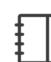
Lógica de Programação

Lógica de Programação - Versão 2

Nome: ______________________________

Lógica de programação



Quantidade de Aulas
14 aulas

Carga horária
21 horas

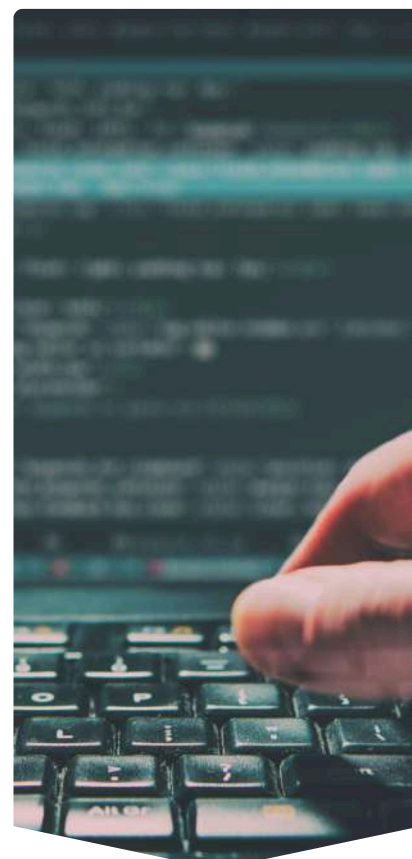
Programas Utilizados
Visualg (Versão 2)

Sobre o curso

A lógica de programação é uma linguagem usada para criar um programa de computador. A lógica de programação é essencial para desenvolver programas e sistemas informáticos, pois ela define o encadeamento lógico para esse desenvolvimento.

O que aprender com este curso?

Aprender lógica de programação é o primeiro passo para se tornar um grande programador web porque isso não muda, independente da linguagem que você vier a escolher posteriormente a se especializar. Consiste em aprender técnicas para escrever códigos que possam ser interpretados por computadores.



Sumário

1 - Introdução a programação
1.1 - Lógica Básica
1.2 - Lógica Aristotélica
1.3 - Lógica de Argumentação
1.4 - Raciocínio Lógico
1.5 - Tabela Verdade
1.6 - Exercícios Passo a Passo
1.7 - Exercícios de Fixação
2 - Variáveis, constantes e tipos de dados
2.1 - Tipos de Dados
2.2 - Tipo de dados primitivos
2.3 - Tipo de dados customizados
2.4 - Exercícios Passo a Passo
2.5 - Exercícios de Fixação
3 - Primeiro programa (algoritmos)
3.1 - Algoritmos não estruturados
3.2 - Algoritmos estruturados
3.3 - Algoritmos no Facebook
3.4 - Algoritmos no Google
3.5 - Algoritmos no Instagram
3.6 - Algoritmos no Spotify
3.7 - Representações de um Algoritmo
3.8 - Fluxograma
3.9 - Linguagem Natural
3.10 - Linguagem de Programação
3.11 - Pseudocódigo
3.12 - VisuAlg
3.13 - Exercícios Passo a Passo
3.14 - Exercícios de Fixação
4 - Tipos de operadores
4.1 - Operadores Aritméticos
4.2 - Precedência entre Operadores Aritméticos
4.3 - Operadores Relacionais
4.4 - Operadores Lógicos
4.5 - Tipos de Dados Lógicos
4.6 - Operador E (AND)
4.7 - Operador OU (OR)
4.8 - Operador NÂO (NOT)
4.9 - Operador NÂO-E (NAND)
4.10 - Operador NÂO-OU (NOR)
4.11 - Operador OU-EXCLUSIVO (XOR)
4.12 - Operador NÂO-OU-EXCLUSIVO (XNOR)
4.13 - Exercícios Passo a Passo
4.14 - Exercícios de Fixação
5 - Estrutura de decisão parte 1
5.1 - Estrutura de Decisão Simples (Se...Então)
5.2 - Estrutura de Decisão Composta (Se...Então...Senão)
5.3 - Exercícios Passo a Passo
5.4 - Exercícios de Fixação
6 - Estrutura de decisão parte 2
6.1 - Escolha-Caso
6.2 - OUTROCASO
6.3 - Exercícios Passo a Passo
6.4 - Exercícios de Fixação
7 - Estrutura de repetição parte 1
7.1 - ENQUANTO-FAÇA
7.2 - Exercícios Passo a Passo
7.3 - Exercícios de Fixação
8 - Estrutura de repetição parte 2
8.1 - REPITA-ATÉ
8.2 - PARA-FAÇA
8.3 - Exercícios Passo a Passo
8.4 - Exercícios de Fixação
9 - Manipulação de vetores
9.1 - VETOR
9.2 - Exercícios Passo a Passo
9.3 - Exercícios de Fixação
10 - Manipulação de matrizes
10.1 - Matriz
10.2 - Exercícios Passo a Passo
10.3 - Exercícios de Fixação
11 - Funções e procedimentos
11.1 - Funções
11.2 - Procedimentos
11.3 - Variáveis Globais e Locais
11.4 - Funções Pré-Definidas
11.5 - Funções Numéricas, Algébricas e Trigonométricas
11.6 - Funções para manipulação de cadeias de caracteres (Strings)
11.7 - Criando Funções
11.8 - Passagens de Parâmetros em Procedimentos
11.9 - Exercícios Passo a Passo
11.10 - Exercícios de Fixação
12 - Modularização
12.1 - Modularização ou Sub-rotinas

12.2 - Escopo de Dados e Códigos
12.3 - Alguns comandos no VisuAlg
12.4 - Parâmetros Interrompa
12.5 - Comando Aleatório
12.6 - Comando Arquivo
12.7 - Comando Timer
12.8 - Comandos de Depuração
12.9 - Comando Pausa
12.10 - Comando Debug
12.11 - Comando Eco
12.12 - Comando Cronômetro
12.13 - Comando LimpaTela
12.14 - Exercícios Passo a Passo
12.15 - Exercícios de Fixação
13 - Prática 1
13.1 - Exercícios Passo a Passo
13.2 - Exercícios de Fixação
14 - Prática 2
14.1 - Exercícios Passo a Passo
14.2 - Exercícios de Fixação

1. Introdução a programação

Steve Jobs disse uma vez: "Todo mundo deveria aprender a programar um computador porque isso te ensina a pensar". A afirmação do fundador do império da maçã logo foi confirmada pela ciência, e cada vez mais estudos mostram que aprender a programar desenvolve diversas habilidades cognitivas, principalmente em crianças.

Diferente do que muita gente pensa, você não precisa ser um gênio para aprender a programar. Lembra de quando você não sabia ler? As letras eram como desenhos ou rabiscos e pra você não formavam palavras, muito menos frases. Mas, aos poucos você aprendeu, primeiro as vogais, depois as consoantes e então vieram as sílabas, palavras e por fim você estava lendo e escrevendo. A programação também é uma linguagem, que pode ser lida por computadores, e você não precisa ser nenhum gênio para aprendê-la, assim como não precisa ser um gênio para aprender a ler e escrever.

Segundo o Centro para Crianças e Tecnologia, jovens e adultos também se beneficiam desse aprendizado: quem aprende a programar alcança um desempenho cerca de 16% superior nos estudos do que os demais estudantes. Os mais velhos também saem ganhando, porque a lógica de programação ajuda a blindar o cérebro contra doenças cognitivas muito comuns com o avançar da idade.

Quem já estudou outros idiomas sabe o quanto é difícil se tornar fluente. Você precisa se dedicar, estudar, fazer exercícios e praticar. Com a programação funciona da mesma forma, ninguém decide aprender a programar e no dia seguinte já sabe tudo sobre códigos. Lembra como foi com o seu idioma nativo? Primeiro você aprendeu as letras, passou a reconhecê-las, depois começou a fazer combinações e quando viu já estava lendo e escrevendo.

Aprender programação também é assim. Primeiro você vai aprender uma linguagem de programação, aos poucos começará a escrever códigos e então poderá começar a criar coisas. E, o fato de saber programar também não significa que você terá que trabalhar com isso. Mas, certamente poderá aplicar as habilidades em várias outras situações do dia a dia. Inclusive, entender a lógica da programação é interessante para várias outras profissões.

Aprender a programar possui diversos benefícios, vamos citar alguns deles:

1. Desenvolvimento do pensamento crítico

Os códigos ajudam na habilidade de se fazer suposições, ao mesmo tempo que desenvolvem o raciocínio preciso e específico. Essas vantagens vêm da necessidade das máquinas de executar algoritmos muito específicos, exigindo precisão de quem os executa.

2. Resolução de desafios

A programação traz facilidade de lidar com problemas e inventar soluções. Torna seus desenvolvedores capacitados na arte de inventar, de fazer descobertas aplicáveis a qualquer tipo de resolução de desafios.

3. Raciocínio múltiplo

Quando se constrói um código ou um algoritmo, dificilmente existe "o melhor". Por isso, ao programar, desenvolvemos a capacidade de decidir, entre várias opções, quais têm custos e benefícios mais vantajosos para seus objetivos.

4. Lógica Matemática

Graças aos códigos e algoritmos, a programação estimula o contato prático com a matemática através de atividades exploratórias e contextualização para resolução de problemas.

1.1. Lógica Básica

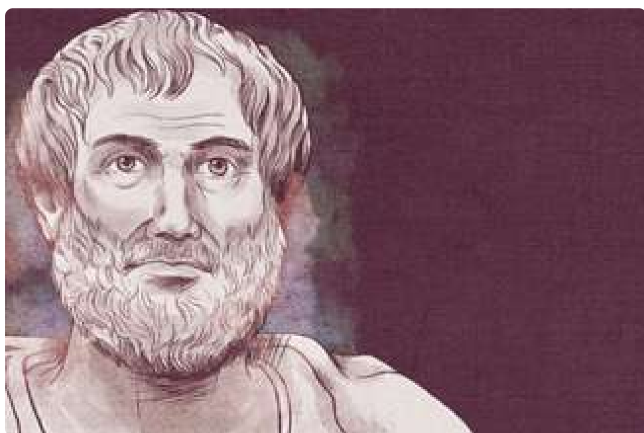
Lógica é um substantivo feminino com origem no termo grego logiké, relacionado com o logos, razão, palavra ou discurso, que significa a ciência do raciocínio.

Em sentido figurado, a palavra lógica está relacionada com uma maneira específica de raciocinar, de forma acertada. Por exemplo: Isso nunca vai funcionar! O teu plano não tem lógica nenhuma!

1.2. Lógica Aristotélica

Para Aristóteles, a lógica não é ciência e sim um instrumento (órganon) para o correto pensar. O objeto da lógica é o silogismo.

Silogismo nada mais é do que um argumento constituído de proposições das quais se infere (extrai) uma conclusão. Assim, não se trata de conferir valor de verdade ou falsidade às proposições (frases ou premissas dadas) nem à conclusão, mas apenas de observar a forma como foi constituído. É um raciocínio mediado que fornece o conhecimento de uma coisa a partir de outras coisas (buscando, pois, sua causa).



“A lógica aristotélica baseia-se no pressuposto de que a razão humana é capaz de deduzir conclusões a partir de afirmações ou negações anteriores. Se as premissas forem verdadeiras, as conclusões também serão”, diz o filósofo Carlos Matheus, da PUC-SP.

1.3. Lógica de Argumentação

A lógica de argumentação permite verificar a validade ou se um enunciado é verdadeiro ou não. Não é feito com conceitos relativos nem subjetivos. São proposições tangíveis cuja validade pode ser verificada. Neste caso, a lógica tem como objetivo avaliar a forma das proposições e não o conteúdo. Os silogismos (compostos por duas premissas e uma conclusão), são um exemplo de lógica de argumentação. Por exemplo:

O Fubá é um cachorro.

Todos os cachorros são mamíferos.

Logo, o Fubá é um mamífero.

Pode-se pensar que o raciocínio lógico não é utilizado para determinadas situações no cotidiano. No entanto, todas as pessoas utilizam o raciocínio lógico: seja cientista, engenheiro, psicólogo, advogado, homem de negócios, arquitetos, serralheiros e você.

Muitas vezes é usado de modo mais informal, outras vezes de modo a ter um conhecimento apurado do mesmo e saber aplicar em momentos que pareçam necessários.

1.4. Raciocínio Lógico

Para um raciocínio lógico que converge em atitudes inteligentes no cotidiano, proporcionando um dia agradável e organizado, primeiro faz-se necessário o saber da definição correta de Lógica e de raciocínio lógico. Todos que fazem parte do universo pesquisado e colaboraram para essa pesquisa, de maneira formal e informal, conseguiram definir a Lógica e entender bem o que é necessário para ter o raciocínio lógico.

Em um dia com muitas atividades, a organização e a ordem lógica tornam-se muito importantes para que se obtenha um rendimento razoável e satisfatório. Alguns dos colaboradores dessa pesquisa têm uma vida agitada, corrida e

precisam que todas as atividades do dia sejam cumpridas.

O raciocínio, muitas vezes, pode ser de maneira incorreta. Entretanto, se há uma compreensão de que a Lógica é a correção do pensamento, a arte de bem pensar e coloca ordem no pensamento, todas as pessoas em geral, dedicando tempo para pensar e organizar suas atividades conseguem um raciocínio lógico perfeito. Imprevistos acontecem, porém se a arte de raciocinar bem for uma constância, as pessoas terão facilidade em resolver a situação da forma mais adequada e satisfatória.

Frequentemente, o raciocínio lógico é usado para fazer inferências, sendo que começa com uma afirmação ou proposição inicial, seguido de uma afirmação intermediária e uma conclusão. Assim, ele também é uma ferramenta analítica e sequencial para justificar, analisar, argumentar ou confirmar alguns raciocínios. É fundamentado em dados que podem ser comprovados, e por isso, é preciso e exato.

É possível resolver problemas usando o raciocínio lógico. No entanto, ele não pode ser ensinado diretamente, mas pode ser desenvolvido através da resolução de exercícios lógicos que contribuem para a evolução de algumas habilidades mentais.

Muitas empresas utilizam exercícios de raciocínio lógico para testarem a capacidade dos candidatos. Este tipo de avaliação também é comum em concursos públicos.

Tradicionalmente a lógica foi considerada um portal de acesso ao estudo da filosofia e das ciências. Faz sentido. Discutir e argumentar faz parte do debate sobre qualquer questão. No caso das ciências, conhecer um pouco de lógica pode ser muito valioso. As ciências foram construídas usando procedimentos lógicos e o método científico pode ser visto como lógica aplicada.

Segundo Irving Copi, o estudo da lógica é o estudo dos métodos e princípios usados para distinguir o raciocínio correto do incorreto. Consequentemente, uma pessoa com conhecimento de lógica tem mais probabilidades de raciocinar corretamente.

O estudo da lógica proporciona às pessoas analisar e identificar métodos incorretos de raciocinar. Não apenas os das outras pessoas, mas também seus próprios. Sempre foi importante e atualmente a lógica ganha um contexto ainda mais interessante dado a massa de desinformação que se propaga à uma velocidade nunca antes vista.

Não devemos nos apoiar no argumento de que a proliferação de conteúdo com objetivo de pura desinformação (que é diferente de um conteúdo apenas mentiroso) seria um motivo para limitar o uso da internet ou das redes sociais.

A peça fundamental da lógica é a proposição. Por definição, uma proposição é uma frase declarativa, escrita ou representada por símbolos, que aceita apenas dois valores - lógicos - possíveis: verdadeiro ou falso. Uma proposição pode ser simples ou composta, caso seja formada por duas ou mais proposições. Quer um exemplo? Se eu digo "O João possui um canal no Youtube", isto é uma proposição, pois aceita o resultado de verdadeiro ou falso. Diferentemente de quando você diz - "Tenha um bom dia!", onde aqui não cabe a atribuição dos dois valores lógicos.

Da proposição chegamos na contradição. Uma contradição acontece apenas quando temos uma proposição composta e seu resultado é sempre falso. A proposição composta "Eu gosto de todas as frutas, mas não gosto de morango" é uma contradição, pois morango é uma fruta.

Há outra coisa dentro da lógica que é o chamado argumento. Ele é um grupo de proposições iniciais que te conduz à uma outra proposição final, que será consequência das primeiras. Todo argumento possui uma premissa e conclusão (ou hipótese e tese). Um argumento pode ser considerado válido (ou legítimo) quando sua conclusão é uma consequência obrigatória de suas premissas. Intuitivamente conclui-se que um argumento é considerado inválido quando as suas premissas não são suficientes para provar a verdade da conclusão.

1.5. Tabela Verdade

Tabela verdade ou tabela de verdade é uma ferramenta de natureza matemática muito utilizada no campo do raciocínio lógico. Seu objetivo é verificar a validade lógica de uma proposição composta (argumento formado por duas ou mais proposições simples).

Exemplos de proposições compostas:

João é alto **e** Maria é baixa.

Pedro é alto **ou** Joana é loira.

Se Pedro é alto, **então** Joana é ruiva.

Cada uma das proposições compostas acima é formada por duas proposições simples unidas pelos conectivos em negrito. Cada proposição simples pode ser verdadeira ou falsa e isso implicará diretamente no valor lógico da proposição composta. Se adotarmos a frase "João é alto e Maria é baixa", as possíveis valorações dessa afirmação serão:

Se João for alto e Maria for baixa, a frase "João é alto e Maria é baixa" é VERDADEIRA.

Se João for alto e Maria não for baixa, a frase "João é alto e Maria é baixa" é FALSA.

Se João não for alto e Maria for baixa, a frase "João é alto e Maria é baixa" é FALSA.

Se João não for alto e Maria não for baixa, a frase "João é alto e Maria é baixa" é FALSA.

A tabela verdade esquematiza esse mesmo raciocínio de forma mais direta. Além disso, as regras da tabela verdade pode ser aplicada independentemente do número de proposições na frase.

