janela de simulação o valor digitado aparece abaixo da mensagem de início da execução. Pronto! Você já sabe como acompanhar a execução do algoritmo. Esteja atento a estas duas janelas.

Automaticamente a segunda instrução foi executada. Como também é uma instrução para ler um dado de entrada, nova pausa ocorreu, aguardando que o usuário digite novo dado de entrada. Digite um número, por exemplo: 5, e tecla ENTER. Observe novamente os valores das variáveis na janela de variáveis. Observando a janela de simulação você verifica que a execução do algoritmo já chegou ao final, com o valor da variável soma sendo exibida. **Não houve pausa para executar os comandos de atribuição e de escrita, pois eles não necessitam da intervenção do usuário.** Eles são executados automaticamente pelo simulador.



Assista no vídeo postado no link <http://youtu.be/dG8vhXiwsjE> a execução de um algoritmo em cada um dos modos de execução disponíveis do Visualg.

Feche a janela de execução do DOS para voltar ao Visualg.

Você deve estar pensando: quando houve as pausas para entrada dos dados, eu soube que era para digitar números para cada pausa porque fui eu quem digitou o algoritmo. Mas se o usuário fosse outra pessoa, ela não saberia que dado deveria digitar. Então, como você já aprendeu um comando que escreve na tela, antes de cada comando **leia** você pode acrescentar um comando **escreva** com uma mensagem indicando o dado que deverá ser digitado. Acrescente as duas linhas em destaque exatamente antes de cada comando **leia**, como demonstrado no algoritmo a seguir:

Não se esqueça de que o texto deve estar entre aspas.



Que tal também acrescentar um texto ao resultado final da variável soma? Lembre-se da vírgula para separar o texto da variável.

```
algoritmo "Somar 2 números"  
// Seção de Declarações  
var  
    numero1, numero2, soma: inteiro  
// seção de comandos  
Início  
    escreva ("Digite um número inteiro")  
    leia (numero1)  
    escreva ("Digite outro número inteiro")  
    leia (numero2)  
    soma := numero1 + numero2  
    escreva ("A soma dos números é:", soma)  
fimalgoritmo
```

Execute novamente o algoritmo e veja como ficou mais profissional. A partir de agora, é só você exercer sua criatividade nos exercícios.

ATIVIDADE



As atividades a seguir são para que você possa praticar e aprofundar sua aprendizagem.

1. Escreva um algoritmo para ler quatro números **inteiros** e mostrar a soma deles.
2. Escreva um algoritmo para ler quatro números **reais** e mostrar a soma deles.
Escreva um algoritmo para ler quatro números, sendo dois inteiros e dois reais e mostrar:
 - a. A soma dos números inteiros;
 - b. A soma dos números reais;
 - c. A soma do resultado do item "a" com resultado do item "b".
4. Escreva um algoritmo para ler dois números para as variáveis primeiro e segundo, inverter os valores das variáveis e mostrá-los na tela. Exemplo: se o valor digitado para a variável primeiro for 100 e o valor digitado para a variável segundo for 200, o processamento do algoritmo deverá fazer com que a variável primeiro fique com o valor 200 e a variável segundo fique com 100.



2.Competência 02 | Desenvolver um Algoritmo para a Realização de Operações Matemáticas

No capítulo anterior, aprendemos os princípios de lógica de programação algorítmica, e até já elaboramos alguns algoritmos utilizando a operação matemática de adição. Neste capítulo, aprenderemos a desenvolver algoritmos para realizar diversas operações matemáticas, além da adição.

2.1 Operadores Aritméticos Básicos

Na tabela 1 estão relacionados os operadores aritméticos que são utilizados nas operações básicas da aritmética: adição, subtração, multiplicação e divisão.

OPERADORES	OPERAÇÃO
+	Adição
-	Subtração
*	Multiplicação
/	Divisão

Tabela 1 - Operadores aritméticos básicos
Fonte: o autor (2013)

Nos algoritmos, as expressões matemáticas são escritas de forma linear, como nos exemplos a seguir:

$10+20/2$
 $4*2+10/2$

Observe no próximo algoritmo como realizar as operações matemáticas básicas, utilizando os operadores aritméticos sobre os valores armazenados nas variáveis numero1 e numero2.

```
algoritmo "Operações Matemáticas"  
var  
    numero1: inteiro  
    numero2: inteiro  
    somar: inteiro  
    subtrair: inteiro
```



```
multiplicar: inteiro
dividir: real
// como uma divisão pode ocasionar um
// valor que não é inteiro, a variável
// dividir foi declarada como real.
início
  escreva ("Digite um número inteiro: ")
  leia (numero1)
  escreva ("Digite outro número inteiro: ")
  leia (numero2)
  somar := numero1 + numero2
  subtrair := numero1 - numero2
  multiplicar := numero1 * numero2
  dividir := numero1 / numero2
  escreva ("Soma: ", somar)
  escreva ("Subtração:", subtrair)
  escreva ("Multiplicacao:", multiplicar)
  escreva ("Divisão:", dividir)
fimalgoritmo
```



O exemplo anterior apresenta quatro saídas (quatro comandos escreva). Os resultados serão exibidos na mesma linha. Para uma melhor apresentação podemos utilizar o comando ESCREVAL. Com este comando, cada resultado é apresentado em uma linha.

Sempre que houver um operador de divisão, pode ocorrer um resultado com números decimais, portanto a variável que armazena o resultado da divisão deverá ser do tipo real. Caso a variável seja declarada do tipo inteiro, ocorrerá um erro durante a simulação do algoritmo.

2.2 Prioridade dos Operadores

Seguindo a regra da matemática, as expressões matemáticas são resolvidas dando prioridade aos operadores de multiplicação e divisão. Relembre desta regra a partir dos exemplos que seguem:

Exemplo 1: $10+20/2$

Resulta em 20, pois primeiro é resolvida a divisão e depois a adição.



Exemplo 2: $4*2+10/2$

Resulta em 13, pois primeiro são resolvidas a multiplicação e a divisão e depois a adição.

Para alterar as prioridades é possível utilizar parênteses. Se nos exemplos anteriores a necessidade fosse resolver primeiro as operações de adições, deveríamos ter envolvido a parte da expressão que requer prioridade entre parênteses. Observe nos próximos exemplos como os parênteses mudam o resultado da expressão:

Exemplo 1: $(10+20)/2$

Resulta em 15, pois os parênteses obrigam a resolver primeiro o que estiver dentro dos parênteses, neste caso a adição. Só depois a divisão é resolvida.

Exemplo 2: $4*(2+10)/2$

Resulta em 24, pois primeiro é resolvida a adição priorizada pelos parênteses e depois a multiplicação e divisão.

Para fixar esta regra de forma prática, vamos analisar um algoritmo para calcular a média aritmética de duas notas.

```
1  algoritmo "Calcula Media"
2  var
3  nota1: real
4  nota2: real
5  media: real
6  inicio
7  escreva ("Digite a 1ª nota: ")
8  leia (nota1)
9  escreva ("Digite a 2ª nota: ")
10 leia (nota2)
11 media := (nota1 + nota2) / 2
12 escreva ("A média é: ", media)
13 fimalgoritmo
```



Primeiro ponto a ser observado é que as variáveis foram declaradas do tipo real, pois tanto as notas quanto a média podem ter valores decimais.

Observe que a expressão aritmética que calcula a média na linha 11 dá prioridade para a adição das notas, como realmente deve ser. Só após a adição das notas é que deve ocorrer a divisão por 2.



Assista no vídeo postado no link <http://youtu.be/FdENDhbUoWg> um resumo sobre os assuntos estudados neste item.

2.3 Outros Operadores Aritméticos

Além dos operadores aritméticos básicos já estudados, existem outros operadores aritméticos que são utilizados em operações matemáticas. Na tabela 2 apresentamos dois operadores utilizados em operações matemáticas, que estão representados na figura 23.