Primeiramente, transformar-se as proposições da questão em símbolos utilizados na lógica. A lista de símbolos universalmente usada é:

Símbolo	Operação Lógica	Significado
p	-	Proposição 1
q	-	Proposição 2
~	Negação	não
^	Conjunção	e
v	Disjunção	ou
→	Condicional	se...então
↔	Bicondicional	se e somente se

Em seguida, monta-se uma tabela com todas as possibilidades de valoração de uma proposição composta, substituindo as afirmações por símbolos. Vale esclarecer que nos casos em que existirem mais de duas proposições, elas poderão ser simbolizadas pelas letras r, s, e assim por diante.

Por fim, aplica-se a operação lógica definida pelo conectivo mostrado. Conforme a lista acima,

essas operações podem ser: negação, conjunção, disjunção, condicional e bicondicional.

Nós vamos estudar melhor todos estes conceitos mais a frente em nosso curso, neste primeiro momento é importante saber que é comum que os estudiosos da tabela verdade memorizem as conclusões de cada uma das operações lógicas. Para economizar tempo na resolução de questões, tenha sempre em mente que:

1. Proposições Conjuntivas: Só serão verdadeiras quando todos os elementos forem verdadeiros.

2. Proposições Disjuntivas: Só serão falsas quando todos os elementos forem falsos.

3. Proposições Condicionais: Só serão falsas quando a primeira proposição for verdadeira e a segunda falsa.

4. Proposições Bicondicionais: Só serão verdadeiras quando todos os elementos forem verdadeiros, ou todos os elementos forem falsos.

1.6. Exercícios Passo a Passo

1. Nos exercícios a seguir, descreva a opção correta que preenche a lacuna da série: FEG, GEF, HEI, IEH, ____

2. ELFU, GLHU, ILJU,____, MLNU

3. BMM, DOO, FQQ,____, JUU

4. Considere a série de números: 26, 24, 20, 18, 14,... Qual é o próximo número?

5. Considere a série de números: 23, 24, 27, 28, 31, 32,... Qual é o próximo número?

6. Maria corre mais rápido do que Ana. Sílvia corre mais rápido do que Maria. Ana corre mais rápido do que Sílvia. Se as duas primeiras sentenças são verdadeiras, a terceira é:

7. Os apartamentos no bairro A custam menos do que no bairro B. Os apartamentos no bairro C custam mais do que no bairro B. Os apartamentos no bairro C são os mais caros. Se as duas primeiras sentenças são verdadeiras, a terceira é:

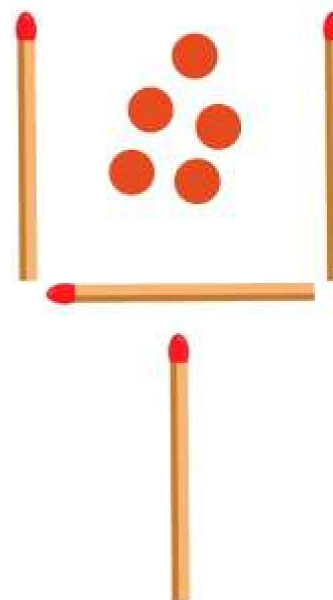
8. Durante o último mês, Diogo assistiu a mais filmes do que Felipe. Felipe assistiu a menos filmes do que Daniel. Daniel assistiu a mais filmes do que Diogo. Se as duas primeiras sentenças são verdadeiras, a terceira é:

9. O preço do ouro no mercado interno permaneceu inalterado durante a última semana. O preço do ouro no mercado internacional subiu durante a última semana. a) A sentença 1 é a causa e a sentença 2 é o seu efeito b) A sentença 2 é a causa e sentença 1 é o seu efeito c) As sentenças 1 e 2 são causas independentes d) As sentenças 1 e 2 são efeitos de causas independentes

10. O índice de estabelecimentos fiscalizados e multados por baixa qualidade de alimentos do município aumentou no último ano. A prefeitura implantou um programa de treinamento para os trabalhadores envolvidos no setor de vigilância sanitária. a) A sentença 1 é a causa e a sentença 2 é o seu efeito b) A sentença 2 é a causa e sentença 1 é o seu efeito c) As sentenças 1 e 2 são causas independentes d) As sentenças 1 e 2 são efeitos de causas independentes

1.7. Exercícios de Fixação

1. Solicite ao seu professor (caso esteja realizando o curso de forma EAD, busque estes objetos na sua casa) quatro palitos ou quatro canetas e uma pequena "bolinha" de papel, para realizar este exercicio. Ele consiste em montar uma espécie de "pazinha" e então o objetivo será remover o lixo de dentro da pazinha (bolinha de papel). É permitido mover apenas dois palitos (ou canetas) para realizar o objetivo.



2. Busque na internet qualquer desafio de travessia do rio. Conclua o desafio e mostre ao seu professor, no caso de estar realizando o seu curso de forma presencial na sua escola.

3. Realize o jogo de 7 erros abaixo, anote em uma folha qualquer os erros encontrados por você.

2. Variáveis, constantes e tipos de dados

Programas de computador utilizam os recursos de hardware mais básicos para executar algoritmos. Enquanto o processador executa os cálculos, a memória é responsável por armazenar dados e servi-los ao processador.

Por hora não se preocupe com o que seria algoritmos, vamos estudar mais profundamente sobre este assunto na próxima aula.

O recurso utilizado nos programas para escrever e ler dados da memória do computador é conhecido como variável, que é simplesmente um espaço na memória ao qual reservamos e damos um nome. Por exemplo, podemos criar uma variável chamada "idade" para armazenar a idade de uma pessoa. Você pode imaginar uma variável como uma gaveta "etiquetada" em um armário.

Quando criamos uma variável em nosso programa, especificamos que tipo de dados pode ser armazenado nela (dependendo da linguagem de programação). Por exemplo, a variável nome só poderia armazenar valores do tipo texto. Já a variável idade, só poderia armazenar valores do tipo número (inteiro).

Chamamos este espaço alocado na memória de variável, porque o valor armazenado neste espaço de memória pode ser alterado ao longo do tempo, ou seja, o valor ali alocado é "variável" ao longo do tempo. Diferente das constantes, que é um espaço reservado na memória para armazenar um valor que não muda com o tempo. Por exemplo, o valor PI (3.14159265359...) que nunca vai mudar!

Suponha que você precise fazer um programa que solicite ao usuário dois números inteiros, some esses dois números e apresente o resultado da soma para o usuário.

Para resolver esse problema, teremos de DECLARAR duas variáveis do TIPO inteiras. Vamos supor que essas duas variáveis se chamem X e Y. Além disso, você precisará de uma terceira variável para armazenar o resultado da soma. Vamos chamar então essa variável de SOMA. Sendo assim, teremos o seguinte algoritmo passo a passo:

```
algoritmo soma;

inicio

//DECLARAÇÃO DE VARIÁVEIS

inteiro x;

inteiro y;

inteiro soma;

// ESCREVA UMA MENSAGEM NA TELA

escreva ( “por favor, digite o valor do número x”) ;

// LE VALORES DO TECLADO DIGITADOS PELO USUÁRIO E ARMAZENA NA VARIÁVEL

leia (x);

escreva ( “por favor, digite o valor do número y”) ;

leia (y);

// REALIZA UMA OPERAÇÃO DE SOMA
```

soma <-- x + y;

escreva (" o resultado da soma x + y é:", soma);

fim

Não se preocupe com alguns detalhes desse algoritmo, o que é importante no momento é entender como funcionam as variáveis. Veja, o programa solicita ao usuário para digitar os valores de X e Y.

Nesse instante é impresso na tela a mensagem que está dentro dos parênteses do comando ESCREVA e, em seguida, o usuário digita um valor usando o teclado. No momento em que o usuário digita o valor desejado, esse valor DECIMAL é "passado" para a variável X, por meio do comando LEIA. Lembre-se, a variável X é um espaço de memória reservado, com o tamanho de bits para o tipo de dado INTEIRO. O mesmo acontece com a variável Y.

Esses valores ficarão armazenados nas variáveis enquanto nenhum outro valor for digitado para eles, ou enquanto o programa estiver em execução. Isso significa que depois que o programa terminar de executar, esses valores deixarão de existir, assim como as variáveis, e o espaço de memória que estava reservado será também liberado.

Então, quando alguém digita algo no teclado, isso pode ser obtido por um comando da linguagem de programação, armazenado em uma variável específica, e depois manipulado no resto do programa. A cada vez que você executar esse programa, essas variáveis terão valores diferentes, por isso são variáveis, são valores que mudam com o tempo.

Utilizando o exemplo da batata-frita, podemos dizer que o óleo é uma variável pois pode estar quente ou frio. Já a batata é uma constante, pois do início ao fim ela continua sendo uma batata. Seja ela frita, inteira, cortada, crua. Isso são atributos ou propriedades, mas ela ainda é uma batata.

Pare e pense em sua rotina ao acordar. Se você colocar no papel verá que ela é mais complexa do que parece, mas a execução é automática. Imagine então começar a pensar no que poderia ser variável e constante nessa rotina?

Constantes, é um espaço reservado na memória para armazenar um valor que não muda com o tempo. Por exemplo, o valor PI (3.14159265359...) que nunca vai mudar!

Suponha que você precise trabalhar com o número PI, que é um valor fixo de aproximadamente 3.14. Você pode simplesmente declará-lo e utilizá-lo em todo o seu programa:

algoritmo pi;

inicio

//DECLARAÇÃO DE VARIÁVEIS

real PI = 3.14;

real x;

real result;

// ESCREVA UMA MENSAGEM NA TELA

escreva ("por favor, digite o valor do número x") ;

// LE VALORES DO TECLADO DIGITADOS PELO USUÁRIO E ARMAZENA NA VARIÁVEL

leia (x);

// REALIZA UMA OPERAÇÃO MATEMÁTICA

result <-- x * PI;

escreva (" o resultado é:", result);

fim

Uma constante chamada PI foi declarada no início do algoritmo e, posteriormente, usada para realizar uma operação de multiplicação. Portanto, lembre-se: constantes e variáveis são espaços de memória reservados para o tipo de dados que você deseja trabalhar. Constantes são valores fixos que você utilizará em seu programa e variáveis são valores que precisam variar durante o tempo de execução do seu programa.

Nós sempre precisaremos declarar variáveis em nossos programas. Então, declarar variável é um termo importante em computação. Outro termo relevante é o IDENTIFICADOR. Um identificador é o nome que damos às nossas variáveis.

A sintaxe de declaração pode ser diferente para muitas linguagens de programação. Por exemplo, em Java fazemos assim: String nome, em que STRING é um TIPO DE DADO e NOME é o nome da variável, ou seja, o seu IDENTIFICADOR.

Algumas linguagens também aceitam letras minúsculas e maiúsculas, outras só minúsculas, é o que chamamos de sensitive case. Java, por exemplo, é uma linguagem sensitive case, isto é, aceita que você defina nomes de variáveis com letras maiúsculas e minúsculas.

2.1. Tipos de Dados

Para otimizar a utilização da memória, nós definimos um tipo de dados para cada variável. Por exemplo, a variável *nome*, deve armazenar textos, já a variável *idade* deve armazenar apenas números inteiros (sem casa decimal), na variável *sexo* podemos armazenar apenas um caractere ("M" ou "F"). Seria correto armazenarmos o valor "M" na variável *idade*? Não né? Por isso devemos especificar em nossos algoritmos o tipo de cada variável.

Podemos classificar os tipos de dados em basicamente duas categorias, os tipos de dados primitivos e os tipos de dados customizados.

2.2. Tipo de dados primitivos

Em computação existem apenas 4 tipos de dados primitivos, algumas linguagens subdividem esses tipos de dados em outros de acordo com a capacidade de memória necessária para a variável. Mas de modo geral, os tipos de dados primitivos são:

1. INTEIRO: Representa valores numéricos negativo ou positivo sem casa decimal, ou seja, valores inteiros.
2. REAL: Representa valores numéricos negativo ou positivo com casa decimal, ou seja, valores reais. Também são chamados de ponto flutuante.
3. LÓGICO: Representa valores booleanos, assumindo apenas dois estados, VERDADEIRO ou FALSO. Pode ser representado apenas um bit (que aceita apenas 1 ou 0).
4. TEXTO: Representa uma sequencia de um ou mais de caracteres, colocamos os valores do tipo TEXTO entre " " (aspas duplas).

Algumas linguagens de programação dividem esses tipos primitivos de acordo com o espaço necessário para os valores daquela variável. Na linguagem Java, por exemplo, o tipo de dados inteiro é dividido em 4 tipos primitivos: byte, short, int e long. A capacidade de armazenamento de cada um deles é diferente.

· **BYTE:** é capaz de armazenar valores entre -128 até 127.

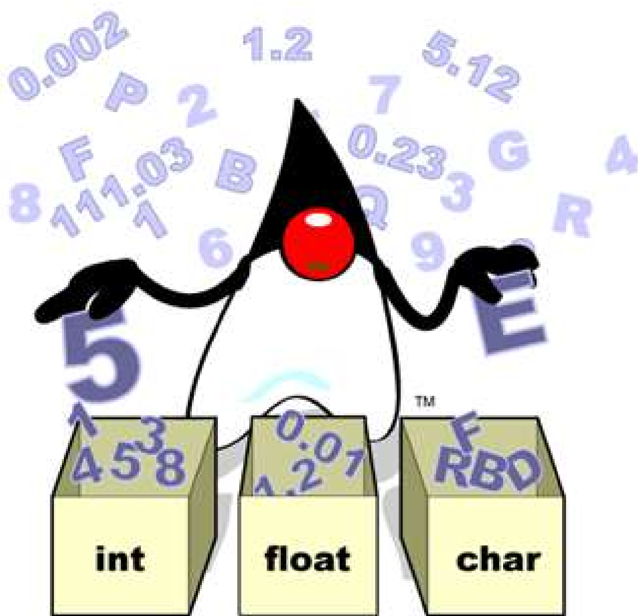
· **SHORT:** é capaz de armazenar valores entre – 32768 até 32767.

· **INT:** é capaz de armazenar valores entre

–2147483648 até 2147483647.

· **LONG:** é capaz de armazenar valores entre –9223372036854775808 até 9223372036854775807.

Mas essa divisão é uma particularidade da linguagem de programação. O objetivo é otimizar a utilização da memória. Em algumas linguagens de programação não é necessário especificar o tipo de dados da variável, eles são identificados dinamicamente. Porém, é necessário informar o tipo de dados de cada variável em algoritmos.



2.3. Tipo de dados customizados

A partir dos tipos de dados primitivos podemos criar outros tipos de dados utilizando uma combinação de variáveis. São estruturas de dados, classes, vetores, matrizes, etc.

Por exemplo, uma classe chamada *Carro* é um tipo de dados que agrupa outras variáveis básicas como **marca**, **cor**, **ano**, **modelo**, etc. Um *vetor* é um agrupamento de variáveis do mesmo tipo, uma *matriz* é um agrupamento de vetores. Enfim, a base de todos os tipos de dados são os tipos de dados primitivos, independente da linguagem de programação.

Claro, em Programação Orientada a Objetos há todo um conceito para a criação de classes que, além de atributos também tem operações, o estudo de estruturas de dados também vai muito além de apenas formar tipos de dados a partir de outros. Mas não se preocupe com isso por agora. Apenas entenda que são tipos de dados formados a partir de outros tipos de dados.

Diferente dos tipos de dados primitivos que já são implementados internamente pelas linguagens de programação, esses tipos de dados são criados pelo programador.

A todo o momento durante a execução de qualquer tipo de programa, os computadores estão manipulando informações representadas pelos diferentes tipos de dados que acabamos de descrever.

Para que não se "esqueça" das informações, o computador precisa guardá-las em sua memória, seja através de uma variável ou de uma constante.

Mas para tanto é necessário fazer a declaração das variáveis ou constantes.

O ato de declarar consiste em duas ações distintas. A primeira está relacionada a dar um nome para a variável ou constante. A segunda diz respeito a associar um tipo de dado, os quais acabamos de explicar.

Lembrando que uma variável ou constante só pode ser declarada como sendo de um mesmo tipo de dado.

Algo importante sobre os tipos de dados, e que quase ninguém se dá conta, é que eles devem primeiro ser possíveis de existirem, e de ser manipulados pelo hardware. Isso significa que é o Hardware quem diz quais os tipos de dados primitivos que podem ser processados por uma máquina computacional.

Hoje consumir memória não é um problema para microprocessadores, pois a temos em abundância, mas no começo da Computação, os programadores tinham que ficar atentos às quantidades de bits consumidas pelos seus tipos de dados, pois a memória era algo bem escasso. Os microcontroladores se encaixam nesse probleminha, pois sua memória é realmente bem

mais curta que as memórias usadas com microprocessadores.

Saber como funcionam as variáveis/constantes e os tipos de dados é de suma importância para você se tornar um bom programador.

2.4. Exercícios Passo a Passo

1. O que são tipos de dados primitivos?

2. Defina o que seria uma constante?

3. Defina o que seria dados customizados?

4. O que seria um identificador?

5. Defina o tipo de dados REAL?

6. Qual a diferença entre os dados primitivos e os customizados?

2.5. Exercícios de Fixação

1. No seguinte algoritmo existe algum erro? Onde?

```
ALGORITMO Teste
VARIÁVEIS
idade : INTEIRO
letra : CARACTER
Maria : REAL
INICIO
idade <- 23
idade <- 678
idade <- letra
letra <- ABC
letra <- A
letra <- 2
FIM
```

2. É correto definir uma variável como sendo Caracter e atribuirmos a ela o valor: ‘PEDRO’? E se a variável fosse definida como LÓGICO, a mesma poderia receber um valor do tipo CARACTER?

3. Identifique os erros e reescreva os identificadores abaixo de forma correta:

a) 13salário

b) salário$

c) salário_mínimo

d) salário+reajuste

e) novoSalário

f) fumante?

g) Preço médio

h) %desconto

i) km/h

4. Classifique os dados de acordo com o seu tipo, sendo (I = Inteiro, R = Real, C = Caractere e L = Lógico):

( ) 0

( ) + 36

( ) 0,3257

( ) F

( ) 1

( ) “F”

( ) “+3257”

( ) ?1

( ) 0,0

( ) ? 0,001

( ) “?0,0”

( ) “.F.”

( ) “o”

( ) + 0,05

( ) “.V.”

( ) 7/2

( ) 32

( ) + 3257

( ) V

( ) ?32

( ) “A”

( ) “abc”

( ) ?1,9E123

5. Na lista seguinte, assinale com V os nomes de variáveis válidos e com I os inválidos.

( ) abc

( ) 3abc

( ) a

( ) 123ª

( ) _a

( ) acd1

( ) _

( ) Aa

( ) 1

( ) A123

( ) _1

( ) A0123

( ) a123

( ) _a123

( ) b312

( ) AB CDE

( ) etc...

( ) guarda-chuva

anotações

3. Primeiro programa (algoritmos)

Algoritmo não é o bicho papão dos contos infantis. Se realmente pararmos para ver, ele é bastante simples, já que está frequentemente presente em nossas vidas. Ao começarmos a fazer algoritmos, fica muito mais fácil desenvolvê-los com segurança, quando realizamos analogias com coisas simples do dia a dia.

Algoritmo é a base da ciência da computação e da programação. Quando falamos em programar, falamos, basicamente, em construir um algoritmo. Todo programa de um computador é montado por algoritmos que resolvem problemas matemáticos lógicos com objetivos específicos.

Logo, o primeiro passo para se aprender programação não envolve computador, envolve educar a sua mente a explicar em detalhes os passos necessários para executar uma determinada tarefa.

Você deve aprender a modelar um roteiro que explica quando tomar decisões e quando realizar determinadas tarefas, esse roteiro é chamado de "algoritmo". Você sabia que os primeiros processadores só sabiam realizar somas? A partir dessa operação básica que o computador sabia fazer, você já imagina um algoritmo para fazer multiplicações? Talvez você ainda não saiba exatamente como é esse algoritmo, mas com certeza já imaginou que precisa fazer repetidas somas. Certo? É assim que nós aprendemos fazer multiplicação na escola. E essa também é uma forma de ensinar uma máquina a fazer multiplicação.

Um algoritmo não passa de passos sequenciais e lógicos que são organizados de forma a realizar a conclusão de certo problema. Mas precisamos entender que existem dois tipos de algoritmos – os Não Estruturados e os Estruturados.

Programadores usam algoritmos estruturados, pois se adequam a determinado objetivo ou certo fim.

Mas não são apenas os programadores que usam algoritmos. Em nosso cotidiano, os algoritmos Não Estruturados são trabalhados em nossas mentes sem nem mesmo percebermos. Vamos aprender mais sobre eles.

3.1. Algoritmos não estruturados

Para o desenvolvimento de um algoritmo eficiente, é necessário obedecermos a algumas premissas básicas no momento de sua construção:

- Definir ações simples e sem ambiguidade.
- Organizar as ações de forma ordenada.
- Estabelecer as ações dentro de uma sequência finita de passos.

Os algoritmos são capazes de realizar tarefas como:

- Ler e escrever dados
- Avaliar expressões algébricas, relacionais e lógicas.
- Tomar decisões com base nos resultados das expressões avaliadas.
- Repetir um conjunto de ações de acordo com uma condição.

Algoritmo Troca de pneu do carro:

1: desligar o carro.

2: pegar as ferramentas (chave e macaco).

3: pegar o estepe.

4: suspender o carro com o macaco.

5: desenroscar os 4 parafusos do pneu furado.

6: colocar o estepe.

7: enroscar os 4 parafusos.

8: baixar o carro com o macaco.

9: guardar as ferramentas.

O algoritmo exemplificado é um exemplo simples de algoritmo (sem condições ou repetições) para troca de um pneu.

Em nossa rotina, executamos algoritmos sem perceber. Quando você levanta pela manhã, quando você sai de casa, quando almoça. Você está sempre executando tarefas enquanto realiza análises de decisões, análises de possibilidades, valida argumentos e diversos outros processos.

Há muitos exemplos de algoritmos. Um deles são os manuais de instruções. Manuais de instruções sempre contêm informações detalhadas sobre o que fazer em cada situação e nos previnem de maiores problemas.

Agora pense desde quando você acorda até quando você volta a dormir. Quantas tarefas necessitam de suas decisões? Com certeza muitas. Lógico que não paramos para ver a vida detalhadamente, mas quando percebemos que o que fazemos faz parte de um grande algoritmo de decisões, escolhas, entendemos como funciona um algoritmo computacional.

Os algoritmos são descritos em uma linguagem chamada pseudo-código. Este nome é uma alusão à posterior implementação em uma linguagem de programação, ou seja, quando formos programar em uma linguagem, por exemplo Visual Basic, estaremos gerando código em Visual Basic. Por isso os algoritmos são independentes das linguagens de programação. Ao contrário de uma linguagem de programação não existe um formalismo rígido de como deve ser escrito o algoritmo.

O algoritmo deve ser fácil de se interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação.

3.2. Algoritmos estruturados

São aqueles que buscam resolver problemas através do uso de um computador. São criados com base em uma linguagem de programação e podem ser escritos de diversas formas.

Um algoritmo pode ser representado pelo chamado Português Estruturado, que é uma ferramenta que usa combinações de sequências, seleções e repetições. São evitados advérbios e adjetivos, formas verbais diferentes da imperativa, muita pontuação e descrição esparsa.

Para que um algoritmo em Português Estruturado seja conciso, ele deve seguir alguns pontos:

- Evitar comandos longos.
- Evitar adjetivos e advérbios.
- Ter comandos legíveis.
- Ter os comandos bem alinhados.
- Possuir comentários para o esclarecimento de comandos.
- Evitar ninhos de SE (muitas possibilidades desnecessárias para uma única situação. Ex.: se chover, eu pego o guarda-chuva; se não chover, eu não pego o guarda-chuva ou se eu pegar o vermelho, ele combinará com a roupa; se eu usar o roxo, não combinará).

Além de ações simples, como fazer um bolo ou abrir uma página no navegador, os algoritmos são muito utilizados no mercado da tecnologia. Atualmente possuímos máquinas que alimentam outras com informações e dados, conceito próximo do *machine learning*. Com isso, são construídos algoritmos capazes de reter informações e transformá-las em algo entendido pelos computadores para daí formar outro algoritmo.

A evolução humana nas últimas décadas está totalmente atrelada à ideia de algoritmos. Aparelhos como smartphones, computadores, smart TVs e tablets funcionam com sistemas baseados em algoritmos.

Conforme novos comandos e possibilidades de uso surgem, significa que mais aprimorados e complexos estão os níveis de instrução de um algoritmo.

3.3. Algoritmos no Facebook

Atualmente, diversas polêmicas estão relacionadas a como as grandes empresas de tecnologia têm usado os algoritmos para impactar a vida das pessoas.

Um dos casos mais famosos é o algoritmo do Facebook, que define o que será exibido no feed de notícias de cada usuário.

O Facebook foi a primeira rede social a usar algoritmos para categorizar os posts e utilizar critérios para definir o que seria ou não exibido para cada usuário. O objetivo era mostrar os conteúdos mais relevantes de acordo com o comportamento, preferências e engajamento do usuário.

Apesar das críticas à empresa, o excesso de publicações faz com que seja difícil uma pessoa acompanhar tudo que acontece na rede. Dessa forma, alguns dos milhares de critérios usados pela rede social para definir a composição do feed são:

- Postagens a serem mostradas.
- Nível de proximidade do usuário com quem postou o conteúdo.
- Engajamento de outros amigos com a publicação.
- Potencial engajamento do usuário considerando o comportamento prévio.

Portanto, diversos elementos são considerados pelo algoritmo antes de definir quais conteúdos serão exibidos no feed de notícias, buscando mais relevância e engajamento do usuário no Facebook.

3.4. Algoritmos no Google

As máquinas do Google são responsáveis por varrer a internet para indexar - armazenar as informações no banco de dados da Google - os bilhões de páginas encontradas na rede. Essas páginas, por sua vez, fornecem ao buscador uma série de dados, como textos, imagens e vídeos.

Seguindo mais de 200 critérios, o Google é capaz de categorizar essas páginas através de um sistema bem conhecido da empresa: o PageRank. Utilizando-se de algoritmos definidos por inteligência artificial, as máquinas ranqueiam os conteúdos de acordo com a relevância, mostrando-os na ordem do mais para o menos relevante nas páginas com os resultados da sua busca.

3.5. Algoritmos no Instagram

Quando foi fundado, o Instagram seguia a mesma lógica cronológica do Twitter, exibindo todas as postagens por ordem, das mais novas às mais antigas.

Essa estrutura foi alterada em 2016 e, atualmente, a rede social considera os seguintes fatores para escalar como serão os feeds dos usuários:

- temporalidade: ainda que não seja o único critério, a ordem de postagem ainda é considerada na definição do feed.
- engajamento: o número de comentários e curtidas determina se um post será priorizado ou não na rede, em especial, considerando esse engajamento logo após a postagem.
- relacionamento: considera a proximidade dos usuários por meio de engajamento, mensagens diretas etc.

Diante disso, verifica-se que os algoritmos usados tendem a ficar mais complexos, incluindo

novas variáveis, com o objetivo de torná-los mais certeiros.

3.6. Algoritmos no Spotify

Com mais de 40 milhões de usuários, o Spotify lança semanalmente a playlist “descobertas da semana”, que é personalizada para cada ouvinte e contém 30 músicas. Algumas delas, provavelmente, o usuário nunca ouviu, mas pode gostar, de acordo com o algoritmo.

O objetivo é criar experiências novas, mas, para concretizar esse objetivo, a ferramenta utiliza diferentes dados para processar as informações. Entre os critérios usados, estão:

- O perfil musical do usuário, considerando o gênero que ele gosta de ouvir e também artistas recentes;
- Como os demais usuários combinam as músicas, adicionando determinados grupos musicais ou canções em playlists;
- Considerando as preferências do usuário e o que o algoritmo aprendeu com os demais, são eleitas 30 músicas que não tenham sido tocadas pelo ouvinte na ferramenta.

O Spotify tem se dedicado a melhorar o algoritmo usado para aprimorar as sugestões e listas criadas, usando um amplo modelo de Machine Learning.

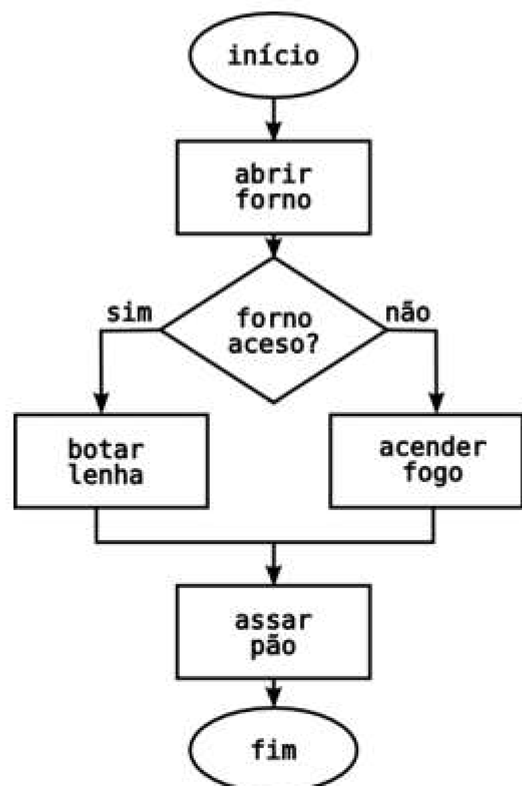
O sistema aprende as preferências e até mesmo identifica se uma música é feliz ou triste, buscando uma recomendação mais acertada para melhorar a experiência do usuário.

3.7. Representações de um Algoritmo

Um algoritmo pode ser representado através de algumas formas. Cada uma dessas formas de representar um algoritmo, possuem suas particularidades atribuindo a elas, vantagens e desvantagens, a melhor forma de se apresentar será a qual o programador ou o grupo de programadores está mais familiarizado ou se sentem mais confortáveis para elaborar seus algoritmos. Veremos a seguir alguns exemplos:

3.8. Fluxograma

Os Fluxogramas ou Diagramas de Fluxo são uma representação gráfica que utilizam formas geométricas padronizadas ligadas por setas de fluxo, para indicar as diversas ações (instruções) e decisões que devem ser seguidas para resolver o problema em questão. Eles permitem visualizar os caminhos (fluxos) e as etapas de processamento de dados possíveis e, dentro destas, os passos para a resolução do problema.



As desvantagens no uso de fluxogramas é a de que fluxogramas detalhados podem obscurecer a estrutura do programa.

O detalhamento através de textos dentro da estrutura gráfica causa confusão visual, obscurecendo a estrutura principal do programa.

3.9. Linguagem Natural

A linguagem natural é a maneira como expressamos nosso raciocínio e trocamos informação. Foi a linguagem que abordamos no início dos nossos estudos.

Como é a expressão da cultura de uma sociedade, desenvolvida através das gerações e em diferentes situações, raramente constitui um sistema de regras rígidas que possa ser implementada numa máquina ou que possa ser transcrita logicamente. Além da linguagem falada, fazem parte da nossa comunicação os gestos e as posturas, que não podem ser diretamente adaptados para compreensão de uma máquina. Por fim, toda a comunicação eficiente pressupõe um conhecimento prévio comum entre os interlocutores, por exemplo a mesma língua, a mesma bagagem cultural e assim por diante.

3.10. Linguagem de Programação

Para facilitar a tarefa de programar um computador, foram criadas várias linguagens de programação. Estas linguagens são uma maneira de tentar escrever as tarefas que o computador vai realizar de maneira mais parecida com a linguagem natural. Embora ainda seja muitas vezes complexo em comparação com a linguagem natural, um programa escrito em uma linguagem de programação é muito mais fácil de ser implementado, compreendido e modificado.

As linguagens de programação são um meio termo entre a linguagem de máquina e a linguagem natural. Deste modo são classificadas de acordo com o nível entre a linguagem natural ou de máquina que ocupam. As linguagens muito parecidas com linguagem de máquina são chamadas de linguagens de baixo nível e suas instruções parecem-se muito com aquelas que serão executadas pelo processador. As linguagens de alto-nível são as que guardam mais semelhanças com a linguagem natural. Exemplo de linguagens de baixo nível é a linguagem de montagem (assembly). Exemplos de linguagens de alto-nível são: Pascal, C, Fortran, Java, Perl, Python, Lisp, PHP, entre outras.

3.11. Pseudocódigo

Pseudocódigo é uma forma de representação de algoritmos, é praticamente um programa escrito em português que, depois, podemos passar para o computador. Para escrevermos códigos com pseudocódigo precisaremos conhecer alguns comandos básicos.

- escreva (" ") = comando usado para imprimir uma mensagem na tela.
- leia () = comando usado para ler valores digitados no teclado.
- <- = comando de atribuição.
- inicio = palavra usada para iniciar o programa principal.
- fimalgoritmo = palavra usada para finalizar o algoritmo.
- var = palavra usada para declarar variáveis;
- algoritmo = palavra usada para indicar o início do programa.

Pseudocódigo, será a forma em que vamos representar nos algoritmos praticamente o nosso curso inteiro.

Ela utiliza um conjunto restrito de palavras-chave, em geral na língua nativa do programador, que tem equivalentes nas linguagens de

programação. Além disso, o pseudocódigo não requer todo a rigidez sintática necessária numa linguagem de programação, permitindo que o aprendiz se detenha na lógica dos algoritmos e não no formalismo da sua representação. Na medida em que se obtém mais familiaridade com os algoritmos, então o pseudocódigo pode ser traduzido para uma linguagem de programação.

Um pseudocódigo, não pode ser executado num sistema real (computador), de outra forma deixaria de ser *pseudo*, assim, o português Estruturado na verdade é uma simplificação extrema da língua portuguesa, limitada a pouquíssimas palavras e estruturas que têm significado pré-definido, pois se deve seguir um padrão.

No caso da língua portuguesa existam alguns interpretadores de pseudocódigo, nenhum tem a projeção das linguagens C, Pascal ou BASIC, que no caso da língua inglesa se assemelham bastante a um pseudocódigo.

Os vocábulos desta metodologia são divididos em duas categorias: de aplicação pré-determinada, chamadas palavras reservadas e de definição customizada (durante criação do algoritmo), mas com regras pré-estabelecidas. Palavras reservadas podem ser comandos ou elementos de estruturas. Os comandos são invocações de tarefas específicas nos algoritmos, por exemplo, um comando de impressão deve imprimir uma mensagem de texto. A invocação de um comando é conhecida como chamada do comando. Os elementos de estruturas associam-se nas estruturas de controle para efetuar decisões ou repetições (dependendo do contexto).

As regras de sintaxe de um algoritmo tratam-se das restrições pré-estabelecidas para possibilitar a criação de pseudocódigos. Neste texto são definidas várias regras de sintaxe e mantidas até o final. É bom lembrar que tais regras não são universais, mas que em qualquer apresentação de metodologia algorítmica, um conjunto de regras de sintaxe se mostrará necessário para garantir concisão e clareza dos algoritmos.

Algoritmos, em pseudocódigo, apresentam-se como um texto contendo diversas linhas de código. Uma mesma linha pode conter um ou mais comandos ou ainda estruturas de controle. Várias linhas de código podem estar encapsuladas por um conceito lógico denominado **bloco**. O código de um bloco possui dessa forma início e fim e representa um sub-processo do processo geral de execução do algoritmo.

3.12. VisuAlg

Para quem está começando a aprender sobre programação de softwares, interpretar e executar algoritmos pode ser um verdadeiro desafio. Foi pensando nestes programadores em início de carreira que foi criado o software Visualizador de Algoritmos, mais conhecido como VisuAlg.