OPERADORES	OPERAÇÃO
DIV	Quociente de uma divisão inteira
MOD	Resto de uma divisão inteira

Tabela 2 - Operadores para divisão entre números inteiros

Fonte: o autor (2013)

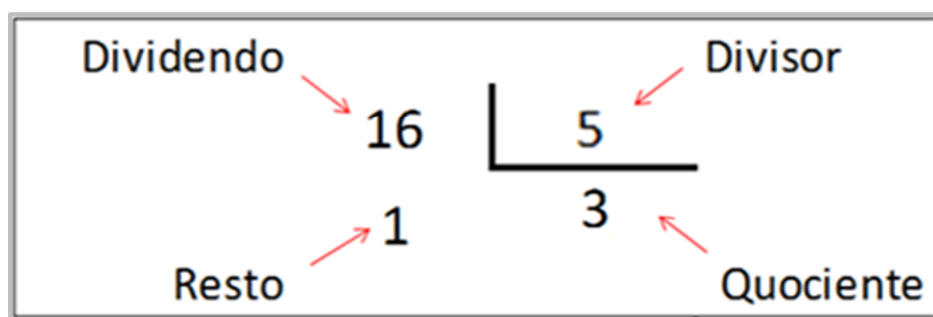


Figura 23- Demonstração de uma divisão inteira entre números inteiros, onde são identificados os elementos envolvidos na divisão: dividendo, divisor, quociente e resto.

Fonte: o autor (2013)

Descrição: a imagem mostra uma divisão inteira entre números inteiros. Os números envolvidos na divisão são 16, colocado no lado esquerdo da imagem, e 5, colocado no lado direito da imagem. Um texto aponta para o número 16 indicando que ele é o dividendo e outro texto aponta para o número 5 indicando que ele é o divisor. Abaixo do número 5 foi colocado o número 3 como resultado da divisão inteira, onde um texto aponta este resultado como o quociente. Abaixo do número 16 foi colocado o número 1 como o número que sobrou da divisão inteira e um texto aponta para ele indicando que ele é o resto.



Considerando-se que na divisão do exemplo mostrado na figura 23 foi utilizado o operador MOD, que pega apenas a parte inteira do quociente, o resultado é 3.



As variáveis que armazenarão os valores de uma operação DIV ou MOD deverão ser do tipo inteiro.

Diferentes dos operadores aritméticos básicos que são símbolos, os operadores apresentados na tabela 02 são palavras reservadas que representam operadores aritméticos. Veja no algoritmo a seguir exemplos de operações matemáticas com esses operadores. Exemplos mais significativos com os operadores MOD e DIV serão explorados quando a próxima competência for abordada.

```
algoritmo "Operadores Aritméticos"
var
    numero1, numero2, resto, quociente: inteiro
inicio
    escreva ("Digite o 1º número: ")
    leia (numero1)
    escreva ("Digite o 2º número: ")
    leia (numero2)
    resto := (numero1 MOD numero2)
    quociente := (numero1 DIV numero2)
    escreval ("Resto da divisão: ", resto)
    escreval ("Quociente da divisão: ", quociente)
finalgoritmo
```

O símbolo % também pode ser utilizado em substituição ao operador MOD, como no exemplo: `resto := (numero1 % numero2).`



Assista no vídeo postado no link <http://youtu.be/LRT7Udjn00Q> o exemplo anterior sendo executado e comentado.



2.4 Funções Aritméticas

Para que operações matemáticas mais complexas possam ser realizadas, algumas rotinas pré-definidas, denominadas **funções**, foram estabelecidas. Os nomes destas funções compõem o conjunto de palavras reservadas da linguagem de programação algorítmica.

Em linhas gerais, a sintaxe de uma função é:

Nome_função (<lista de argumento>): tipo

Onde:

- **Nome_funcao** = é a palavra reservada que representa a função.
- **Lista de argumento** = um ou mais dados necessários para que a função possa processar o seu objetivo.
- **Tipo** = tipo do resultado que a função retorna após o processamento. Toda função retorna (resulta) em um valor que pode ser usado como parte de uma expressão matemática, armazenado ou exibido. O tipo não é escrito no algoritmo, ele apenas representa o tipo do resultado esperado.

Tomemos por exemplo a função **EXP** que executa a operação de **exponenciação**, operação escrita como a^n ("a" elevado a "n", onde "a" é a base e "n" o expoente). Sua sintaxe específica é:

EXP (<base>,<expoente>): real

Neste caso, a lista de argumentos é composta de dois elementos, o primeiro a base e o segundo o expoente. O resultado do processamento desta função em um valor do tipo real:

Exemplos:

```
numero1 := 10  
numero2 := 2
```




```
resultado:= EXP(numero1, numero2)
// resultado da exponenciação sendo
// armazenada em uma variável

resultado:= 2 * EXP(numero1, numero2) + 10
// resultado da exponenciação sendo usada
// como parte de uma expressão matemática
escreva("Resultado: ", EXP(numero1, numero2))
// resultado da exponenciação sendo exibida
// diretamente
```

Como o resultado da função exponenciação é do tipo real, a variável que irá armazenar o resultado de uma exponenciação deverá ser declarada do tipo real.

Na tabela 3 informamos três funções aritméticas que podemos utilizar em nossos próximos exercícios:

FUNÇÃO	OPERAÇÃO	SINTAXE
INT	Converte um valor real em valor inteiro	INT(valor:real): inteiro
RAIZQ	Raiz quadrada	RAIZQ(valor:real): real
QUAD	Quadrado	QUAD(valor:real): real

Tabela 3 - Algumas funções aritméticas
Fonte: o autor (2013)



Assista ao vídeo postado no link <http://youtu.be/nBnaIT6oG-8> onde temos um exemplo de um algoritmo sendo desenvolvido, nele é utilizada a função INT. Não deixe de assisti-lo.

2.5 Tabela de Prioridades

Além das prioridades apresentadas anteriormente entre os operadores aritméticos básicos, os demais operadores e as funções aritméticas também têm prioridades, conforme tabela 4:



PRIORIDADE	OPERADOR / FUNÇÃO
1º	()
2º	Funções
3º	* / DIV MOD
4º	+ -

Tabela 4-Tabela geral de prioridades

Fonte: o autor (2013)

Os parênteses são resolvidos primeiros, depois as funções, seguidas dos operadores de multiplicação, divisão, quociente e resto, e por fim os operadores de adição e subtração.

ATIVIDADE



1. Escreva um algoritmo para ler um número e mostrar:

- A raiz quadrada deste número
- O número recebido elevado a quarta (n^4)
- O número recebido vezes o quociente do número dividido por 2.

2. Escreva um algoritmo para ler o valor do raio de um círculo, calcular e mostrar o valor da sua área.

Obs.: Fórmula para calcular a área de um círculo é: $\pi \times \text{raio}^2$. Considere que o valor de π é 3,14.



3.Competência 03 | Desenvolver um Algoritmo para Resolução de um Problema Utilizando Estrutura de Decisão

Nos algoritmos desenvolvidos nos capítulos anteriores, **todas** as instruções eram executadas sequencialmente, ou seja, **todas eram executadas na ordem em que foram codificadas**. Nem todos os problemas exigem que todas as instruções sejam executadas, como por exemplo: foi codificada uma instrução para solicitar o estado civil de uma pessoa e outra instrução para solicitar o nome do cônjuge (esposo ou esposa). Embora a instrução solicitando o nome do cônjuge tenha de ser codificada no algoritmo - o algoritmo tem que atender a qualquer tipo de pessoa - se a resposta ao estado civil for **solteira**, não faz sentido solicitar o nome do cônjuge. A execução do algoritmo deve “pular” esta instrução.

Este capítulo irá tratar da **estrutura de decisão** que pode ser colocada em um algoritmo, permitindo que haja uma tomada de decisão durante a execução do programa.