Foi criado pelo professor Cláudio Morgado de Souza da Apoio Informática, e teve a ajuda do professor Antonio Carlos Nicolodi, que inicialmente contribuiu com a manutenção do programa e posteriormente assumiu o desenvolvimento do mesmo a pedido do professor Cláudio que, por motivos pessoais, teve que se afastar do projeto que já contava com milhares de usuários por todo o Brasil.

É uma ferramenta na qual pode-se simular pseudocódigos, podemos dizer que é a interpretação de uma linguagem algorítmica, utilizando comandos e instruções em Português para representar as ações dos algoritmos, também conhecida como Portugol ou Português Estruturado.

O Portugol é uma versão portuguesa dos pseudocódigos que são realizados nos exemplos dos livros de introdução à programação ou lógica.

Ele permite o exercício prático dos conhecimentos teóricos em um ambiente próximo da realidade, e por isso é super indicado para quem está começando a trabalhar com a programação de softwares.

Funciona como se fossem as rodinhas de

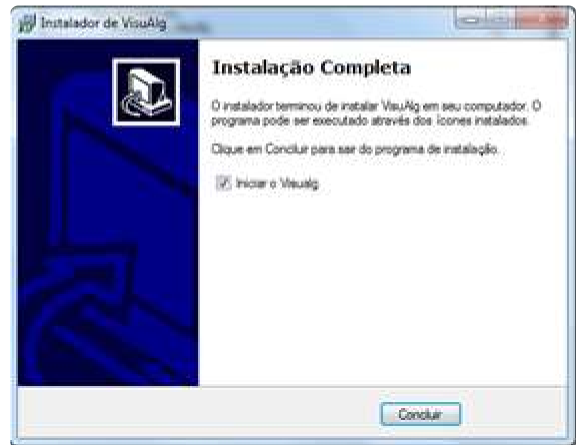
apoio para quem está aprendendo a andar de bicicleta, e que são retiradas quando deixam de ser necessárias.

É uma ferramenta capaz de simular o que acontece na tela do computador, com o uso de comandos como “leia” e “escreva”, e que possibilita a visualização de variáveis, acompanhamento passo a passo da execução de um algoritmo, e suporta um modo simples de depuração. Tudo isso com um editor de texto com recursos simples, como abrir e salvar arquivos, e que dispõe de todos os principais recursos de um ambiente gráfico.

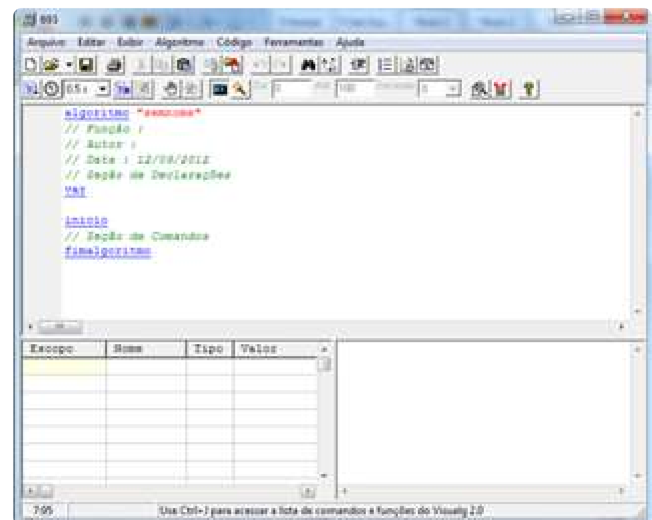
Antes do VisuAlg, os estudantes de programação não tinham uma opção simples de aprendizado. Ou executavam um programa apenas no papel, o que era um grande obstáculo para o aprendizado efetivo das técnicas de elaboração de algoritmos; ou tinham que se submeter aos rigores de uma linguagem de programação, que ainda não dominavam e por isso não conseguiam interpretar de forma eficiente.

Hoje em dia, já existem algumas versões do programa disponível para download na intenet. Nós utilizaremos em nosso curso, da versão 2 do programa. Todas as versões se encontram de forma gratuita na internet para baixar, basta colocar o nome do programa em seu buscador internet, que diversos links serão disponibilizados. Vamos agora a um breve passo a passo de como instalar o programa:

Após a primeira tela de instalação do programa, não existe nenhum mistério. Basta avançar deixando todas as opções padrões marcadas até chegar à tela final de instalação.



Após a conclusão de instalação do programa, basta executar através do atalho criado na área de trabalho do seu computador. A tela inicial do programa fica assim:



Explicando o código, o primeiro comando é o “Algoritmo” e o último é o “fimalgoritmo “, o comando “algoritmo” declara o início de um algoritmo, onde está escrito em vermelho “semnome” você coloca o nome do seu algoritmo (sempre entre aspas), por exemplo: algoritmo "exemplo1"

Quando se declara uma variável em VisuAlg existe uma regra. Primeiro você deve “dizer” que você está criando uma variável com tal nome e que ela vai ser de tal tipo, ou seja, seu conteúdo vai armazenar algum dado específico. Para isso você usa o comando “var” seguido do nome da variável, dois pontos, e o tipo de variável.

As variáveis podem ser do seguinte tipo:

- Inteiro – aceita somente valores numéricos que sejam inteiros (sem casas decimais).
- Real – aceita valores numéricos inteiros ou valores com casas decimais.
- Caractere – aceita textos alfanuméricos (letras, números e símbolos).
- Lógico – aceita somente valores booleanos: verdadeiro ou falso.

Exemplos:

Criar uma variável “nome” e “idade” em um programa

var nome : caractere

var idade : inteiro

criar uma variável "disponível" e "valor" em um programa

var disponivel : logico

var valor : real

- Leia - Recebe valores digitados pelos usuários, atribuindo-os às variáveis cujos nomes estão em (é respeitada a ordem especificada nesta lista). É análogo ao comando read do Pascal
- Escreva(l) - Escreve no dispositivo de saída padrão (isto é, na área à direita da metade inferior da tela do VisuAlg) o conteúdo de cada uma das expressões que compõem. As expressões dentro desta lista devem estar separadas por vírgulas; depois de serem avaliadas, seus resultados são impressos na ordem indicada. É equivalente ao comando write do Pascal.

A tela principal do VisuAlg é composta pela barra de tarefas no topo, o editor de texto que ocupa a maior parte da tela, o quadro de variáveis ao lado inferior esquerdo, o simulador de saída ao lado inferior direito e a barra de status na parte inferior. Quando o usuário abre o programa, já é carregado no editor um "esqueleto" de pseudocódigo, para dar menos trabalho e também mostrar o formato que deve ser seguido.

- Barra de Tarefas: A barra de tarefas contém os principais comandos do VisuAlg, que também podem ser utilizados através das teclas de atalho do teclado.
- Editor de Texto: Espaço onde será digitado todo o pseudocódigo.
- Quadro de Variáveis: É formado por uma grade onde é mostrado o escopo (se for do programa principal, será global; se for local, será apresentado o nome do subprograma onde foi declarada), nome (também com os índices, nos casos em que sejam vetores), tipo ("I" para inteiro, "R" para real, "C" para caractere e "L" para lógico) e o valor de cada variável.
- Simulador de Saída: Mostra como foi a última execução de código no programa.
- Barra de Status: Na barra de status o usuário pode ver em qual linha e coluna o cursor está posicionado, além de também ser avisado caso o código tenha sido alterado (o segundo painel da esquerda para a direita mostra a palavra "Modificado" quando o código está com alterações).

Agora que você já tem o Visualg, é hora de criar o seu primeiro programa. O famoso "Hello World". Abra o visualg e escreva o algoritmo abaixo:

```
algoritmo "BoasVindas"
// Função :
// Autor :
// Data : 08/04/2013
// Seção de Declarações
var
   nome: CARACTERE
inicio
// Seção de Comandos
   ESCREVA ("Olá! Digite o seu nome: ")
   LEIA (nome)
   ESCREVA ("Seja bem vindo ", nome, "!")
fimalgoritmo
```

Para executar, clique no menu Algoritmo e depois clique em Executar.

Vamos entender esse primeiro programa que você criou:

1. Na primeira linha, nós colocamos o nome do algoritmo "BoasVindas".

2. As quatro linhas seguintes são comentários, ou seja, é ignorado pelo compilador, não é um comando de algoritmo. Toda linguagem de programação tem alguma forma de fazer comentários no código. No Visualg os comentários começam com duas barras. Assim: //

Embora os comentários não sejam interpretados como comandos na hora de executar o programa, eles são muito importantes quando se escreve software, pois através dos comentários a gente explica o que uma parte do código faz para um outro programador que trabalhará neste mesmo código no futuro. Lembre-se: este programador pode ser você! É uma boa prática comentar códigos.

3. Em seguida vemos as declarações de variáveis. Nós declaramos uma variável chamada **nome** do tipo **CARACTERE**.

4. O programa começa de fato após a cláusula **inicio**. Perceba que depois do início tem outro comentário.

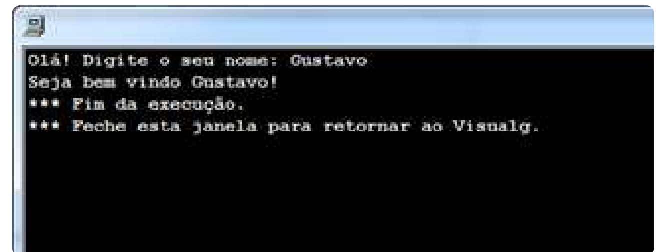
5. A primeira coisa que fazemos no programa é escrever na tela para o usuário: **Olá! digite o seu nome:** Nós fizemos isso através da função **ESCREVA**. Também falaremos sobre as funções mais pra frente neste curso, por hora, pense que a função vai fazer alguma coisa pra gente. No caso, escrever um texto na tela.

6. Na linha seguinte, nós capturamos o que o usuário digitou através da função **LEIA**. E armazenamos o texto que o usuário digitou na variável **nome**.

7. Por fim, nós mostramos na tela (novamente através da função **ESCREVA**): **Seja bem vindo (o valor da variável nome)!**

Note que nós juntamos ao texto **Seja bem vindo** o valor da variável nome. Se o usuário digitou **José** o programa vai exibir na tela: **Seja bem vindo José!**

Veja na imagem abaixo como acontece a execução do programa que acabamos de criar:



O VisuAlg é um programa simples, que não depende de DLLs, OCXs ou outros componentes. Sua instalação não copia arquivos para nenhuma outra pasta a não ser aquela em que for instalado, e exige cerca de 1 MB de espaço em disco. Pode ser executado sob Windows 95 ou posterior, e tem melhor aparência com resolução de vídeo de 800×600 ou maior.

Em nosso curso vamos desenvolver nossos programas usando o VisuAlg em praticamente todas as aulas. A cada aula vamos implementar novos comandos, deixando nossos projetos cada vez mais funcionais e complexos.

3.13. Exercícios Passo a Passo

1. Este exercício é apenas um desafio de lógica. Escreva um algoritmo que armazene o valor em uma variável A e outro valor em uma variável B. A seguir (utilizando apenas atribuições entre variáveis) troque os seus conteúdos fazendo com que o valor que está em A passe para B e vice-versa. Ao final, escrever os valores que ficaram armazenados nas variáveis.

2. Abra o programa VisuAlg em seu computador e digite o nome do algoritmo assim como determine as variáveis e aplique o comentário.

3. No campo Início, aplique o comentário indicado assim como o comando necessário para que o usuário digite um valor para A e o programa armazene este valor.

4. No campo Início, digite o comando solicitando que o usuário informe um valor para B e que este valor seja armazenado.

5. Insira o comentário referente ao local e insira o comando que determine o programa a mostrar na tela os valores de A e B que o usuário digitou.

6. Insira os comentários necessários que exemplifiquem como será realizada a troca das variáveis.

7. Insira os comandos necessários para realizar a troca de valores entre as variáveis.

8. Insira o comando necessário para que o programa retorne na tela os valores digitados pelo usuário, invertidos em cada variável.

9. Teste o programa clicando no menu Algoritmos e depois em Executar. No caso de você digitar os valores e o VisuAlg não quebrar as linhas, apenas acrescente a letra L minúscula ao lado de todos, os escreva (ficará assim: “escreval”) e teste novamente o código.

3.14. Exercícios de Fixação

1. Desenvolva um algoritmo que solicite para o usuário digitar duas notas e imprimir na tela sua média aritmética.

2. Desenvolva um algoritmo que solicite ao usuário que informe o valor de um produto qualquer, depois o valor pago por este produto. Ao final o programa deve retornar na tela o troco referente à compra.

3. Desenvolva um algoritmo que leia o nome de um vendedor e seu salário, mostrando o nome e seu salário com acréscimo de 10%.

4. A Loja Mamão com Açúcar está vendendo seus produtos em 5 (cinco) prestações sem juros.

Desenvolva um algoritmo que receba um valor de uma compra e mostre o valor das prestações.

5. Desenvolva um algoritmo que receba o preço de custo de um produto e mostre o valor de venda.

Sabe-se que o preço de custo receberá um acréscimo de acordo com um percentual informado pelo usuário.

anotações

4. Tipos de operadores

Geralmente as primeiras palavras que ouvimos em um curso de programação são: um programa é um conjunto de instruções lógicas que, quando executadas, produzem algum resultado. Com isso em mente, ao começar a criar as primeiras linhas de código, logo você notará que é comum receber dados do usuário, prover alguma lógica para processá-los e então apresentar o resultado desse processamento.

Na construção de nossos algoritmos, utilizaremos com frequência operadores para realizar o trabalho de entrada, processamento e saída de dados.

Um operador, em termos gerais, simboliza uma operação efetuada sobre variáveis e constantes, executando cálculos e manipulação sobre os dados.

Os operadores de atribuição, aritméticos, relacionais e lógicos são utilizados principalmente na etapa de processamento para a construção da lógica possibilitando realizar ações específicas sobre os dados. Adição, subtração, multiplicação, comparação são apenas alguns exemplos.

4.1. Operadores Aritméticos

Todo mundo já usou operadores aritméticos na escola! Nos primeiros anos de estudo aprendemos a fazer continhas de soma, subtração, multiplicação e divisão. Em algoritmos eles também são simples e têm a mesma simbologia em todas as linguagens de programação (+, -, * e /).

Além desses mais simples, dois outros operadores aritméticos não recebem muita atenção e pode ser que você não os conhece, eles são o div e o mod, que resultam, respectivamente, o quociente (a parte inteira do resultado da divisão) e o resto da divisão.

O operador **mod** em muitas linguagens de programação (java por exemplo) é representado pelo símbolo "**%** ".

Um outro operador aritmético que existe em algumas linguagens de programação é o **^** e executa a operação de potência, mas geralmente essa operação é realizada através de uma função chamada **pow**, bem como a operação de radiciação (função **sqrt)**.

Operadores aritméticos de radiciação também são fornecidos por algumas linguagens de programação, mas esses são bem mais raros. O Postgres por exemplo oferece os símbolos |/ e ||/ para operações de raiz quadrada e raiz cúbica, respectivamente.

4.2. Precedência entre Operadores Aritméticos

Da mesma forma que na matemática, os operadores de multiplicação e divisão têm precedência de execução em relação aos operadores de soma e subtração. Aliás se tiver parênteses na expressão estes têm precedência ainda maior. A sequência abaixo indica a precedência dos operadores:

1ª Parênteses internos.

2ª Potência (^) e raiz (quando a linguagem oferece esses operadores).

3ª * / div e mod

4ª + e -

Os operadores de mesma prioridade são interpretados da esquerda para a direita. Para exemplificar essa questão de precedência, observe a expressão:

*5 + 3 * (3 – 1) – 2 ^ 5 / 4 – 1*

O computador executa o cálculo na seguinte sequência:

*5 + 3 * **2** - 2 ^ 5 / 4 – 1*

*5 + 3 * 2 - **32** / 4 – 1*

*5 + **6** - 32 / 4 – 1*

*5 + 6 - **8** – 1*

***11** - 8 - 1*

***3** – 1*

2

Os operadores aritméticos realmente todo mundo deve saber desde criança, mas para criarmos algoritmos é muito importante conhecermos mais detalhes, como o operador **mod** ou a ordem de precedência de cada um.

4.3. Operadores Relacionais

Operadores relacionais são utilizados para comparar valores, o resultado de uma expressão relacional é um valor booleano (VERDADEIRO ou FALSO). Os operadores relacionais são: **igual**, **diferente**, **maior**, **menor**, **maior ou igual**, **menor ou igual**. Não é necessário explicar cada um, pois eles são auto-explicativos. Mas para quem é iniciante em desenvolvimento de softwares algumas informações podem ser importantes, principalmente pelo fato de haver diferença entre linguagens de programação.

Os operadores relacionais são diferentes dependendo da linguagem de programação, mas conhecendo os símbolos mais comuns, a maioria das linguagens de programação fica mais fácil aprender. No VisuAlg, os símbolos dos operadores relacionais são: =, <>, >, <, >=, <=.

Operação	Símbolo	Exemplo	Resultado
Igual	=	7 = 7	VERDADEIRO
Maior que	>	10 > 20	FALSO
Menor que	<	100 < 1000	VERDADEIRO
Menos ou igual a	<=	1.25 <= 2.50	VERDADEIRO
Maior ou igual a	>=	1234 >= 1234	VERDADEIRO
Diferente de	<>	10 <> 10	FALSO

O operador = é o único que pode ser usado com todos os tipos de dados. Abaixo, os tipos de dados e os operadores relacionais:

Logico: Só aceita **=** e **<>.**

Inteiro e real: Todos os operadores.

Literal: Aceita **=** e **<>** (Em alguns casos, pode-se usar **>**, **<** para verificar ordem alfabética do tipo literal).