3.1 Operação Condicional

Para que haja uma decisão, uma operação condicional precisa ser estabelecida, como no exemplo citado da introdução deste capítulo, onde a condição para solicitar o nome do cônjuge é **ser solteiro**. Na programação, uma operação condicional ocorre quando relacionamos dois elementos.

O relacionamento de dois elementos é feito através de operadores relacionais e sempre resulta em um valor lógico: VERDADEIRO ou FALSO. Na tabela 5 estão listados os operadores relacionais que podem ser usados em uma operação condicional.

OPERADOR	OBJETIVO
=	Igual
<>	Diferente
>	Maior que
<	Menor que
>=	Maior ou igual
<=	Menor ou igual

Tabela 5 - Operadores relacionais
Fonte: o autor (2013)



A partir dos operadores da tabela 5, podemos elaborar operações condicionais, desde que sigamos a seguinte sintaxe:

<elemento_1> <operador_relacional> <elemento_2>

Onde:

operador_relacional é um dos operadores relacionais da tabela 5.

elemento_1 e **elemento_2** são os elementos que queremos comparar.

Qualquer um destes elementos pode ser uma variável, uma constante ou uma expressão matemática.

Exemplos de operações condicionais:

OPERAÇÃO	num1 = num2
COMENTÁRIO	Está sendo comparado se os dados armazenados nas variáveis num1 e num2 são iguais.
RESULTADO	Depende dos valores armazenados em num1 e num2 . Supondo que as variáveis armazenem os valores 10 cada uma, o resultado da comparação é VERDADEIRO. Se a variável num1 armazena o valor 10 e a variável num2 armazena o valor 5, o resultado da comparação é FALSO.

OPERAÇÃO	4 > 6
COMENTÁRIO	Está sendo comparado se a constante numérica 4 é maior que outra constante numérica 6.
RESULTADO	O resultado da comparação é FALSO, pois o número 4 não é maior que o número 6.

OPERAÇÃO	"A" <> "B"
COMENTÁRIO	Como "A" e "B" estão entre aspas, não se tratam de variáveis, mas de constantes literais (texto). Está sendo comparado se a letra "A" maiúscula é diferente da letra "B" maiúscula .
RESULTADO	O resultado da comparação é VERDADEIRO.



OPERAÇÃO	num1 <> 6
COMENTÁRIO	Está sendo comparado se o dado armazenado na variável num1 é diferente da constante numérica 6.
RESULTADO	Depende do tipo da variável num1 e do valor armazenado nela. Considerando que a variável num1 seja numérica (inteiro ou real): se a variável num1 armazena o valor 6, o resultado da comparação é FALSO. Se a variável num1 armazena qualquer outro valor numérico, o resultado da comparação é VERDADEIRO. Considerando que a variável num2 não seja numérica: o resultado é FALSO, pois estão sendo comparados elementos de tipos diferentes. Variável literal ou lógica sendo comparado com uma constante numérica.
OBSERVAÇÃO	O correto é compararmos elementos numerais com numerais, lógico com lógico ou literal com literal. Este exemplo é apenas ilustrativo, pois a maioria das linguagens de programação entende como erro a comparação entre tipos diferentes. O Visualg não está preparado para este tipo de comparação, resultando sempre em FALSO.

OPERAÇÃO	1 = "1"
COMENTÁRIO	O primeiro elemento da comparação é o número 1 e o segundo elemento é o literal "1" (observe que está entre aspas, o que caracteriza ser um literal).
RESULTADO	O resultado da comparação é FALSO, pois estão sendo comparados elementos de tipos diferentes. Veja a observação do exemplo anterior.

OPERAÇÃO	"A" < "a"
COMENTÁRIO	Como "A" e "a" estão entre aspas, não se tratam de variáveis, mas de constantes literais (texto). Está sendo comparado se a letra "A" maiúscula é menor que a letra 'a' minúscula .
RESULTADO	O resultado da comparação é VERDADEIRO.
OBSERVAÇÃO	Aqui você pode ter achado essa comparação estranha, pode ter pensado: "mas como eu posso afirmar que uma letra é menor ou maior que a outra?". A explicação pra isso segue logo abaixo, não deixem de ler o texto do link.



Cada caractere (letra, número ou símbolo) do computador corresponde a um código que está relacionado a uma tabela do computador, denominada Tabela ASCII, que você poderá consultar no link <http://pt.wikipedia.org/wiki/ASCII>, acessado em 21/06/2014. No tópico **Caracteres imprimíveis** da página deste link, as tabelas apresentam na segunda coluna os códigos em números decimais e na quarta coluna a que caractere corresponde. Em uma operação condicional, o computador compara os códigos correspondentes aos caracteres comparados. Por isso, no exemplo acima a constante “A” (código 65) é menor que a constante “a” (código 97).

3.2 Operadores Lógicos

Alguns problemas computacionais exigem que uma operação condicional tenha mais que uma relação. É o caso da obrigatoriedade de votar. A pessoa tem que ter **a idade maior que 17 anos e menor que 66 anos**. Supondo que a idade esteja armazenada na variável **idade**, escrevendo estas condições na sintaxe de uma operação condicional temos:

Condição 1: **idade > 17**

Condição 2: **idade < 66**

Escrevendo-as em linha temos:

(idade > 17) E (idade < 66)

Note que para uma pessoa votar deve atender a duas condições (a primeira condição **E** a segunda condição). Surge então a necessidade de conectarmos duas condições. Nesse caso, a sintaxe da linguagem algorítmica exige que cada condição esteja entre parênteses.

A tabela 6 apresenta dois operadores lógicos de conexão que permitem a elaboração de expressões lógicas mais complexas:



CONECTOR
E
OU

Tabela 6 - Operadores Lógicos
Fonte: o autor (2013)

Como cada condição resulta em um valor lógico (VERDADEIRO ou FALSO), aplicam-se as tabelas 7 e 8, denominadas **Tabela Verdade**, para definir o resultado final da expressão.

E		
RESULTADO DA 1ª CONDIÇÃO	RESULTADO DA 2ª CONDIÇÃO	RESULTADO DA EXPRESSÃO
V	V	V
V	F	F
F	V	F
F	F	F

Tabela 7 - Tabela Verdade E
Fonte: o autor (2013)

OU		
RESULTADO DA 1ª CONDIÇÃO	RESULTADO DA 2ª CONDIÇÃO	RESULTADO DA EXPRESSÃO
V	V	V
V	F	V
F	V	V
F	F	F

Tabela 08 - Tabela Verdade OU
Fonte: o autor (2013)

Exemplos de operações condicionais:

Exemplo 1: para verificar se uma pessoa é obrigada a votar, podemos ter a seguinte expressão:

(idade > 17) E (idade < 66)

Se a primeira condição (idade > 17) for VERDADEIRA E a segunda condição (idade < 66) também for VERDADEIRA, pela tabela verdade do E, o resultado final da expressão é VERDADEIRO. Ainda pela tabela verdade do E, se alguma das condições resultarem em FALSO, o resultado final da expressão é FALSO.



Exemplo 2: Para verificar se uma pessoa não é obrigada a votar, podemos ter a seguinte expressão:

(idade < 18) **OU** (idade > 65)

Se alguma das condições (idade < 18) **OU** (idade > 65) for VERDADEIRA, pela tabela verdade do OU, o resultado final da expressão é VERDADEIRO. Ainda pela tabela verdade do OU, o resultado final da expressão só será FALSO se as duas condições resultarem em FALSO.

Além dos operadores E OU, para conectar condições, há também o operador de negação **NAO**, que inverte o resultado de uma expressão lógica. Este operador é escrito sem o acento “~” na letra “A”.