Dentre os operadores relacionais, os vilões dos iniciantes são os símbolos para testar igualdade e diferença. Em cada linguagem é de um jeito! Em java, C, C#, javascript, etc. Por exemplo, os símbolos de igual e diferente são: == e !=. Já em Pascal, SQL, Visual Basic, ... os símbolos de igual e diferente são: = e <>. Então, fique esperto quando for aprender alguma dessas linguagens!

Operador	Comparação
==	Igual
!=	Diferente
<	Menor
>	Maior
<=	Menor Igual
>=	Maior Igual

Em java, não é possível testar Strings com o operador de igualdade (==), pois String é uma classe e não um tipo primitivo, e para testar a igualdade entre objetos deve-se utilizar o método equals. Assim: nome.equals("João").

É comum na programação a necessidade de conhecer a relação entre diversos operandos para que então, o nosso programa assuma determinada característica, ou invoque alguma funcionalidade. A linguagem Python por exemplo, trabalha com os operadores relacionais, também chamados de operadores comparativos, da mesma forma que a maioria das outras linguagens, tais como C, C++, Java, C# e etc.

Os operadores relacionais são muito utilizados em programação, as decisões dos algoritmos geralmente são tomadas nas operações relacionais, ou seja, as decisões baseiam-se em testes do estado das variáveis. Então é muito importante entender o que é uma operação relacional e quais os operadores utilizados nesse tipo de expressão.

4.4. Operadores Lógicos

As operações lógicas são ensinadas em vários cursos de tecnologia de diferentes formas, por exemplo, em cursos de eletrônica é ensinado portas lógicas, já em programação aprendemos os operadores lógicos. Mas no fundo é a mesma coisa e se você entender a ideia das operações lógicas você pode usar esse conhecimento em qualquer área da tecnologia.

4.5. Tipos de Dados Lógicos

O tipo de dados primitivo mais simples é o chamado **booleano** (ou lógico). Pra quem não conhece esse tipo de dados, um dado booleano só pode assumir dois valores (**VERDADEIRO** ou **FALSO**). Em eletrônica, costuma-se ensinar apresentando como exemplo uma lâmpada, que pode estar acesa (verdadeiro) ou apagada (falso). Isso é o básico. Na literatura você pode encontrar esses dados de diferentes formas, por exemplo: verdadeiro/falso, aceso/apagado, 1/0,

ligado/desligado, true/false, sim/não, etc....

Operação	Símbolo	Descrição
Negação	nao	Operador unário de negação.
Conjunção	ou	Operador que resulta VERDADEIRO quando um dos seus operandos lógicos for verdadeiro.
Disjunção	e	Operador que resulta VERDADEIRO somente se seus dois operandos lógicos forem verdadeiros.
Disjunção exclusiva	xou	Operador que resulta VERDADEIRO se seus dois operandos lógicos forem diferentes, e FALSO se forem iguais.

As operações lógicas trabalham sobre valores booleanos, tanto os valores de entrada como o de saída são desse tipo. Os operadores lógicos são: E, OU, NÃO, NÃO-E, NÃO-OU, OU-EXCLUSIVO E NÃO-OU-EXCLUSIVO.

4.6. Operador E (AND)

O Operador “E” ou “AND” resulta em um valor VERDADEIRO se os dois valores de entrada da operação forem VERDADEIROS, caso contrário o resultado é FALSO. Abaixo a **tabela-verdade** da operação E.

x1	x2	x1 AND x2
0	0	0
0	1	0
1	0	0
1	1	1

4.7. Operador OU (OR)

O Operador “OU” ou “OR” resulta em um valor VERDADEIRO se ao menos UM dos dois valores de entrada da operação for VERDADEIRO, caso contrário o resultado é FALSO. Abaixo a **tabela-verdade** da operação OU.

x1	x2	x1 OR x2
0	0	0
0	1	1
1	0	1
1	1	1

4.8. Operador NÂO (NOT)

O Operador “NÃO” ou “NOT” é o único operador que recebe como entrada apenas um

valor, e sua função é simplesmente inverter os valores. Ou seja, se o valor de entrada for VERDADEIRO, o resultado será FALSO e se o valor de entrada for FALSO, o resultado será VERDADEIRO. Abaixo a tabela-verdade da operação NÃO.

x1	NOT x1
0	1
1	0

4.9. Operador NÂO-E (NAND)

O Operador "NÃO-E" ou "NAND" é o contrário do operador E (AND), ou seja, resulta em VERDADEIRO, se ao menos um dos dois valores for FALSO, na verdade este é o operador E (AND) seguido do operador NÃO (NOT). Abaixo a **tabela-verdade** da operação NÃO-E.

x1	x2	x1 NAND x2
0	0	1
0	1	1
1	0	1
1	1	0

4.10. Operador NÂO-OU (NOR)

O Operador "NÃO-OU" ou "NOR" é o contrário do operador OU (OR), ou seja, resulta em VERDADEIRO, se os dois valores forem FALSO, na verdade este é o operador OU (OR) seguido do operador NÃO (NOT). Abaixo a **tabela-verdade** da operação NÃO-OU.

x1	x2	x1 XOR x2
0	0	1
0	1	0
1	0	0
1	1	0

4.11. Operador OU-EXCLUSIVO (XOR)

O Operador "OU-EXCLUSIVO" ou "XOR" é uma variação interessante do operador OU (OR), ele resulta em VERDADEIRO se apenas um dos valores de entrada for VERDADEIRO, ou seja, apenas se os valores de entrada forem DIFERENTES. Abaixo a **tabela-verdade** da operação OU-EXCLUSIVO.

x1	x2	x1 XOR x2
0	0	0
0	1	1
1	0	1
1	1	0

4.12. Operador NÂO-OU-EXCLUSIVO (XNOR)

O Operador "NÃO-OU-EXCLUSIVO" ou "XNOR" é o contrário do operador OU-EXCLUSIVO (XOR), ou seja, resulta VERDADEIRO se os valores de entrada forem IGUAIS. Observe a tabela abaixo:

x1	x2	x1 XOR x2
0	0	1
0	1	0
1	0	0
1	1	1

Cada linguagem de programação tem uma forma de representar os operadores lógicos. A simbologia mais encontrada são:

- AND, OR e NOT em linguagens como: Pascal, Visual Basic e SQL.
- &&, || e ! em linguagens como: Java e C#

Algumas linguagens oferecem operadores lógicos para o nível de bit (também chamado de operadores bitwise). Ou seja, podemos fazer operações lógicas com os bits de dois números. Em java, por exemplo esses operadores são & e |.

Além dos operadores que estudamos nesta aula, existem também os operadores de **incremento e de decremento.**

Neste primeiro momento, não vamos nos aprofundar muito no estudo deles, basta compreendermos que os operadores de incremento (++) e de decremento (--) são operadores unários que adicionam e subtraem uma unidade do conteúdo da variável respectiva:

Instrução	Equivalência
var++	var = var + 1
++var	var = var + 1
var--	var = var – 1
--var	var = var – 1

O valor da variável será incrementado (ou decrementado) depois ou antes da execução da instrução de que ela faz parte.

Conhecer esses operadores é muito importante para qualquer área da tecnologia que você for trabalhar. Em programação por exemplo, utilizamos esses operadores praticamente o tempo todo, principalmente para controle de fluxo de execução e tomadas de decisão.

4.13. Exercícios Passo a Passo

1. Abra o VisuAlg em seu computador e digite o nome correspondente ao algoritmo a ser confeccionado.

2. Insira os nomes das variáveis no programa e determine que os valores usados serão do tipo REAL.

3. Determine que o usuário escreva o primeiro valor no programa e que ele seja armazenado pelo VisuAlg.

4. Determine que o usuário escreva o segundo valor no programa e que ele seja armazenado pelo VisuAlg.

5. Determine que o usuário escreva o terceiro valor no programa e que ele seja armazenado pelo VisuAlg.

6. Determine que o usuário escreva o quarto valor no programa e que ele seja armazenado pelo VisuAlg.

7. Determine que o usuário escreva o quinto valor no programa e que ele seja armazenado pelo VisuAlg.

8. Insira o comentário referente a operação realizada e então aplique o comando que deverá realizar o cálculo do valor da média.

9. Insira o comentário referente a exibição do resultado na tela, assim como o comando necessário para o procedimento.

10. Realize alguns testes no programa, inserindo valores aleatoriamente.

4.14. Exercícios de Fixação

1. Desenvolva um algoritmo que leia um número e mostre o seu quadrado e o seu cubo.

2. Desenvolva um algoritmo que calcule e apresente o valor do volume de uma lata de óleo, utilizando a fórmula V = 3.14159 * R * R * A, em que R é o valor do raio e A da altura, que são fornecidas pelo usuário.

3. Desenvolva um algoritmo que receba o nome e o ano de nascimento de uma pessoa e calcule a idade que a pessoa fez no ano passado e a idade que terá daqui 10 anos.

4. Desenvolva um algoritmo que receba um número inteiro e mostre qual a unidade do número.

5. Escreva um algoritmo que receba o valor de um produto em reais e o desconto em porcentagem a ser oferecido na venda do produto. Informe o valor do desconto e o valor que o produto deverá ser vendido.

5. Estrutura de decisão parte 1

A maioria dos algoritmos precisam tomar decisões ao longo de sua execução. Para isso existem as estruturas de decisão, e a mais utilizada é a estrutura **SE-ENTÃO-SENÃO** (Em inglês **IF-THEN-ELSE**). O funcionamento é simples: com base no resultado de uma expressão booleana (**VERDADEIRO** ou **FALSO**), o fluxo do algoritmo segue para um bloco de instruções ou não.

Observe o esquema da estrutura **SE-ENTÃO-SENÃO**:

SE expressão booleana ENTÃO

instruções a serem executadas caso a expressão booleana resulte em VERDADEIRO

SENÃO

instruções a serem executadas caso a expressão booleana resulte em FALSO

FIM-SE

O bloco de código SENÃO é opcional. É comum encontrar instruções de decisão apenas com SE-ENTÃO sem o bloco SENÃO.

A classificação das estruturas de decisão é feita de acordo com o número de condições que devem ser testadas para que se decida qual o caminho a ser seguido.

Chamamos de estruturas de decisão encadeadas, quando uma estrutura de decisão está localizada dentro do lado falso da outra. Este tipo de estrutura também é conhecida como seleção "aninhada" ou seleção "encaixada".

Qualquer que seja o termo usado para identificar a estrutura, o importante é que esse formato com uma estrutura de seleção dentro da outra permite fazer a escolha de apenas um entre vários comandos possíveis.

5.1. Estrutura de Decisão Simples (Se...Então)

Nesta estrutura uma única condição (expressão lógica) é avaliada. Dependendo do resultado desta avaliação, um comando ou conjunto de comandos serão executados (se a avaliação for verdadeira) ou não serão executados (se a avaliação for falsa).

Há duas sintaxes possíveis para a estrutura de decisão simples:

SE ***<condição>*** **ENTÃO** ***<comando_único>***

Ex: **SE** X > 10 **ENTÃO** Escreva "X é maior que 10"

SE ***<condição>*** **ENTÃO**
INÍCIO
<comando_composto>
FIM

Ex: **SE** X > 10 **ENTÃO**
INÍCIO
cont := cont + 1
soma := soma + x
Escreva "X é maior que 10"
FIM

A semântica desta construção é a seguinte - A condição é avaliada:

Se o resultado for verdadeiro, então o **comando_único** ou o conjunto de comandos (**comando_composto**) delimitados pelas palavras-reservadas **início** e **fim** serão executados. Ao término de sua execução o fluxo do algoritmo prossegue pela instrução seguinte à construção, ou seja, o primeiro comando após o **comando_único** ou a palavra-reservada fim.

No caso de a condição ser falsa, o fluxo do algoritmo prossegue pela instrução seguinte à construção, ou seja, o primeiro comando após o **comando_único** ou a palavra-reservada **fim**, sem executar o **comando_único** ou o conjunto de comandos (**comando_composto**) entre as palavras-reservadas **início** e **fim**.

Exemplo de algoritmo que lê um número e escreve se o mesmo é maior que 10:

```
Algoritmo exemplo_estrutura_de_decisão_simples
Var X : inteiro
Início
Leia X
Se X > 10 Então Escreva "X é maior que 10"
Fim.
```

5.2. Estrutura de Decisão Composta (Se...Então...Senão)

Nesta estrutura uma única condição (expressão lógica) é avaliada. Se o resultado desta avaliação for verdadeiro, um comando ou conjunto de comandos serão executados. Caso contrário, ou seja, quando o resultado da avaliação for falso, um outro comando ou um outro conjunto de comandos serão executados. Há duas sintaxes possíveis para a estrutura de decisão composta:

```
SE <condição> ENTÃO <comando_único_1>
              SENÃO <comando_único_2>

     Ex:  SE   X > 100      ENTÃO  Escreva "X é maior que 100"
                            SENÃO  Escreva "X não é maior que 100"

SE <condição> ENTÃO
    INÍCIO
    <comando_composto_1>
    FIM
    SENÃO
    INÍCIO
    <comando_composto_2>
    FIM

     Ex: SE X > 100 ENTÃO
              INÍCIO
              cont_a := cont_a + 1
              soma_a := soma_a + x
              Escreva "X é maior que 100"
              FIM
              SENÃO
              INÍCIO
              cont_b := cont_b + 1
              soma_b := soma_b + x
              Escreva "X não é maior que 100"
              FIM
```

A semântica desta construção é a seguinte - A condição é avaliada:

Se o resultado for verdadeiro, então o **comando_único_1** ou o conjunto de comandos (**comando_composto_1**) delimitados pelas palavras-reservadas **início** e **fim** serão executados. Ao término de sua execução o fluxo do algoritmo prossegue pela instrução seguinte à construção, ou seja, o primeiro comando após o **comando_único_2** ou a palavra-reservada **fim** do **comando_composto_2.**

Nos casos em que a condição é avaliada como falsa, o **comando_único_2** ou o conjunto de comandos (**comando_composto_2**) delimitados pelas palavras-reservadas **início** e **fim** serão executados. Ao término de sua execução o fluxo do algoritmo prossegue pela instrução seguinte à construção, ou seja, o primeiro comando após o **comando_único_2** ou a palavra-reservada **fim** do **comando_composto_2**.

Exemplo de algoritmo que lê um número e escreve se o mesmo é ou não maior que 100:

```
Algoritmo exemplo_estrutura_de_decisão_composta
Var X : inteiro
Início
Leia X
Se X > 100 Então Escreva "X é maior que 100"
            Senão Escreva "X não é maior que 100"
Fim.
```

Nos algoritmos, a correta formulação de condições, isto é, expressões lógicas, é de fundamental importância, visto que as estruturas de seleção são baseadas nelas. As diversas formulações das condições podem levar a algoritmos distintos.

Considerando o problema:

"Dado um par de valores x, y, que representam as coordenadas de um ponto no plano, determinar o quadrante ao qual pertence o ponto, ou se está sobre um dos eixos cartesianos."

A solução do problema consiste em determinar todas as combinações de x e y para as classes de valores positivos, negativos e nulos.

Os algoritmos podem ser baseados em estruturas concatenadas uma em sequência a outra ou em estruturas aninhadas uma dentro da outra, de acordo com a formulação da condição.

O algoritmo a seguir utiliza estruturas concatenadas:

```
Algoritmo estruturas_concatenadas
Var x, y : inteiro
Início
Ler x, y
Se x=0  e y=0  Então Escrever "Ponto na origem"
Se x=0  e y<>0 Então Escrever "Ponto sobre o eixo y"
Se x<>0 e y=0  Então Escrever "Ponto sobre o eixo x"
Se x>0  e y>0  Então Escrever "Ponto no quandrante 1"
Se x<0  e y>0  Então Escrever "Ponto no quandrante 2"
Se x<0  e y<0  Então Escrever "Ponto no quandrante 3"
Se x>0  e y<0  Então Escrever "Ponto no quandrante 4"
Fim.
```

O algoritmo a seguir utiliza estruturas aninhadas ou encadeadas.

```
Algoritmo estruturas_aninhadas
Var x, y : inteiro
Início
Ler x, y
Se x<>0
Então Se y=0
        Então Escrever "Ponto sobre o eixo x"
        Senão Se x>0
              Então Se y>0
                    Então Escrever "Ponto no quandrante 1"
                    Senão Escrever "Ponto no quandrante 4"
              Senão Se y>0
                    Então Escrever "Ponto no quandrante 2"
                    Senão Escrever "Ponto no quandrante 3"
Senão Se y=0
        Então Escrever "Ponto na origem"
        Senão Escrever "Ponto sobre o eixo y"
Fim.
```

As estruturas concatenadas tem a vantagem de tornar o algoritmo mais legível, facilitando a correção do mesmo em caso de erros. As estruturas aninhadas ou encadeadas têm a vantagem de tornar o algoritmo mais rápido pois são efetuados menos testes e menos comparações, o que resulta num menor número de passos para chegar ao final do mesmo.

Normalmente se usa estruturas concatenadas nos algoritmos devido à facilidade de entendimento das mesmas e estruturas aninhadas ou encadeadas somente nos casos em que seu uso é fundamental.

As Estruturas de Decisões são muito utilizadas no dia-a-dia de nós programadores, ela vem nos auxiliar na tomada de decisões (por exemplo: Fazer login de usuário, se o login e senha estiverem corretos, é liberado o acesso se não, é emitido uma mensagem de erro). Por isso, é de extrema importância dominarmos este assunto.

5.3. Exercícios Passo a Passo

1. Neste exercício, vamos desenvolver um algoritmo que receba o nome do aluno, leia duas notas e retorne a média em caso de o aluno estar aprovado, caso contrário o aluno deve comparecer na coordenação do seu curso. Insira as variáveis dentro do programa VisuAlg.