Para exemplificarmos seu uso vamos considerar uma situação onde teremos como entrada o tipo de ligação telefônica, que pode ser: L (local fixo), C (celular local), E (estadual) ou I (internacional). Sempre que a ligação for do tipo C, E ou I deverá ser solicitada uma senha. A expressão condicional para atender a este critério pode ser assim escrita:

`(Tipo="C") OU (Tipo="E") OU (Tipo="I")`

O uso do operador NAO pode simplificar a expressão da seguinte forma:

`NAO (Tipo="L")`

A expressão lógica fica mais simples utilizando o operador de negação e comparando apenas o único tipo que não exige senha.

3.3 Prioridade dos Operadores Lógicos

Assim como os operadores matemáticos, os operadores lógicos também têm prioridade de execução em uma expressão lógica. A tabela 9 apresenta as prioridades dos operadores lógicos.



OU		
RESULTADO DA 1ª CONDIÇÃO	RESULTADO DA 2ª CONDIÇÃO	RESULTADO DA EXPRESSÃO
V	V	V
V	F	V
F	V	V
F	F	F

Tabela 9 – Prioridade dos operadores lógicos

Fonte: o autor (2013)

Exemplo de uma expressão lógica composta:

$(6=3) \text{ OU } (5<8) \text{ E } (3=3) \text{ OU } (5\leq 4) \text{ E } (3=2)$

Para resolver esta expressão, primeiro resolve-se cada relacionamento da expressão condicional.

Como resultado, temos a seguinte expressão:

falso OU verdadeiro E verdadeiro OU falso E falso

Agora resolvemos primeiro os operadores lógicos E, conforme a tabela verdade, que resulta na seguinte expressão:

falso OU verdadeiro OU falso

Com base na expressão acima resolvemos o 1º operador lógico OU, resultando na expressão abaixo:

verdadeiro OU falso

Por fim resolvemos o último operador OU, e temos o resultado final: **verdadeiro**

Podem-se utilizar parênteses para alterar a prioridade. No exemplo abaixo, o segundo operador OU deverá ser resolvido primeiro, por conta dos parênteses que agrupam os elementos da parte da expressão.

$(6=3) \text{ OU } (5<8) \text{ E } ((3=3) \text{ OU } (5\leq 4)) \text{ E } (3=2)$

Para resolver esta expressão, primeiro resolve-se cada relacionamento da expressão condicional, que resulta na seguinte expressão:



falso OU verdadeiro E (verdadeiro OU falso) E falso

Agora, vamos resolver primeiro a parte da expressão que está dentro de parênteses, conforme a tabela verdade, o que resulta na seguinte expressão:

falso OU verdadeiro E verdadeiro E falso

Em seguida resolvemos o primeiro operador E, resultando na expressão:

falso OU verdadeiro E falso

Vamos resolver agora o segundo E, que resulta na expressão:

falso OU falso

E finalmente, resolvendo o operador OU que resta, chegamos ao resultado final: falso

Para melhor fixação deste assunto, vamos observar alguns exemplos:

Exemplo 1: Considere a situação das pessoas que não são obrigadas a votar. De acordo com a legislação eleitoral, existem dois casos onde o voto é opcional:

Caso 1: Para quem tem mais de 15 anos e menos de 18 anos, que podemos representar com a seguinte expressão lógica:

$(idade > 15) \text{ E } (idade \leq 17)$

Caso 2: Para quem tem mais de 65 anos, que podemos representar com a seguinte expressão lógica:

$idade > 65$

Como no caso 1 OU no caso 2 o voto é opcional, podemos juntar as duas expressões através do operador lógico OU e quaisquer uma das seguintes expressões pode ser escrita:



Expressão 1: **idade>15 E idade<=17 OU idade>65**

Expressão 2: **idade>65 OU idade>15 E idade<=17**

Você deve lembrar-se de que pela tabela 7 de prioridade dos operadores lógicos, independente da ordem em que os operadores lógicos estão colocados na expressão, primeiro deve ser resolvido o operador E (verificando se a idade está na faixa de 16 a 17 anos) para só depois verificar o operador OU.

Se o valor armazenado na variável **idade** for 7 anos, teremos pela primeira expressão:

(7>15) E (7<=17) OU (7>65)

Resolvendo as condições teremos:

falso E verdadeiro OU falso

Resolvendo o operador E (pela tabela 7 - tabela verdade E), **falso E verdadeiro** resulta em **falso**, ficando assim a expressão:

falso OU falso

Por fim, resolvendo o operador OU (pela tabela 8 – tabela verdade OU), **falso OU falso** resulta em **falso**, ou seja, é falso para uma pessoa de 7 anos que o voto é opcional.

Agora faça o teste acima com as idades 16, 30 e 70 anos e verifique como o resultado corresponde à realidade em relação ao que a lei eleitoral estabelece quanto à idade para o voto opcional.

Também resolva a expressão 2 abaixo, com as idades 7, 16, 30 e 70 e comprove que o resultado é o mesmo da expressão 1.

Expressão 2: **(idade>65) OU (idade>15) E (idade<=17)**



Exemplo 2: Agora consideremos a situação onde para ser aprovado o aluno tem que:

Caso 1: ter frequentado pelo menos 75% das aulas, que podemos representar com a seguinte expressão lógica:

$$\text{perc_freq} \geq 75$$

Caso 2: ter obtido o conceito "A" ou "B", que podemos representar com a seguinte expressão lógica:

$$(\text{conc} = \text{"A"}) \text{ OU } (\text{conc} = \text{"B"})$$

No exemplo 1, um dos dois casos deveria ser verdadeiro para que a pessoa pudesse votar de forma opcional, por isso, conectamos as expressões lógicas com o conector E. Na situação deste exemplo, os dois casos devem ser atendidos para que o aluno seja aprovado. Logo, iremos conectar as expressões com o conector E, que pode resultar em uma das expressões:

Expressão 1: $(\text{perc_freq} \geq 75) \text{ E } (\text{conc} = \text{"A"}) \text{ OU } (\text{conc} = \text{"B"})$

Expressão 2: $(\text{conc} = \text{"A"}) \text{ OU } (\text{conc} = \text{"B"}) \text{ E } (\text{perc_freq} \geq 75)$

A princípio você pode achar que as expressões estão corretas, mas há um erro lógico, considerando-se as tabelas verdades 8 e 9.

Vamos resolver as sentenças para entendermos o erro lógico da expressão:

Se o valor armazenado na variável **perc_freq** for 60 e o valor armazenado na variável **conceito** for "A", teremos pela primeira expressão:

$$(\text{60} \geq 75) \text{ E } (\text{"A"} = \text{"A"}) \text{ OU } (\text{"A"} = \text{"B"})$$

Resolvendo as condições teremos:

falso E verdadeiro OU falso



Resolvendo o operador E (pela tabela 7 - tabela verdade E), **falso** E **verdadeiro** resulta em **falso**, ficando assim a expressão:

falso OU **falso**

Por fim, resolvendo o operador OU (pela tabela 8 – tabela verdade OU), **falso** OU **falso** resulta em **falso**, ou seja, é falso para uma pessoa com 60% de frequência e conceito “A” que esteja aprovado.

Também resolva a expressão 2 com os mesmos valores e comprove que o resultado é o mesmo da expressão 1.

Expressão 2: **(conc="A")** OU **(conc="B")** E **(perc_freq>=75)**

Bem, aí você pergunta: “Não vi erro nenhum, cadê o erro lógico de que você falou?”. Realmente, para os dados colocados o erro não aparece. Mas é aí que está o segredo do que estamos estudando – a lógica de programação utilizando algoritmos. Para que um algoritmo esteja correto, ele tem que atender corretamente a situação de todos os alunos que possamos imaginar. Vamos colocar agora um aluno também com 60% de frequência e com o conceito “B”, ficando assim, pela primeira expressão:

(60>=75) E **(“B”=“A”)** OU **(“B”=“B”)**

Resolvendo as condições teremos:

falso E **falso** OU **verdadeiro**

Resolvendo o operador E (pela tabela 7 - tabela verdade E), **falso** E **falso** resulta em **falso**, ficando assim a expressão:

falso OU **verdadeiro**



Resolvendo o operador OU (pela tabela 8 – tabela verdade OU), **falso** OU **verdadeiro** resulta em **verdadeiro**, ou seja, é **verdadeiro** para uma pessoa com 60% de frequência e conceito “B” que esteja aprovado, o que não faz o menor sentido, porque ele tem apenas 60% de frequência. Mesmo se utilizarmos a expressão 2, o resultado também estaria errado.

Esse tipo de erro é o que chamamos de erro lógico, uma vez que a sintaxe da expressão está correta, mas a lógica em relação ao problema que desejamos resolver não está.

Vamos pensar mais sobre o problema?

Para ser aprovado um aluno deve atender aos dois casos seguintes:

Caso 1: `perc_freq >= 75`

Caso 2: `(conc="A") OU (conc="B")`

Como temos dois casos, mas três condições, o computador não sabe se são três casos ou dois casos, combinando a 1ª condição com a 2ª condição ou combinando a 2ª condição com a 3ª condição. Nessa situação, ele obedece à regra de prioridade dos conectores lógicos. Analisando novamente as duas condições vistas anteriormente, o computador estará resolvendo primeiro o conector lógico **E**, que tem mais prioridade que o conector lógico **OU**, conforme destacado nas expressões abaixo:

Expressão 1: `(perc_freq >= 75) E (conc="A") OU (conc="B")`

Expressão 2: `(conc="A") OU (conc="B") E (perc_freq >= 75)`

Para fazer o Caso 2 ser resolvido prioritariamente e só depois ser conectado ao Caso 1, você deve recorrer ao uso dos parênteses, pois ele faz com que a expressão que está dentro dos parênteses seja resolvida primeiro, conforme as expressões a seguir:

Expressão 1: `(perc_freq >= 75) E ((conc="A") OU (conc="B"))`

Expressão 2: `((conc="A") OU (conc="B")) E (perc_freq >= 75)`



Vamos resolver agora expressão 1 com os mesmos valores que serviram para identificarmos o erro lógico: 60% de frequência e conceito “B”, ficando assim, a solução:

$$(60 \geq 75) \text{ E } (("B" = "A") \text{ OU } ("B" = "B"))$$

Resolvendo as condições teremos:

$$\text{falso E (falso OU verdadeiro)}$$

Resolvendo primeiro o operador OU por causa dos parênteses (pela tabela 8 - tabela verdade OU), **falso OU verdadeiro** resulta em **verdadeiro**, ficando assim a expressão:

$$\text{falso E verdadeiro}$$

Resolvendo o operador E (pela tabela 7 – tabela verdade E), **falso E verdadeiro** resulta em **falso**, ou seja, é **falso** para uma pessoa com 60% de frequência e conceito “B” que esteja aprovado.

Dominar a elaboração e a resolução de expressões lógicas é uma habilidade fundamental para a competência que estamos estudando e para poder desenvolver algoritmos com a estrutura de decisão que estudaremos no próximo item.

3.4 Estrutura de Decisão

Durante todos os tópicos anteriores, estivemos fundamentando os conceitos que envolvem a tomada de decisão por parte do computador, via linguagem de programação algorítmica. Agora chegou a hora de conhecermos o que irá suportar a tomada de decisão e alterar o fluxo do programa. Denominamos de **estrutura**, porque não é uma simples instrução de uma linha, mas no mínimo duas linhas que formarão um bloco de instruções.

3.4.1 Estrutura de Decisão Simples

Esta estrutura é utilizada em situações onde uma mudança da sequência normal de execução das

instruções deve ocorrer apenas se a condição avaliada resultar em um valor lógico VERDADEIRO.

Se o resultado da condição avaliada for um valor lógico FALSO, não haverá mudança da sequência e a execução segue seu fluxo normal.

A representação gráfica da figura 24 faz uma comparação com uma avenida. Se em um trecho houver uma obstrução, você pode tomar um desvio e retomar a avenida em um ponto mais a frente, após a obstrução.

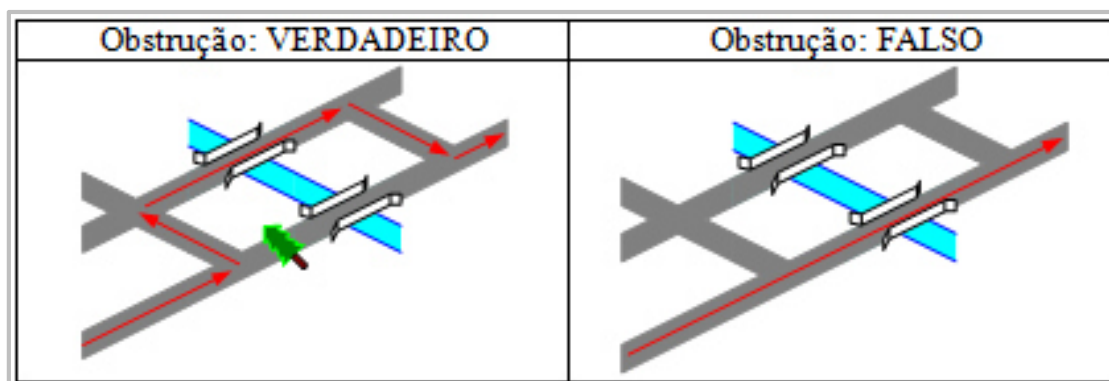


Figura 24 - Representação gráfica de uma estrutura de desvio simples

Fonte: o autor (2008)

Descrição: dois quadros representados de maneira idêntica, com dois caminhos em paralelos que atravessam um rio. Existem pontes sobre o rio. Antes e depois das pontes há uma ligação entre os caminhos. No primeiro quadro há um texto como título informando obstrução: verdadeiro. Uma árvore caída sobre o caminho do lado direito impede o trajeto em frente, sobre a ponte. Setas vermelhas demonstram que para passar para o outro lado da ponte será necessário fazer um desvio e seguir para o caminho do lado oposto, atravessar a ponte deste caminho e depois retornar ao caminho original. No segundo quadro há um texto como título informando obstrução: falso. Não há árvore obstruindo o caminho possibilitando um trajeto sem desvio. Uma seta vermelha demonstra que não será necessário fazer um desvio, sendo possível seguir em frente normalmente.

A sintaxe da estrutura condicional que permite este tipo de desvio é:

SE <expressão_condicional> ENTAO

<bloco_de_instruções>

FIMSE

Denominamos de estrutura, porque não é uma simples instrução de uma linha, mas uma estrutura iniciada com o comando SE e finalizado como o comando FIMSE. Entre os dois comandos podemos colocar diversas instruções, tantas quanto forem necessárias para resolver o problema. A expressão condicional deve atender aos conceitos estudados nos itens anteriores deste capítulo.



Consideremos uma situação onde deverá ser concedido um desconto quando a venda de um produto for maior que quatro unidades. Desenvolvemos o seguinte algoritmo para resolver este problema.

```
algoritmo "Estrutura de Decisão Simples"  
var  
    preco_unit: real  
    quantidade: inteiro  
    perc_desconto: real  
    valor_desconto: real  
    preco_total: real  
inicio  
    escreva("Digite o preço unitário: ")  
    leia(preco_unit)  
    escreva("Digite a quantidade vendida: ")  
    leia(quantidade)  
    // calculando o preço total (sem desconto)  
    preco_total := preco_unit * quantidade  
  
    // Tomando decisão sobre desconto  
    se (quantidade>4) entao  
        escreva("Digite o % de desconto: ")  
        leia(perc_desconto)  
        // Calculando o desconto  
        valor_desconto := preco_total*perc_desconto/100  
        // retirando o desconto do preço total  
        preco_total := preco_total - valor_desconto  
    fimse  
    escreva("Preço Total: ", preco_total)  
fimalgoritmo
```

As instruções colocadas entre os comandos **se** e **fimse** só serão executadas se a expressão lógica do comando **se** resultar em verdadeiro. Nesse caso, será solicitado o percentual do desconto, calculado o valor do desconto e recalculado o preço total. Note que o preço total, sem desconto, já foi calculado antes da estrutura **se**. Dentro da estrutura **se** o preço total apenas está sendo recalculado, abatendo-se o valor do desconto.