2. Insira o comando no VisuAlg que solicite o nome do aluno e armazene este nome.

3. Insira o comando no VisuAlg que solicite o valor da primeira nota do aluno e armazene este valor.

4. Insira o comando no VisuAlg que solicite o valor da segunda nota do aluno e armazene este valor.

5. Insira o comando referente ao cálculo necessário para descobrir a nota do aluno.

6. Insira a primeira condição SE, para o caso de o aluno ter obtido uma nota igual ou superior a 7.

7. Insira o comando, que retorna a média do aluno em caso de ele estar aprovado.

8. Insira a segunda condição SE, em caso de o aluno ter obtido uma nota igual ou inferior a 7.

9. Insira o comando que retorna apenas a mensagem de que o aluno precisa comparecer a coordenação do curso.

10. Teste o programa, usando nomes aleatórios, assim como valores aleatórios para as notas do aluno.

5.4. Exercícios de Fixação

1. Desenvolva um algoritmo que receba uma medida em metros e converta em milímetros.

2. Desenvolva um algoritmo que leia dois valores inteiros e distintos e informe qual é o maior.

3. Desenvolva um algoritmo que receba um número e diga se este número está no intervalo entre 100 e 200.

4. Desenvolva um algoritmo que leia o nome e as três notas obtidas por um aluno durante o semestre, calcule a média (aritmética) e informe o nome e sua menção aprovado (média >= 7), reprovado (média <= 5) ou em recuperação (média entre 5,1 a 6,9).

5. Desenvolva um algoritmo que receba um número e mostre uma mensagem, caso este número seja maior que 80, menor que 25 ou igual a 40.

anotações

Lógica de programação

Lógica de Programação - Versão 2

6. Estrutura de decisão parte 2

Aula 6

A última estrutura de decisão que temos à disposição é a estrutura de múltipla escolha, utilizada quando precisamos selecionar as ações do algoritmo com base em diferentes valores de uma variável.

6.1. Escolha-Caso

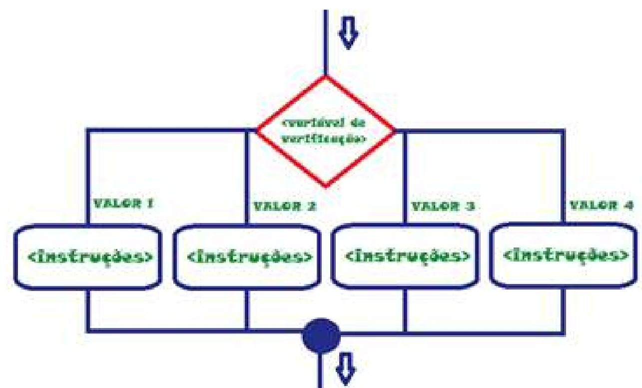
A estrutura ESCOLHA-CASO (em inglês SWITCH-CASE), é uma solução elegante quanto se tem várias estruturas de decisão (SE-ENTÃO-SENÃO) aninhadas. Isto é, quando outras verificações são feitas caso a anterior tenha falhado (ou seja, o fluxo do algoritmo entrou no bloco SENÃO). A proposta da estrutura ESCOLHA-CASO é permitir ir direto no bloco de código desejado, dependendo do valor de uma variável de verificação.

```
escolha (variável)
  caso v1:
    bloco de comandos 1;
  caso v2:
    bloco de comandos 2;
  ...
  caso vn:
    bloco de comandos n;
  caso contrário:
    bloco de comandos padrão;
fimescolha;
```

A utilização da estrutura de múltipla escolha é ilustrada acima, sendo iniciada pela linha "escolha (variável)", e finalizada na linha "fimescolha;". Uma condição opcional ("caso contrário"), pode ser ou não adicionada ao final dessa estrutura de seleção, sendo que o bloco de comandos correspondente à essa opção, será executado somente quando o valor da variável não atender à nenhuma das condições anteriores.

Assim, de forma resumida, o conteúdo da variável é comparado à um valor v1 qualquer e, se forem iguais, o bloco de ações 1 é executado. No caso de serem diferentes, as outras condições serão testadas, até que seja encontrada uma igualdade ou que terminem todos os casos possíveis.

Também nomeado de "**Estrutura de seleção múltipla**", basicamente conseguimos ter várias condições para dados diferentes, simplificando um uso muito grande do **"se"**. *Exemplo*: "Iremos pedir para o usuário digitar um inteiro, caso o número seja 100, apareça a frase "número cem"; caso seja digitado 1, apareça a mensagem "número um", e para todos os valores diferentes apareça uma mensagem diferente: "Números diferentes de um e cem".



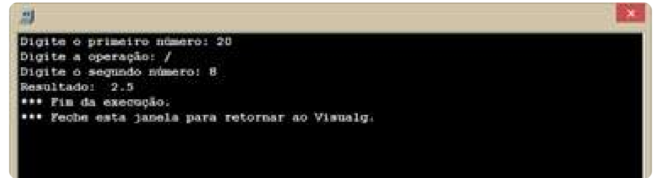
Para exemplificar a melhoria oferecida por essa estrutura, imagine a seguinte situação: Você deseja criar um algoritmo para uma calculadora, o usuário digita o primeiro número, a operação que deseja executar e o segundo número. Dependendo do que o usuário informar como operador, o algoritmo executará um cálculo diferente (soma, subtração, multiplicação ou divisão). Vejamos como seria esse algoritmo implementado no VisuAlg com SE-ENTÃO-SENÃO.

```
algoritmo "CalculadoraBasicaComSE"
var
   numero1 : REAL
   numero2 : REAL
   operacao : CARACTERE
   resultado : REAL
inicio

      ESCREVA ("Digite o primeiro número: ")
      LEIA (numero1)
      ESCREVA ("Digite a operação: ")
      LEIA (operacao)
      ESCREVA ("Digite o segundo número: ")
      LEIA (numero2)

      SE operacao = "+" ENTAO
         resultado := numero1 + numero2
      SENAO
         SE operacao = "-" ENTAO
            resultado := numero1 - numero2
         SENAO
            SE operacao = "*" ENTAO
               resultado := numero1 * numero2
            SENAO
               SE operacao = "/" ENTAO
                  resultado := numero1 / numero2
               FIMSE
            FIMSE
         FIMSE
      FIMSE

      ESCREVA ("Resultado: ", resultado)

fimalgoritmo
```

Veja como os SEs aninhados (dentro dos SENÃOs) deixam o código mais complexo. Dá pra entender a lógica, mas não é muito elegante. Agora vamos ver como ficaria a mesma lógica com a estrutura ESCOLHA-CASO.

```
algoritmo "CalculadoraBasicaComSE"
var
   numero1 : REAL
   numero2 : REAL
   operacao : CARACTERE
   resultado : REAL
inicio

      ESCREVA ("Digite o primeiro número: ")
      LEIA (numero1)
      ESCREVA ("Digite a operação: ")
      LEIA (operacao)
      ESCREVA ("Digite o segundo número: ")
      LEIA (numero2)

      ESCOLHA operacao
         CASO "+"
            resultado := numero1 + numero2
         CASO "-"
            resultado := numero1 - numero2
         CASO "*"
            resultado := numero1 * numero2
         CASO "/"
            resultado := numero1 / numero2
      FIMESCOLHA

      ESCREVA ("Resultado: ", resultado)

fimalgoritmo
```

Bem mais bonito! Agora a lógica tá mais visível e elegante. O resultado dos dois algoritmos é o mesmo, veja um exemplo de execução deste programa.



6.2. OUTROCASO

Existe uma opção a mais nessa estrutura, justamente para tratar quando o valor da variável não é equivalente a nenhum valor informado como opção nos CASOs, ou seja, é um "OUTROCASO". No algoritmo listado anteriormente, imagine se o usuário digitasse um valor diferente de "+", "-", "*" e "/". Caso quiséssemos apresentar uma mensagem para o usuário informando que ele digitou uma opção inválida, utilizaríamos esse recurso da estrutura ESCOLHA-CASO. Veja.

```
ESCOLHA operacao
   CASO "+"
      resultado := numero1 + numero2
   CASO "-"
      resultado := numero1 - numero2
   CASO "*"
      resultado := numero1 * numero2
   CASO "/"
      resultado := numero1 / numero2
   OUTROCASO
      ESCREVA("A operação digitada é inválida!")
FIMESCOLHA
```

Como pudemos observar, em termos de organização de código a estrutura ESCOLHA-CASO é uma opção muito elegante quando se tem muitos SE-ENTÃO-SENÃO para verificar a mesma variável. Facilita a leitura do algoritmo e a manutenção do código.

Exemplo:

Um determinado clube de futebol pretende classificar seus atletas em categorias e para isto ele contratou um programador para criar um programa que executasse esta tarefa. Para isso o clube criou uma tabela que continha a faixa etária

do atleta e sua categoria. A tabela está demonstrada abaixo:

IDADE CATEGORIA:

De 05 a 10 Infantil

De 11 a 15 Juvenil

De 16 a 20 Junior

De 21 a 25 Profissional

Resolução:

```
Algoritmo "CLASSIFICAÇÃO DE ATLETAS
var
nome, categoria  : caractere
idade : inteiro
inicio
Escreva("Nome do Atleta = ")
Leia (nome)
Escreva("Idade do Atleta = ")
Leia (idade)
Escolha idade
caso 5,6,7,8,9,10
   categoria <- "Infantil"
caso 11,12,13,14,15
   categoria <- "Juvenil"
caso 16,17,18,19,20
   categoria <- "Junior"
caso 21,22,23,24,25
   categoria <- "Profissional"
outrocaso
   categoria <- "INVALIDO"
Fimescolha
Escreva ("Categoria = ",categoria)
fimalgoritmo
```

Resumindo, usamos as decisões de múltipla escolha para: escolher apenas um conjunto de ações dentre vários alternativos. Aqui o teste não é mais uma operação lógica: o próprio valor de algum dado ou resultado anterior (que pode ser de outros tipos além do lógico) é que vai determinar qual desses conjuntos de ações será executado.

6.3. Exercícios Passo a Passo

1. Aplique a variável idade dentro do VisuAlg assim como o primeiro comando, necessário para que o usuário insira a idade do nadador, o programa armazene e escolha a opção.

2. Insira o comando responsável pelo primeiro caso.

3. Insira o comando responsável pelo segundo caso.

4. Insira o comando responsável pelo terceiro caso.

5. Insira o comando responsável pelo quarto caso.

6. Insira o comando responsável pelo quinto caso.

7. Insira o comando responsável por retornar na tela uma mensagem no caso do nadador não se enquadrar em nenhum dos casos anteriores.

8. Teste o programa, usando valores aleatórios.

6.4. Exercícios de Fixação

1. Desenvolva um algoritmo que dê ao usuário a possibilidade de digitar uma letra, sendo que se ele digitar a letra A será informado que ele digitou a letra A, se ele digitar a letra B, será informado que ele digitou a letra B. Caso o usuário insira outra letra qualquer, o programa deve retornar uma outra mensagem.

2. Desenvolva um algoritmo que diga se uma capital brasileira é da região Nordeste ou Sudeste, de acordo com a opção digitada pelo usuário. Em caso de o usuário digitar o nome de uma capital que não esteja entre as opções, escreva que a capital é de outra região.

3. Desenvolva um algoritmo que que leia um número de 1 a 5 e escreva por extenso. Caso o

usuário digite um número que não esteja neste intervalo, exibir a mensagem: número inválido.

4. Desenvolva um algoritmo que receba o número do mês e mostre o mês correspondente. Informe uma mensagem, caso o mês inserido seja invalido.

5. Desenvolva um algoritmo que dados três valores A, B e C, em que A e B são números reais e C é um caractere, pede-se para imprimir o resultado a operação de A por B se C for um símbolo de operador aritmético, caso contrário deve ser impressa uma mensagem de operador não definido. Tratar erro de divisão por zero.

anotações

7. Estrutura de repetição parte 1

Em todos os sistemas existirá sempre um momento em que uma determinada instrução deverá ser executada um número repetido de vezes. Para que essa repetição seja facilitada existem comandos que formam a Estrutura de Repetição.

Independentemente da linguagem de programação escolhida para o desenvolvimento do sistema, existem três tipos básicos Estrutura de Repetição: PARA-FAÇA, ENQUANTO-FAÇA e REPITA. A questão básica para quem está iniciando no mundo da programação de computadores é definir qual a diferença entre elas e quando utilizá-las. Aqui segue algumas dicas de quando e por que utilizar cada uma delas.

Sempre vale lembrar a seguinte regra da lógica de programação: "Sempre que uma condição é verdadeira, a linha de código imediatamente abaixo é executada". Isso fará diferença na hora de decidir a melhor Estrutura de Repetição.

Nesta aula, nosso foco ficará sobre a estrutura ENQUANTO.

7.1. ENQUANTO-FAÇA

Em nossos algoritmos, hora ou outra precisamos executar alguns passos mais de uma vez. Ou mesmo executar repetidamente alguns passos até que alguma condição seja atendida. A partir dessa necessidade surgem as estruturas de repetição, também conhecidas como **LOOP**. Nesta aula, vamos tratar de forma especial a estrutura de repetição **ENQUANTO** (em inglês, **WHILE**). Seu funcionamento é tão simples quanto a **estrutura de decisão SE-ENTÃO**. A diferença é que os passos dentro deste bloco, são repetidos enquanto a expressão booleana (VERDADEIRO ou FALSO) resultar VERDADEIRO.

Para entender na prática como usamos essa estrutura de repetição, vejamos um exemplo de algoritmo utilizando a estrutura:

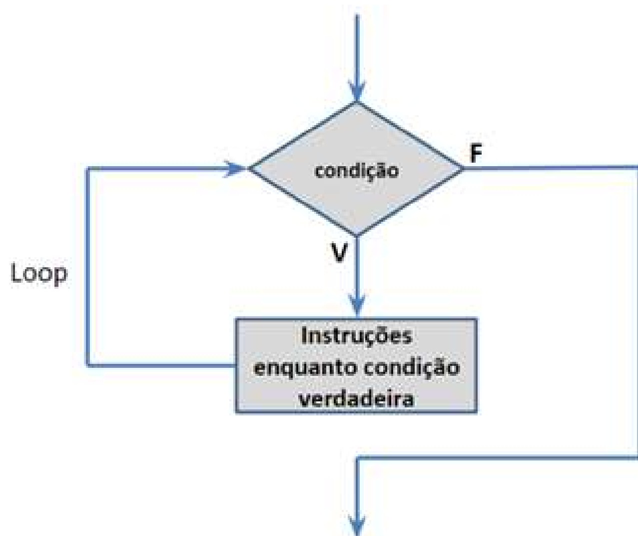
- Exemplo:

```
nota: real
escreva("Digite uma nota:")
leia(nota)
enquanto (nota < 0) ou (nota > 10) faca
    escreval("Erro! A nota deve ser entre 0 e 10")
    escreva("Digite novamente a nota: ")
    leia(nota)
fimenquanto
se (nota >= 7) entao
    escreval("Aluno aprovado!")
senao
    escreval("Aluno reprovado!")
fimse
```

Se não é possível determinar o número de repetições e o bloco de comandos não tem obrigatoriedade de execução então a melhor Estrutura de Repetição a ser utilizada é ENQUANTO-FAÇA. Essa estrutura valida a condição antes da execução e, se for verdadeira, executa o bloco de comandos em seu interior enquanto tal condição for verdadeira. Ao final da execução do bloco o ponteiro de execução do fluxo é colocado novamente no início do bloco e a condição é testada outra vez.

É importante assegurar que a condição testada possa ser modificada dentro bloco, passando de verdadeira para falsa, caso contrário o sistema entra em LOOP infinito, "travando" seu sistema.

Esta estrutura de repetição é também chamada de loop pré-testado, pois a expressão booleana é verificada antes da primeira execução. Se inicialmente ela já resultar em FALSO, as instruções que estão dentro do bloco não são executadas nenhuma vez.



As estruturas de repetição são muito utilizadas em desenvolvimento de softwares. Entender como funciona é muito importante para resolver problemas que precisam executar tarefas repetidas vezes.

Podemos dizer que temos dois tipos de repetição dentro de um programa, o primeiro seria a repetição programada. A repetição programada seria quando o programador já sabe previamente quantas vezes o comando deve ser repetido, ou seja, você sabe quantas vezes o programa deve executar determinada tarefa.

Mas é quando não sabemos quantas vezes um determinado comando deve ser repetido, ou quando queremos que um comando seja repetido até que um determinado número seja digitado ou quando temos alguma dessas duas ocorrências que devemos usar a estrutura de repetição enquanto.

Resumindo, a sua sintaxe e universal, seria:

```
Enquanto <expressão> Faça
        //bloco de comandos
Fim_enquanto
```

Aplicação real para a estrutura Enquanto:

```
Enquanto i <> 0 Faça
        Escreva "digite um numero"
        Leia i
Fim_enquanto
```

O código anterior irá executar enquanto o usuário digitar um número diferente de 0, ou seja, se ele digitar 0 a estrutura de repetição enquanto é encerrada.

7.2. Exercícios Passo a Passo

1. No VisuAlg, insira as variáveis do tipo inteiro.

2. Insira a variável do tipo real.

3. Aplique os comandos responsáveis por zerar as variáveis.

4. Insira o comando ENQUANTO dentro do algoritmo em desenvolvimento.

5. Determine que o usuário deva inserir um número e então o VisuAlg deve armazenar este número.

6. Insira o comando que realiza a conta dos valores.

7. Insira o comando que realiza a soma dos números.

8. Determine que o VisuAlg armazene a média na variável resul.

9. Determine que o VisuAlg retorne na tela o valor da média dos números digitados.

10. Teste o programa, inserindo valores

aleatórios.

7.3. Exercícios de Fixação

1. Desenvolva um algoritmo que apenas retorne na tela os números de 1 a 10 em ordem decrescente.

2. Supondo que a população de um país A seja da ordem de 90.000.000 habitantes com uma taxa anual de crescimento de 3% e que a população de um país B seja aproximadamente de 200.000.000habitantes com uma taxa anual de crescimento de 1,5%. Fazer um algoritmo que calcule e escreva o número de anos necessários para que a população do país A ultrapasse ou iguale a população do país B, mantidas essas taxas de crescimento.