3.4.2 Estrutura de Decisão Composta

Esta estrutura é utilizada em situações onde duas mudanças de sequência são previstas. Uma



mudança de sequência é executada se o resultado da condição avaliada for um valor lógico VERDADEIRO e a outra mudança de sequência será executada se o resultado da condição avaliada for um valor lógico FALSO.

Caro aluno, vamos comparar novamente com uma avenida? Observe na figura 25, onde em certo ponto da avenida não dá para seguir em frente, mas se deve tomar uma decisão entre duas alternativas possíveis.

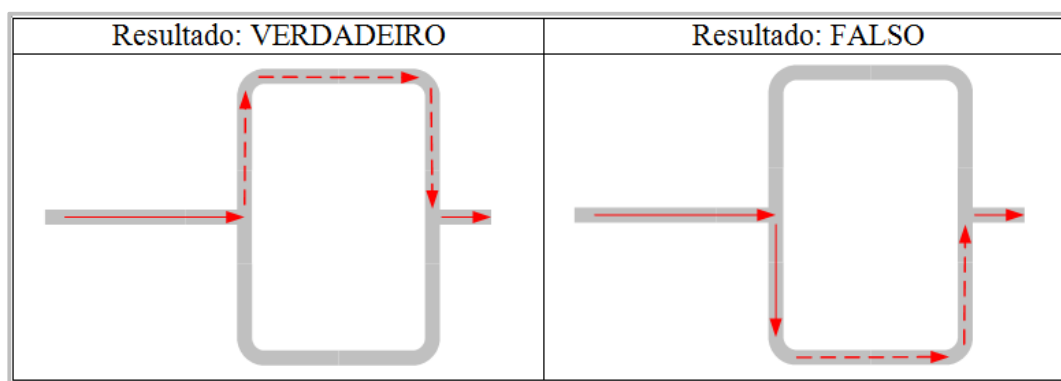


Figura 25 - Representação gráfica de uma estrutura de desvio composta

Fonte: o autor (2008)

Descrição: dois quadros de formas idênticas, indicando o início de um caminho. Em certo ponto do caminho há uma bifurcação, permitindo seguir um trajeto por um caminho à direita ou por um caminho à esquerda. Posteriormente, estes caminhos se encontram formando um único caminho. O primeiro quadro tem um título informando resultado verdadeiro. Uma seta tracejada indica que se o resultado para uma determinada situação for verdadeiro, deve-se tomar o caminho da direita. O segundo quadro tem um título informando resultado falso. Uma seta tracejada indica que se o resultado para uma determinada situação for falso, deve-se tomar o caminho da esquerda.

A sintaxe da estrutura condicional que permite este tipo de desvio é:

```
SE <expressão_condicional> ENTAO
    <bloco_de_instruções_1>
SENAO
    <bloco_de_instruções_2>
FIMSE
```

Esta estrutura permite a definição de dois blocos de comandos. O **bloco_de_instrução_1** a ser executado se o teste da condição resultar em VERDADEIRO. Se não for VERDADEIRO, isto é, for FALSO, o **bloco_de_instrução_2** será executado, tendo início a partir do comando SENAO.

Como aplicação prática desta estrutura, vamos elaborar um algoritmo que receba duas notas,



calcule e mostre a média. Considerando que a média para aprovação seja 6, mostre também se o aluno foi aprovado ou reprovado.

```
algoritmo "Calcula Media"  
var  
    nota1, nota2, media: real  
inicio  
    escreva ("Digite a 1ª nota: ")  
    leia (nota1)  
    escreva ("Digite a 2ª nota: ")  
    leia (nota2)  
  
    media := (nota1 + nota2) / 2  
  
    escreva ("A média é: ", media)  
  
    // Tomando decisão sobre aprovação  
    se (media < 6) entao  
        escreval ("Aluno Reprovado")  
    senao  
        escreval ("Aluno Aprovado")  
    fimse  
fimalgoritmo
```

Entendeu? Após calcular e mostrar a média é preciso informar uma das alternativas: aprovado ou reprovado. Como são duas alternativas colocamos uma **estrutura de decisão composta**, testando se a média é inferior a 6. Se resultar em verdadeiro, será mostrado “Aluno Reprovado”, se não resultar em verdadeiro só pode resultar em falso, neste caso será mostrado “Aluno Aprovado”. Não é legal? É assim que o computador toma decisões.

Agora, vamos desenvolver um algoritmo para informar se um número inteiro é par ou ímpar. Você já pode imaginar que uma estrutura de decisão composta deve ser utilizada porque dois caminhos são possíveis: um caminho se o número for par e outro caminho se o número não for par, ou seja, for ímpar. E como saber se um número é par? Bem, todo número par quando dividido por 2, resta 0. Opa! Lembrou-se do operador MOD, estudado na competência anterior, que mostra o resto de uma divisão inteira? Então, vamos combinar uma estrutura condicional e uma expressão utilizando o operador MOD para resolver o problema.

```
algoritmo "Par ou Impar"  
var
```



```
    numero: inteiro
início
    escreva ("Digite um número inteiro: ")
    leia (numero)

    se numero MOD 2 = 0 então
        escreva ("O número é par")
    senao
        escreva ("O número é impar")
    fimse
fimalgoritmo
```

Muito simples, não foi? Digite este algoritmo no Visualg e faça o teste para conferir .

3.4.3 Estrutura de Decisão Encadeada

Um bloco de comando de uma estrutura condicional pode conter qualquer instrução, inclusive outras estruturas condicionais. Quando isto é necessário, temos uma série de estruturas condicionais, que denominamos de estrutura encadeada ou estrutura aninhada.

Esta estrutura é utilizada em situações onde **mais de duas** mudanças de sequência são previstas. Uma mudança de sequência é executada se o resultado da condição avaliada for um valor lógico VERDADEIRO e a outra mudança de sequência será executada se o resultado da condição avaliada for um valor lógico FALSO, mas dentro de cada uma das duas sequências pode haver outras mudanças.

Para exemplificar esta estrutura, consideremos uma situação onde o aluno pode estar aprovado, em recuperação ou reprovado, conforme as condições a seguir:

Condição para aprovação: média igual ou superior a 6.

Condição para recuperação: média inferior a 6 até 3.

Condição para reprovação: média menor que 3.

```
algoritmo "Calcula Media"
var
    nota1, nota2, media: real
início
```



```
escreva ("Digite a 1ª nota: ")
leia (nota1)
escreva ("Digite a 2ª nota: ")
leia (nota2)
media := (nota1 + nota2) / 2
escreval ("A média é: ", media)
se (media < 3) entao
    escreval ("Reprovado")
senao
    se (media < 6) entao
        escreval ("Recuperação")
    senao
        escreval ("Aprovado")
fimse
fimalgoritmo
```

ATIVIDADE

As atividades a seguir são para que você possa praticar e aprofundar sua aprendizagem.

1. Escreva um algoritmo para ler o salário de dois funcionários. Os valores dos salários deverão ser diferentes e poderão ser informados em qualquer ordem: primeiro o menor salário depois o maior salário ou vice-versa. O algoritmo deverá executar um processamento para que o maior salário fique armazenado na 1ª variável declarada e o menor salário fique armazenado na 2ª variável declarada. O algoritmo deverá mostrar os salários em ordem do menor para o maior.

2. O ingresso para um show promovido pelo clube social da cidade tem o valor padrão de R\$ 70,00. Escreva um algoritmo para ler o tipo de expectador que assistirá ao show ("S" para sócio e "N" para não sócio). Deverá ser exibido o valor que deverá ser pago pelo ingresso.

Obs.: Sócio paga 50% do valor padrão do ingresso.

3. Escreva um algoritmo para ler:

- a. O preço normal do ingresso de um show
- b. A quantidade total de ingressos vendidos
- c. A quantidade de ingressos meia-entrada vendidos.

Calcule e mostre o valor total arrecadado, considerando que o ingresso de meia-entrada custa metade do preço normal do ingresso.

4. Escreva um algoritmo para ler dois números e mostrar de acordo com os valores dos números:

- a. O primeiro número é menor que o segundo
- b. O primeiro número é maior que o segundo
- c. Os números são iguais

5. Escreva um algoritmo para ler o valor da uma conta de restaurante, a quantidade de homens, a quantidade de mulheres e a quantidade de crianças que participaram da refeição. Calcule e mostre o valor que cada grupo deve pagar, considerando que mulheres e crianças

pagam a metade do valor que os homens pagam.





4. Competência 04 | Desenvolver um Algoritmo para Resolução de um Problema Utilizando Estrutura de Repetição

Além das estruturas estudadas no capítulo anterior, que permitem diversas sequências de instruções em um algoritmo, existem várias estruturas que permitem a repetição de uma sequência de instruções. Esses tipos de estruturas são úteis em diversas situações, como por exemplo: crítica de dados, onde uma instrução de entrada de dados é repetida, até que um valor válido seja digitado. Nesta competência serão estudados três tipos de estruturas de repetição e você entenderá quando utilizar cada uma delas.

4.1 Estrutura de Repetição Indefinida, com uma Repetição Obrigatória

Em situações onde não sabemos quantas vezes uma sequência de instruções deve ser repetida, você deve utilizar uma estrutura de repetição indefinida. Existem duas estruturas de repetição indefinida. Uma em que o bloco de repetição ocorre pelo menos uma vez e outra em que o bloco de repetição pode não ocorrer nenhuma vez.

Para repetir pelo menos uma vez, podemos utilizar a estrutura REPITA, cuja sintaxe é a seguinte:

REPITA

<bloco_de_instruções>

ATE <expressão_condicional>

Quando a execução do algoritmo atinge o comando REPITA, ele entende que começou um bloco de instruções que pode se repetir. Esse bloco estende-se até o comando ATE. O bloco é executado pela primeira vez, ou seja, ocorre a primeira repetição. Ao atingir o comando ATE, a expressão condicional contida neste comando é avaliada. Se o resultado da expressão condicional for FALSO ocorrerá nova repetição do bloco de instrução. A cada final de repetição a expressão condicional é avaliada. Quando acontecer do resultado da expressão condicional ser VERDADEIRO, a repetição para e a execução do algoritmo continua sequencialmente.



Para compreender melhor o conceito desta repetição, observe o seguinte pseudocódigo:

```
Instrução_1
Instrução_2
Instrução_3
REPITA
    Instrução_4
    Instrução_5
ATE condição
Instrução_6
Instrução_7
```

As instruções 1, 2 e 3 serão executadas sequencialmente no início da execução do algoritmo.

As instruções 4 e 5, que fazem parte de um bloco de repetição, serão executadas uma vez. Como a repetição é definida pelo comando REPITA, cuja condição de repetição é codificada no final do bloco, no comando ATE, a possibilidade de repetição depende da avaliação desta condição. Se o resultado for FALSO, as instruções 4 e 5 serão executadas novamente, ou seja, ocorrerá nova repetição destes comandos. Quando o resultado da condição for VERDADEIRO, a repetição termina e serão executados os comandos 6 e 7.

Vamos considerar o algoritmo que calcula a média aritmética entre duas notas. O valor máximo de cada nota só pode ser 10. Caso o usuário digite um valor maior que 10, o valor não deve ser aceito e a instrução de entrada deve se repetir até que um valor menor ou igual a 10 seja digitado. Nosso algoritmo deve ficar da seguinte forma:

```
algoritmo "Calcula Media"
var
    nota1, nota2, media: real
inicio
    repita
        escreva ("Digite a 1ª nota: ")
        leia (nota1)
    ate (nota1<=10)
    repita
        escreva ("Digite a 2ª nota: ")
        leia (nota2)
    ate (nota2<=10)
```



```
media := (nota1 + nota2) / 2
escreva ("A média é: ", media)
finalgoritmo
```



Assista ao vídeo postado no link <http://youtu.be/9xCtVeIJWXk> e veja como melhorar ainda mais a interação do algoritmo com o usuário.

4.2 Contador

Com o uso de estrutura de repetição surge a necessidade de realizarmos algumas contagens como, por exemplo:

- A quantidade de vezes em que a repetição ocorreu;
- Se a repetição foi por erro, quantas vezes o usuário errou;
- Se no bloco de repetição foi solicitado o sexo de uma pessoa, quantas são do sexo masculino e quantas são do sexo feminino.

Um contador é uma variável que, a partir de um valor inicial, a cada repetição pode ter o seu valor aumentado ou diminuído de **um valor constante**.

A sintaxe de um contador é:

<contador> := <contador> + / - <constante_numerica>

Vamos desenvolver um algoritmo para receber o sexo de **várias** pessoas. Deverá ser digitado M para masculino e F para feminino. Para finalizar o algoritmo deverá ser digitado o sexo X. Ao final da execução o algoritmo deverá exibir a quantidade de pessoas do sexo masculino e a quantidade de pessoas do sexo feminino.

```
algoritmo "Contador"
var
    sexo: literal
```




```
masculino, feminino: inteiro
inicio
// Iniciando o valor dos contadores com zero
masculino := 0
feminino := 0
repita
    escreva ("Digite o sexo (M/F): ")
    leia (sexo)
    se sexo="M" entao
        masculino := masculino + 1
    senao
        se sexo="F" entao
            feminino := feminino + 1
        fimse
    fimse
ate (sexo="X")
escreval ("Masculino(s): ", masculino)
escreval ("Feminino(s): ", feminino)
finalgoritmo
```

Como a contagem é realizada a partir de um valor inicial, inicializamos as variáveis contadoras **masculino** e **feminino** com o valor zero no início da seção de comandos.

Nesse algoritmo, toda vez em que é digitado a letra “M” para uma leitura de um sexo, é adicionado no valor da constante numérica 1 a variável contadora **masculino**. E toda vez em que é digitada a letra “F” para uma leitura de um sexo, é adicionado o valor da constante numérica 1 a variável contadora **feminino**.

Observe a condição colocada no comando ATE. Esta condição (sexo = “X”) que determina até quando a repetição ocorrerá.

4.3 Acumulador

O conceito de acumulador é muito parecido com o conceito de contador, sendo que o acumulador, a partir de um valor inicial, pode aumentar ou diminuir um **valor variável** a cada repetição, cuja sintaxe é a seguinte:

<acumulador> := <acumulador> + / - <variável_numerica>



Vamos desenvolver um algoritmo para receber o preço de **vários** produtos. Para finalizar o algoritmo deverá ser digitado o preço 0 (zero). No final será exibido o valor total dos produtos.

```
algoritmo "Acumulador"
var
    preco, total: real
início
    // Iniciando o valor do acumulador com zero
    total := 0
    repita
        escreva ("Digite o preço: ")
        leia (preco)
        total := total + preco
    ate (preco=0)
    escreval ("Valor total: ", total)
fimalgoritmo
```

Neste algoritmo, o valor do preço digitado a cada repetição é adicionado ao acumulador **total**, até que seja digitado um preço 0.

Como o acúmulo é realizado a partir de um valor inicial, inicializamos a variável acumuladora **total** com o valor zero no início da seção de comandos.

4.4 Estrutura de Repetição Indefinida, com Repetição Opcional

Para situação onde a repetição de um bloco de comando, ainda que seja a primeira repetição, depende de uma condição inicial, você deve utilizar a estrutura ENQUANTO, cuja condição fica no início da estrutura e tem a seguinte sintaxe:

```
ENQUANTO <expressão_condicional> FACA
    <bloco_de_instruções>
FIMENQUANTO
```

Você notou a diferença entre a estrutura REPITA e a estrutura ENQUANTO? A expressão condicional nesta estrutura é colocada no início do bloco que se deseja repetir.