3. Um determinado material radioativo perde metade de sua massa a cada 50 segundos. Dada amassa inicial, em gramas, fazer um programa que determine o tempo necessário para que essa massa se torne menor que 0,5 grama. Escreva a massa inicial, a massa final e o tempo calculado em horas, minutos e segundos.

4. Uma Universidade deseja fazer um levantamento a respeito de seu concurso vestibular. Para cada curso é fornecido o seguinte conjunto de valores: Um código do curso;

- Número de vagas;
- Número de candidatos do sexo masculino;
- Número de candidatos do sexo feminino.

Fazer um programa que:

- Calcule e escreva, para cada curso, o número de candidatos por vaga e a percentagem de candidatos do sexo feminino (escreva também o código correspondente do curso);
- Determine o maior número de candidatos por vaga e escreva esse número juntamente com o código do curso correspondente (supor que não haja empate);
- Calcule e escreva o total de candidatos.

5. Escreva um algoritmo que leia 10 valores inteiros e positivos e (Use a estrutura enquanto faça):

a) Encontre o maior valor.

b) Encontre o menor valor.

c) Calcule a média dos números inseridos.

anotações

8. Estrutura de repetição parte 2

Frequentemente precisamos implementar uma estrutura de repetição em nossos algoritmos para resolver algum problema. Um recurso para fazer isso é a estrutura ENQUANTO, foco da nossa aula anterior.

Nos exemplos e exercícios que vimos até agora sempre foi possível resolver os problemas com uma sequência de instruções onde todas eram necessariamente executadas uma única vez.

Os algoritmos que escrevemos seguiam, portanto, apenas uma sequência linear de operações.

Por exemplo, um algoritmo para ler os nomes e as notas das provas de três alunos de uma escola qualquer e calcular suas médias finais. Uma possível solução seria repetir o trecho de código do algoritmo três 3 seria repetir o trecho de código do algoritmo três vezes.

A solução dada é viável apenas para uma turma de poucos alunos; para uma turma de 50 alunos, a codificação da solução seria por demais trabalhosa.

Agora, veremos um conjunto de estruturas sintáticas que permitem que um trecho de um algoritmo (lista de comandos) seja repetido um determinado número de vezes, sem que o código correspondente tenha que ser escrito mais de uma vez.

8.1. REPITA-ATÉ

Uma estrutura de repetição é utilizada quando um trecho do algoritmo ou até mesmo o algoritmo inteiro precisa ser repetido. O número de repetições pode ser fixo ou estar atrelado a uma condição.

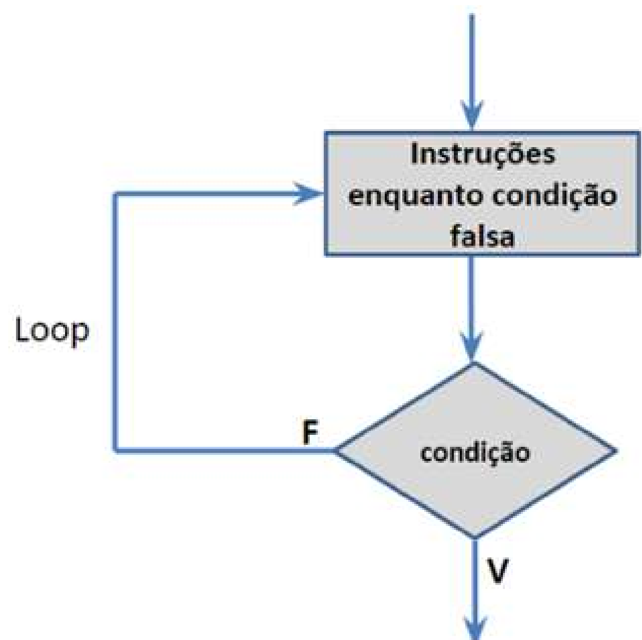
Essa estrutura lembra a estrutura enquanto, porém, executa o conjunto de instruções programado ao menos uma vez antes de verificar a condição testada no laço enquanto, a condição é testada *antes* de executar o bloco de códigos, o que significa que há a chance desse bloco nunca ser executado se a condição de teste retornar falso logo no primeiro teste.

Com o REPITA ATÉ o conjunto de instruções é executado enquanto a condição testada retornar Falso.

Sintaxe:

```
repita
   Instruções executadas enquanto condição  falsa
até (condição seja verdadeira)
```

O fluxograma a seguir ilustra funcionamento da estrutura de repetição repita até:



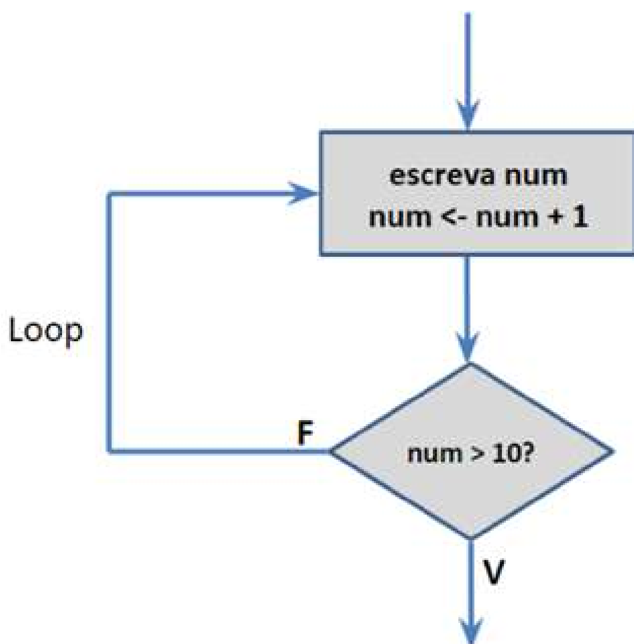
Exemplo:

Imprimir na tela os números de 1 a 10, agora usando estrutura de repetição "repita...até"

(código para o VisualG):

```
Var
num : inteiro
Inicio
num <- 1
repita
   escreval (num)
   num <- num + 1
ate (num > 10)
Fimalgoritmo
```

A seguir temos o fluxograma que mostra o funcionamento deste exemplo mostrado:



Perceba, que além de ser pós-testada, esta estrutura testa o contrário do ENQUANTO. Na estrutura **REPITA-ATÉ**, as instruções do bloco são executadas repetidamente enquanto a expressão booleana resultar FALSO. A partir do momento que a expressão booleana resultar

Vejamos a seguir um exemplo de um algoritmo, aonde vamos resolve-los de duas formas. Primeiramente vamos solucioná-lo usando a estrutura ENQUANTO e posteriormente a estrutura REPITA-ATÉ. Seguem os códigos:

```
algoritmo "SomaEnquantoValorDiferenteDe0"
var
   valorDigitado : REAL
   soma : REAL
inicio

      soma := 0
      ESCREVA ("Digite um valor para a soma: ")
      LEIA (valorDigitado)

      ENQUANTO valorDigitado <> 0 FACA
           soma := soma + valorDigitado
           ESCREVAL ("Total: ", soma)
           ESCREVA ("Digite um valor para a soma: ")
           LEIA (valorDigitado)
      FIMENQUANTO

      ESCREVAL ("Resultado: ", soma)

fimalgoritmo
```

Veja que a leitura de dados é escrita duas vezes neste algoritmo, o motivo para fazer isso é que a estrutura ENQUANTO é pré-testada. Logo, não dá pra testar se o usuário digitou o valor 0 se ele ainda não tiver digitado valor nenhum. Com a estrutura de repetição REPITA-ATÉ não é necessário escrever duas vezes a leitura de dados do usuário, pois ela é pós-testada. Observe a implementação daquele algoritmo com REPITA-ATÉ.

```
algoritmo "SomaAteValorIgualA0"
var
   valorDigitado : REAL
   soma : REAL
inicio

      soma := 0

      REPITA
           ESCREVA ("Digite um valor para a soma: ")
           LEIA (valorDigitado)
           soma := soma + valorDigitado
           ESCREVAL ("Total: ", soma)
      ATE valorDigitado = 0

fimalgoritmo
```

Podemos observar que o teste mudou de (**valorDigitado <> 0**) na estrutura ENQUANTO, para (**valorDigitado = 0**) na estrutura REPITA-ATÉ.

O resultado deste algoritmo pode ser observado abaixo:

```
Digite um valor para a soma: 4
Total:  4
Digite um valor para a soma: 2
Total:  6
Digite um valor para a soma: 8
Total:  14
Digite um valor para a soma: 12
Total:  26
Digite um valor para a soma: 0
Total:  26

*** Fim da execução.
*** Feche esta janela para retornar ao Visualg.
```

Percebemos que é possível utilizar qualquer uma das duas estruturas para implementar LOOPs, porém cada uma é mais apropriada dependendo do problema. Neste problema em particular, a estrutura REPITA-ATÉ se mostrou mais apropriada. Mas essa decisão de qual utilizar entre as duas, sempre será tomada observando a diferença entre PRÉ-TESTADA e PÓS-TESTADA.

8.2. PARA-FAÇA

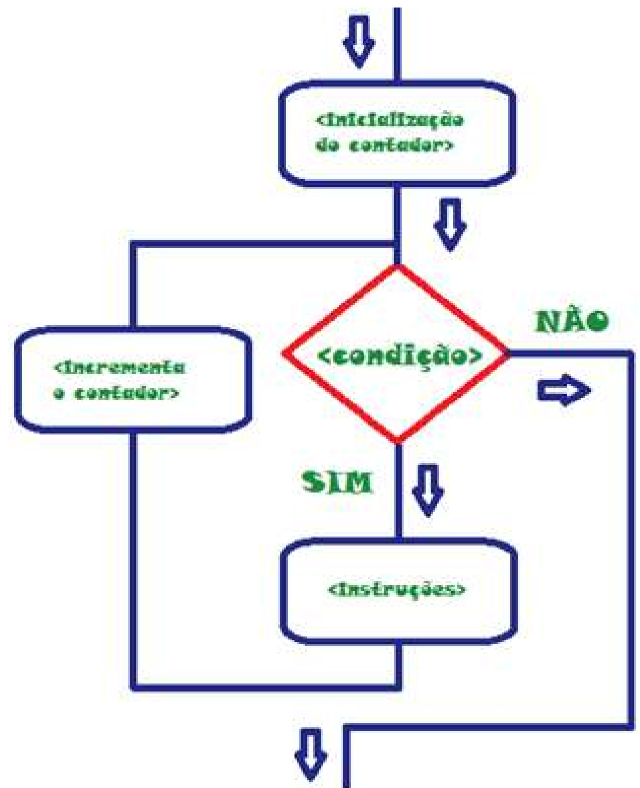
Exemplo, um algoritmo que realiza a soma dos números de 1 a 100, terá um número de iterações pré-definido (100).

Podemos implementar esse LOOP com qualquer estrutura de repetição, mas para isso é necessário utilizar um contador, uma variável que será utilizada para contar quantas iterações foram executadas até o momento A estrutura de repetição PARA, implementa um contador implicitamente.

A estrutura de repetição **PARA-FAÇA** é usada quando se sabe o número exato de vezes de repetição do looping e esse número é um valor inteiro. Veja como é o seu esquema:

```
PARA <variável contadora> DE <valor inicial> ATE <valor final> [PASSO <valor de incremento>] FAÇA
<instruções a serem executadas repetidamente até a <variável contadora> atingir o valor final>
FIM-PARA
```

O passo de incremento é opcional, esse recurso serve pera definir qual o valor do incremento do contador, por exemplo de 1 em 1 (padrão), de 2 em 2, de 3 em 3, etc. Essa estrutura de repetição realiza o incremento de um contator de forma implícita, vejamos graficamente como funciona.



A inicialização da variável contadora é realizada implicitamente, com o informado da declaração da estrutura PARA. A condição para executar a iteração é que o valor da variável contadora não tenha atingido o . E ao final de cada iteração, o valor da variável contadora é incrementado em 1 (ou o valor declarado como PASSO ou).

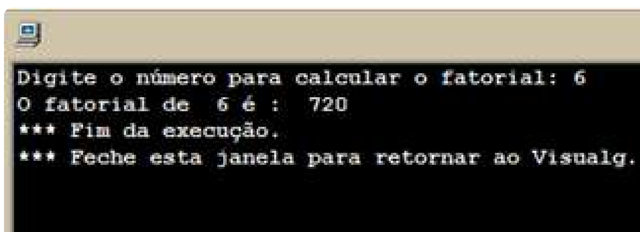
Vamos implementar como exemplo um algoritmo para calcular o fatorial de um número. Para quem não sabe, fatorial é a multiplicação de todos os números de 1 até ao número que se está calculando. Por exemplo: Fatorial de 5 (5!) = 1 * 2 * 3 * 4 * 5 = 120. Vamos criar um algoritmo utilizando o ENQUANTO primeiro.

```
algoritmo "FatorialComENQUANTO"

var
   numero : INTEIRO
   fatorial : INTEIRO
   contador : INTEIRO
inicio

      ESCREVA ("Digite o número para calcular o fatorial: ")
      LEIA (numero)

      fatorial := 1
      contador := 1
      ENQUANTO contador <= numero FACA
          fatorial := fatorial * contador
          contador := contador + 1
      FIMENQUANTO

      ESCREVA ("O fatorial de ", numero, " é : ", fatorial)

fimalgoritmo
```

Veja que foi necessário incrementar o contador explicitamente. Com a estrutura de repetição PARA, isso não é necessário. Vejamos agora o mesmo algoritmo implementado com o PARA.

```
algoritmo "FatorialComPARA"

var
   numero : INTEIRO
   fatorial : INTEIRO
   contador : INTEIRO
inicio

      ESCREVA ("Digite o número para calcular o fatorial: ")
      LEIA (numero)

      fatorial := 1
      PARA contador DE 1 ATE numero FACA
          fatorial := fatorial * contador
      FIMPARA

      ESCREVA ("O fatorial de ", numero, " é : ", fatorial)

fimalgoritmo
```

Nesta estrutura, não é necessário incrementar nem inicializar o contador, isso é feito automaticamente. O resultado dos dois algoritmos é o mesmo.



LOOPs podem ser implementados com qualquer estrutura de repetição, porém, em alguns casos uma estrutura se mostra mais adequada do que outras, como nesse caso do fatorial a mais adequada é a estrutura PARA. Conhecer essas estruturas de repetição é muito importante para criar programas melhores.

8.3. Exercícios Passo a Passo

1. No programa VisuAlg, determine as variáveis do tipo inteiro.

2. Insira o comando necessário para que o usuário digite o número inicial, assim como para que o programa armazene este número.

3. Insira o comando necessário para que o usuário digite o número final, assim como para que o programa armazene este número.

4. Insira o comentário antes da próxima linha de comando.

5. Determine o primeiro teste lógico do nosso algoritmo.

6. Insira a mensagem que o programa deve retornar, em caso de o usuário inserir um valor final, menor do que o inicial.

7. Insira o segundo teste lógico do programa.

8. Insira o comentário referente a próxima linha de comando.

9. Determine o último teste lógico do nosso algoritmo.

10. Determine que o programa retorne na tela os valores.

11. Teste o algoritmo desenvolvido.

8.4. Exercícios de Fixação

1. Faça um programa em que o usuário digite diversos números positivos. Se digitar um número negativo o programa termina.

2. Escreva um programa que lê o sexo de uma pessoa. O sexo deverá ser com o tipo de dado caractere o programa deverá aceitar apenas os valores “M” ou “F”.

3. Escrever um programa de computador que leia 10 números inteiros e, ao final, apresente

a soma de todos os números lidos.

4. Faça um programa em que o usuário digite 2 valores e se a soma deles for maior que 15 o programa encerra, caso contrário, repete.

5. Escreva um programa que leia dois valores reais. Ambos valores deverão ser lidos até que o usuário digite um número no intervalo de 1 a 100. Apresentar a soma dos dois valores lidos.

6. Faça um programa que escreva de 30 a 50 contando de 2 em 2.

7. Faça um programa que escreva de 80 a 40, em ordem decrescente de 5 em 5.

anotações

9. Manipulação de vetores

Vetores e Matrizes são estruturas de dados muito simples que podem nos ajudar muito quando temos muitas variáveis do mesmo tipo em um algoritmo. Imagine o seguinte problema: Você precisa criar um algoritmo que lê o nome e as 4 notas de 50 alunos, calcular a média de cada aluno e informar quais foram aprovados e quais foram reprovados.

Conseguiu imaginar quantas variáveis você vai precisar? Muitas né? Vamos fazer uma conta rápida: 50 variáveis para armazenar os nomes dos alunos, (4 * 50 =) 200 variáveis para armazenar as 4 notas de cada aluno e por fim, 50 variáveis para armazenar as médias de cada aluno. 300 variáveis no total, sem contar a quantidade de linhas de código que você vai precisar para ler todos os dados do usuário, calcular as médias e apresentar os resultados. Mas temos uma boa notícia pra você. Nós não precisamos criar 300 variáveis! Podemos utilizar Vetores e Matrizes (também conhecidos como ARRAYs)!

Nesta aula, nosso foco será na utilização dos Vetores.

9.1. VETOR

Vetor (**array** uni-dimensional) é uma variável que armazena várias variáveis do mesmo tipo. No problema apresentado anteriormente, nós podemos utilizar um vetor de 50 posições para armazenar os nomes dos 50 alunos.

É um conjunto de variáveis do mesmo tipo acessíveis com um único nome. Armazenadas de forma contínua e ocupando as posições de forma fixas. Pode-se dizer por este motivo que vetor é uma matriz unidimensional.

Até agora você manipulou dados num algoritmo usando variáveis e constantes simples.

O que isso significa? Que para cada dado a ser usado num algoritmo foi declarada uma variável (ou uma constante) para armazenar a informação.