Quando a execução do algoritmo encontra a condição contida no comando ENQUANTO, a condição



é avaliada. Se o resultado for VERDADEIRO, o bloco de comando entre o comando ENQUANTO e o comando FIMENQUANTO é executado e a execução retorna ao comando ENQUANTO que testa a condição novamente. Se o resultado da condição for FALSO, o algoritmo “pula” o bloco de comando e segue executando os comandos após o comando FIMENQUANTO.

Vamos utilizar o mesmo pseudocódigo utilizado na estrutura de repetição anterior para compreender melhor o conceito desta repetição.

```
Instrução_1
Instrução_2
Instrução_3
ENQUANTO condição FACA
    Instrução_4
    Instrução_5
FIMENQUANTO
Instrução_6
Instrução_7
```

As instruções 1, 2 e 3 serão executadas sequencialmente no início da execução do algoritmo.

Ao atingir o comando ENQUANTO a condição é avaliada. Se o resultado for FALSO, a execução continua nas instruções 6 e 7. Se o resultado for VERDADEIRO, as instruções 4 e 5 são executadas, o fluxo de execução retorna ao comando ENQUANTO para avaliar a condição novamente e julgar se repetirá as instruções 4 e 5 mais uma vez ou encerrar a repetição e executar os comandos 5 e 6.

Vamos elaborar um algoritmo para solicitar a quantidade de filhos de um casal. Para cada filho deverá ser solicitada a idade. No final do algoritmo deverá ser mostrada a média de idades dos filhos do casal.

```
algoritmo "Repetição"
var
    quant_filhos, idade: inteiro
    contador, acumulador: inteiro
    media: real
inicio
    escreva("Digite a quantidade de filhos: ")
    leia (quant_filhos)
```



```
// Iniciando contador e acumulador com zero
contador := 0
acumulador := 0
enquanto contador < quant_filhos faca
    contador := contador + 1
    escreva ("Idade do ", contador, " filho: ")
    leia (idade)
    acumulador := acumulador + idade
fimenquanto
media:=0
se quant_filhos <> 0 entao
    media:= acumulador / quant_filhos
fimse
escreva ("Média das idades: ", media)
fimalgoritmo
```



Assista no vídeo postado no link <http://youtu.be/SE7Tu1UE3vQ> a execução comentada deste algoritmo.

4.5 Estrutura com Repetição Definida

Em situações onde a quantidade de repetição já é conhecida, podemos utilizar a estrutura de repetição PARA, que tem a seguinte sintaxe:

PARA <contador> DE <valor_inicial> ATE <valor_final> FAÇA
 <bloco_de_instruções>
FIMPARA

Onde:

Contador: é uma variável contadora.

Valor_inicial: é o valor que inicializará o contador automaticamente.

Valor_final: é o valor que, quando atingido pelo contador, encerrará a repetição.

A cada repetição realizada o valor da variável controladora é incrementado automaticamente em 1.



Vamos desenvolver um algoritmo para informar a temperatura média de cada dia da semana. No final do algoritmo deverá ser mostrada a temperatura média da semana.

```
algoritmo "Repetição"
var
    dia: inteiro
    temp_dia, temp_semana, media: real
inicio
    // inicializado o acumulador com zero
    temp_semana := 0
    para dia de 1 ate 7 faca
        escreva ("Temperatura do ", dia, "º dia: ")
        leia (temp_dia)
        temp_semana := temp_semana + temp_dia
    fimpara
    media:= temp_semana / 7
    escreva ("Temp. média da semana: ", media)
finalgoritmo
```



A execução deste algoritmo comentada, você pode assistir no link <http://youtu.be/YWdrW9xQ8Ec>

ATIVIDADE

As atividades a seguir são para que você possa praticar e aprofundar sua aprendizagem da competência 4.

1. Escreva um algoritmo para ler as duas notas de 10 alunos, calcular a média aritmética de cada aluno e mostrar a quantidade de alunos aprovados e a quantidade de alunos reprovados.

Considere que a média para aprovação é 6.

2. Escreva um algoritmo para ler o estado civil de 10 participantes de um curso. O algoritmo deve criticar para que seja aceito apenas as letras "C", "c", "S" e "s" como entrada para o estado civil. No final o algoritmo deve mostrar se houve mais participantes solteiros, mais participantes casados ou se a quantidade de participantes solteiros e casados são iguais.

3. Escreva um algoritmo para ler o nome, o estado civil e a quantidade de dependentes de quatro funcionários. Se houver dependentes, para cada dependente deverá ser lida a idade e o tipo de dependente que pode ser:

C para cônjuge ou

X para outro tipo de dependente

O algoritmo deverá mostrar o valor total do benefício que cada funcionário deve receber, considerando-se que:

- a) Para o funcionário é pago um valor de R\$ 40,00.
- b) Para o cônjuge de funcionário casado é pago um valor de R\$ 20,00.
- c) Para outro tipo de dependente com idade de até 17 anos é pago o valor de R\$ 15,00.





Conclusão

Prezado (a) aluno (a), finalmente concluímos as quatro competências previstas para esta disciplina, onde tivemos a oportunidade de aprender como programar um computador.

Iniciamos pelos princípios da lógica de programação algorítmica, conhecendo sobre dados e informações e linguagem de programação. Também vimos como ocorre a entrada dos dados, como os dados processados são atribuídos a variáveis e como as informações geradas são exibidas aos usuários. Foi nesta primeira etapa que tivemos contato com o Visualg – ferramenta para edição e execução de algoritmo.

Consolidados os princípios da lógica de programação algorítmica, partimos para uma etapa onde tivemos conhecimento sobre vários operadores e funções que muito nos ajudam a desenvolver algoritmos para realização de operações matemática.

Etapa a etapa, formos aprimorando nosso conhecimento e tivemos contato com as estruturas de decisão, lembram-se? Estrutura de decisão simples, composta e encadeada? Agora sim, nossos algoritmos podem fazer comparações e decidir por qual caminho seguir.

Finalizamos esta disciplina estudando as estruturas de repetição, onde foi possível desenvolver algoritmos mais elaborados utilizando contadores e acumuladores.

Espero que você tenha aprendido bem todas as etapas, pois são fundamentais para as disciplinas Linguagem de Programação para Web, Orientação a Objetos e Projeto de Desenvolvimento de Software. É importante que você busque aprofundar e complementar seus conhecimentos, para que possa se tornar um profissional bem qualificado e consequentemente bem sucedido.



Referências

FARRER, Harry *et al.* Algoritmos Estruturados. 3ª ed.: LTC, 2011

MANZANO, José Augusto; OLIVEIRA, Jayr Figueiredo de. Estudo Dirigido de Algoritmos. 1ª ed.: ERICA, 1997

Minicurrículo do Professor



- **Aldo de Moura Lima**

Possui graduação em Análise e Desenvolvimento de Sistemas pela Faculdade de Tecnologia Ibratec e especialização em Produção de Software, com ênfase em software livre, pela Universidade Federal de Lavras-MG. Atualmente é professor universitário da Faculdade de Tecnologia Ibratec.

Possui experiência de ensino de quase 30 anos em cursos de qualificação profissional, técnico, graduação e pós-graduação, em disciplinas de Lógica de Programação, Linguagens de Programação, Banco de Dados, Gerência de Configuração, Engenharia de Software, Sistemas de Informação e Metodologias de Desenvolvimento de Sistemas.

Elaborou as apostilas Técnicas de Programação (2002) e Lógica de Programação (2008) e desenvolveu diversos sistemas para área comercial, industrial e de serviços para empresas nos estados de Pernambuco, João Pessoa, Rio Grande do Norte e Ceará.