Essa ação é válida para problemas de pequeno porte, que envolvem poucas variáveis e constantes. Mas como lidar com um problema que precise manipular dezenas, centenas, milhares de variáveis? Declarar uma variável para cada dado seria muito trabalhoso e seu controle no código do algoritmo ainda mais. Uma das maiores dificuldades em escrever um grande programa de computador não se resume em apenas estabelecer suas metas, os retornos que deverá apresentar, nem mesmo quais métodos serão usados para que os objetivos sejam atingidos, mas sim em como os dados serão organizados internamente e quais as estruturas de dados mais adequadas para serem utilizadas.

Em programação, as estruturas de dados são uma forma coerente e racional de organizar os dados e otimizar o seu uso. Um vetor unidimensional, ou um array, é uma das estruturas de dados mais comuns a serem aplicadas no desenvolvimento de um algoritmo e pode ser considerado uma sequência de posições de memória que é usada para armazenar um conjunto de dados do mesmo tipo e usando o mesmo nome. Para acessar cada informação armazenada, separadamente, usa-se um índice que é representado por uma variável inteira.

Essas informações são chamadas de elementos do vetor.

Considere uma sala de aula, onde há 10 alunos e queremos armazenar suas idades em variáveis, até então seria feito assim:

Var

idade1, idade2, idade3, idade4, idade5, idade6, idade7, idade8, idade9, idade10:

INTEIRO.

Como sabemos, podemos inicializar uma variável de duas formas:

idade1= 10, idade2 = 12, idade3 = 11 ... idade10 =10

Ou pedir que o usuário escreva usando o teclado:

Escreva ("Digite a idade do primeiro aluno")

Leia (aluno1)

Escreva ("Digite a idade do segundo aluno")

Leia (aluno2)

...

Não foi tão difícil criar uma variável que armazena a idade de cada aluno, porém, se formos fazer diversos cálculos com as idades ou ainda considerar uma sala com 1000 alunos, já ficaria muito mais difícil. O vetor é uma estrutura que simplifica essas operações com variável do mesmo tipo, considerando que toda idade é do tipo inteiro, criamos um vetor de 10 posições para seguir o mesmo exemplo usado à cima.

idade:vetor[1..10] de inteiro

Na declaração usamos o tamanho máximo que o vetor pode ter, podendo usar o tamanho total declarado ou menos.

```
algoritmo "vetor"
var
n:inteiro
j:inteiro
valor:inteiro
idade:vetor[1..10] de inteiro

inicio
Escreva("Digite a quantidade de alunos na sala")
Leia(n)

para i de 1 ate n faca
 leia(valor)
 aluno[i] <- valor
fimpara

fimalgoritmo
```

Serão armazenadas todas as idades dos respectivos alunos e assim poderão ser acessadas separadamente. Vem um questionamento: "é quase a mesma dificuldade para acessar em comparação à estrutura simples". Sim, dessa forma foi, porém será mostrado agora algo bem característico da estrutura vetorial que simplifica as operações com a variável.

São usadas duas variáveis do tipo inteiro para auxiliar na inserção de elementos dentro do vetor. Vamos considerar aqui a variável "n" que armazenará o tamanho que nosso vetor tiver, e a variável "i" que será usada com o auxílio da estrutura de repetição "para" ou "for" que acessará cada variável de forma mais fácil e rápida como demonstrado à baixo:

Usando a variável idade[3] onde i=3, poderemos acessar a idade do 3 aluno, a variável idade[2] onde i=2, acessará a idade do 2 aluno, e assim para todos os outros, assim como já foi demostrado.

nota

60	70	90	60	75	91	100	50	78	80
0	1	2	3	4	5	6	7	8	9

Consideramos o fato que precisamos fazer uma média de todos os alunos dentro da sala de aula. Usando as variáveis de tipo simples e considerando que todas as 10 variáveis já tenham sido inicializadas com as idades dos respectivos alunos.

DECLARE media NUMERICO; // Essa variável armazenará valores numéricos

media <- media +(idade1, idade2,idade3, idade4,idade5, idade6, idade7, idade8, idade9, idade10)/10;

Usando vetores

media<- 0; // a variável media precisa ser inicializa com zero Para i=1; i<=n ; i++ faça // percorrendo todas as idades dos n alunos. media<- media +idade[i]; // aqui será somente a soma de todas as idades Fim media<-media/n ; // aqui será de fato calculado e media.

Como visto até agora, vetores são uma estrutura que simplifica as operações com

variáveis do mesmo tipo, trazendo um novo conceito que é bastante usado em quase todos os programas.

Repare que os arrays (vetores) aliados a estrutura de repetição PARA é um ótimo recurso para algoritmos que precisam de muitas variáveis do mesmo tipo.

As observações mais importantes que você precisa saber sobre vetores:

- Os elementos de um vetor são sempre armazenados em posições adjacentes de memória.
- Os elementos de um vetor que foi declarado, mas ao qual os dados ainda não foram atribuídos, conterão valores aleatórios que já estavam na memória do computador.
- O índice do primeiro elemento do vetor pode ser "0" ou "1", a depender da linguagem de programação usada.
- Cada linguagem de programação trabalha de forma distinta com relação à atribuição de dados, à declaração e ao dimensionamento do vetor.
- É muito importante observar se a manipulação do índice do vetor está correta, ou seja, se o elemento correto do vetor está sendo acessado. Esse tipo de erro não é detectável pelos compiladores das linguagens de programação e são difíceis de identificar.
- Ao declarar uma variável como vetor de X elementos de determinado tipo de dado, o compilador reservará espaço em memória para o tipo de dado declarado. Ou seja, se forem X dados reais, haverá necessidade de mais espaço em comparação com X dados inteiros. Por isso é fundamental declarar corretamente o tipo de dado e a quantidade de elementos de um vetor. Volte ao tema 3 para ver o espaço em memória de que cada tipo de dado necessita.

A estrutura de repetição mais adequada para manipular dados de um vetor é o laço "para".

Array é uma das estruturas de dados mais simples que existe e uma das mais utilizadas também. Em basicamente todas as linguagens de programação têm **arrays**, pelo menos ainda não conhecemos uma linguagem que não tem.

Porém, os índices podem mudar dependendo da linguagem, algumas começam os índices do array com 1 e outras com 0, essa é a grande diferença que geralmente encontramos entre linguagens. No caso das linguagens que começam os arrays com o índice 0, o último elemento do array recebe o índice (- 1).

9.2. Exercícios Passo a Passo

1. No VisuAlg, crie um programa que solicite a entrada de 10 números pelo usuário, armazenando-os em um vetor, e então monte outro vetor com os valores do primeiro multiplicados por 5. Exiba os valores dos dois vetores na tela, simultaneamente, em duas colunas (um em cada coluna), uma posição por linha. Comece inserindo as variáveis de vetores.

2. Insira a variável cont determinando o tipo de dado como inteiro.

3. Insira primeiro laço PARA-FAÇA além de determinar que o usuário insira um número e que o sistema armazene este número.

4. Determine os valores que a coluna 2 deverá receber.

5. Conclua o primeiro comando.

6. Determine que o sistema retorne o resultado na tela.

7. Insira o segundo laço e o teste lógico referente a este laço.

8. Conclua o programa.

9. Realize o teste do algoritmo criado.

9.3. Exercícios de Fixação

1. Desenvolva um algoritmo que exibe os números digitados em posições impares.

2. Desenvolva um algoritmo que aceita 5 nomes e exibe-os em ordem inversa.

3. Desenvolva um algoritmo que realiza 5 cadastros diferentes.

4. Construa um algoritmo que leia 50 valores inteiros e positivos e:

a) Encontre o maior valor

b) Encontre o menor valor

c) Calcule a média dos números lidos

5. Utilizando vetores, desenvolva um algoritmo que aceite e exiba:

controle de sorveteria com 10 tipos de sorvetes:

ordem | sabor | descrição

anotações

10. Manipulação de matrizes

Você estudou até o momento os arrays de uma única dimensão, ou seja, vetores que são graficamente representados por uma linha ou uma coluna. Mas também é possível trabalhar com arrays de duas ou mais dimensões. O array bidimensional, ou matriz, é muito conhecido e utilizado em desenvolvimento de algoritmos, sendo formado por linhas e colunas, como numa tabela.

10.1. Matriz

Assim como os vetores, as matrizes também são variáveis compostas homogêneas.

Composta por ser uma variável que contém um número finito de dados e homogênea porque todos estes são do mesmo tipo. Uma matriz é considerada quadrada quando possui o mesmo número de linhas e de colunas.

Matriz quadrada a_{7x7}, com sete linhas e sete colunas.

colunas	1	2	3	4	5	6	7
linha 1	5	9	6	4	3	8	7
linha 2	4	1	3	5	6	8	7
linha 3	3	1	4	5	6	8	7
linha 4	1	3	4	5	6	8	7
linha 5	1	3	4	5	6	8	7
linha 6	1	3	4	5	2	7	8
linha 7	1	3	4	5	6	7	8

Da mesma forma como ocorre com os vetores, cada elemento de uma matriz é acessado por meio da variável declarada como matriz seguida da posição que ocupa no conjunto.

A matriz apresentada na imagem anterior foi declarada como “a” e cada elemento foi atribuído em uma posição específica, que é identificada pela linha e pela coluna da matriz. Essa posição é obtida por meio de índices que auxiliam a manipular os dados da variável. Por exemplo, o elemento a[1,2] é o valor “9”, destacado em na representação da imagem.

O primeiro índice indica a linha e o segundo, a coluna, e essa ordem nunca se altera, ou seja, a linha sempre deve ser indicada em primeiro lugar, e depois se indica a coluna da matriz para manipular algum dado. Você consegue identificar a posição ocupada pelo valor “2” em destaque na matriz da imagem anterior? Está na linha “6” e na coluna “5”, então a sintaxe correta para manipular esse dado é a[6,5] = 2.

Observe o código apresentado na tabela abaixo que mostra como declarar matriz “a” e como atribuir seus valores.

Programa que atribui valores para a matriz “a”.

```
ALGORITMO
   a vetor[1..7, 1..7] de inteiro          // declaração de matriz
   i,j inteiro                              // declaração dos índices
   para i de 1 ate 7 passo 1 faca          // variação da linha da matriz
      para j de 1 ate 7 passo 1 faca       // variação da coluna da matriz
         Escreva("Digite o elemento da matriz na posição", i, j)
         Leia(a[i,j])
      fimpara
   fimpara
FIMALGORITMO
```

A variável “a” é declarada como um vetor bidimensional (matriz) com 49 posições de dados do tipo inteiro. Essa quantidade de elementos é obtida pela multiplicação da dimensão da linha pela dimensão da coluna. Na declaração, assim como na manipulação dos dados na matriz, primeiro é indicada a dimensão da linha e depois, a da coluna.

Considere que “m” seja o número de elementos nas linhas e “n” seja a dimensão da coluna. Nesse exemplo “m=n=7”, uma matriz quadrada, com 7*7 = 49 posições. Então, tem-se que “a” é um vetor[1..m, 1..n] de inteiros. Outro aspecto bastante importante que você deve observar é com relação à variação dos índices da matriz.

Na tabela anterior, o laço mais externo é a variação do índice "i", que significa a linha da matriz que está sendo manipulada, e o laço mais interno varia "j", que é o índice que controla as colunas.

Como você já estudou, ao executar uma estrutura de repetição, todos os seus comandos internos são efetuados para que então a variável de controle seja atualizada e verificada para a condição de parada. Como o primeiro laço é "i", isso significa que, quando "i=1", o comando interno a esse laço "para j" será executado até finalizar, para então a variável de controle "i" ser atualizada novamente.

Ou seja, quando "i=1", a variável de controle "j" assumirá os valores 1, 2, 3, 4, 5, 6 e 7, lendo os valores de a[1,1], a[1,2], a[1,3], a[1,4], a[1,5], a[1,6] e a[1,7]. Observe que esses são os elementos que estão na primeira linha da matriz. Fixou-se a linha e variaram-se as colunas. Após finalizar a variação das colunas, atualiza-se o índice "i =2" e novamente há a variação das colunas, obtendo-se então a[2,1], a[2,2], a[2,3], a[2,4], a[2,5], a[2,6] e a[2,7]. E assim sucessivamente até a última linha da matriz.

Matriz quadrada a_{7x7} – Fixa linha (i) e varia coluna (j)

j =	1	2	3	4	5	6	7
i = 1 →	5	9	6	4	3	8	7
i = 2	4	1	3	5	6	8	7
i = 3	3	1	4	5	6	8	7
i = 4	1	3	4	5	6	8	7
i = 5	1	3	4	5	6	8	7
i = 6	1	3	4	5	2	7	8
i = 7	1	3	4	5	6	7	8

Mas se, em vez de fixar a linha para então variar a coluna, fosse feito o contrário? Seria válido? Veja na Tabela a seguir:

Programa que atribui valores para a matriz "a" coluna a coluna.

```
ALGORITMO
  a vetor[1..7, 1..7] de inteiro        // declaração de matriz
  i,j inteiro                            // declaração dos índices
  para j de 1 ate 7 passo 1 faca         // variação da coluna da matriz
     para i de 1 ate 7 passo 1 faca      // variação da linha da matriz
        Escreva("Digite o elemento da matriz na posição", i, j)
        Leia(a[i,j])
     fimpara
  fimpara
FIMALGORITMO
```

Veja que a única mudança foi colocar o índice da coluna da matriz no laço mais externo e o da linha no mais interno. Isso fixará a coluna "j = 1" enquanto as linhas serão variadas "i = 1, 2, 3, 4, 5, 6 e 7", lendo-se agora os valores da primeira coluna: a[1,1], a[2,1], a[3,1], a[4,1], a[5,1], a[6,1] e a[7,1]. E assim sucessivamente até a última coluna da matriz abaixo:

Matriz quadrada a_{7x7} – Fixa coluna (j) e varia linha (i)

j =	1 ↓	2	3	4	5	6	7
i = 1	5	9	6	4	3	8	7
i = 2	4	1	3	5	6	8	7
i = 3	3	1	4	5	6	8	7
i = 4	1	3	4	5	6	8	7
i = 5	1	3	4	5	6	8	7
i = 6	1	3	4	5	2	7	8
i = 7	1	3	4	5	6	7	8

Compreender essa manipulação de índices em matrizes é muito importante para o desenvolvimento correto de muitos métodos matemáticos que são usados em alguns algoritmos. Considere a necessidade de multiplicar os elementos de duas matrizes. Como essa operação seria realizada? Qual a primeira análise a ser realizada para verificar a validade do cálculo? Suponha uma matriz Amxn e uma Bzxv, sendo "m" a dimensão da linha de A e "n" a de sua coluna, assim como "z" e "v" em B, respectivamente.

A multiplicação só será possível se "n" for igual a "z", n = z, porque a multiplicação de matrizes é feita por meio da multiplicação dos

elementos da linha de A pelos elementos da coluna de B, somando-se seu resultado para obter um elemento da matriz resultante. Dessa forma, a quantidade de elementos na linha de A deve ser igual à quantidade de elementos na coluna de B. E, para essa operação, é fundamental observar a correta variação dos índices das matrizes para que os elementos certos sejam manipulados. Nesse exemplo, a matriz resultante (nesse caso, matriz C) terá dimensão mxv, isto é, "m" linhas e "v" colunas.

As observações mais importantes que você precisa saber sobre matrizes:

- Arrays são muito parecidos com variáveis simples, exceto pela possibilidade que têm de poder armazenar múltiplos elementos do mesmo tipo.
- Os elementos de uma matriz que foi declarada, mas à qual os dados ainda não foram atribuídos, conterão valores aleatórios, que já estavam na memória do computador.
- Num array, os elementos são armazenados em posições adjacentes de memória.
- Da mesma forma como ocorre com um array unidimensional, ao declarar uma variável como uma matriz de X elementos de determinado tipo de dado, o compilador reservará espaço em memória para o tipo de dado declarado. Ou seja, se forem X dados reais, haverá necessidade de mais espaço se comparados com X dados inteiros. Por isso é fundamental declarar corretamente o tipo de dado e a quantidade de elementos de uma matriz. Volte ao tema 3 para ver o espaço em memória de que cada tipo de dado necessita.
- A estrutura de repetição mais adequada para manipular dados de uma matriz é o uso de laços "para" aninhados.

10.2. Exercícios Passo a Passo

1. No VisuAlg, faça um algoritmo para ler uma matriz de 3x4 de números reais e depois exibir o elemento do canto superior esquerdo e do canto inferior direito. Insira a variável matriz.

2. Insira as variáveis i e j de tipo de dados inteiro.

3. Insira o comando para o início do primeiro laço.

4. Insira o comando para sequência do laço.

5. Determine que o usuário deva inserir um número inteiro.

6. Faça com que o VisuAlg armazene estes dados na matriz.

7. Finalize o primeiro laço.

8. Insira o comando para o início do segundo laço.

9. Insira o comando para a sequência do laço.

10. Determine a continuidade do comando, usando escreva para exibir a matriz.

11. Conclua o laço.

12. Determine que o algoritmo retorne o primeiro valor.

13. Determine que o algoritmo retorne o segundo valor.

14. Teste o algoritmo criado.

10.3. Exercícios de Fixação

1. Ler uma matriz 5X5 e gerar outra em que cada elemento é o cubo do elemento respectivo na matriz original. Imprima depois o elemento do meio desta nova matriz.

2. Faça um algoritmo para ler uma matriz 2X3 real e depois gerar e imprimir sua transposta

(matriz 3X2 equivalente).

3. Crie uma matriz 7X8 onde cada elemento é a soma dos índices de sua posição dentro da matriz.

4. Ler uma matriz 4X3 real e imprimir a soma dos elementos de uma linha L fornecida pelo usuário.

5. Ler uma matriz 4X3 real. Depois, mostre qual é o elemento armazenado em uma linha e coluna C fornecidos pelo usuário.

anotações

11. Funções e procedimentos

Aula 11

Vamos tratar de um assunto de extrema importância na programação estruturada: as **Funções** e **Procedimentos**. Sem eles, a programação seria praticamente impossível de ser realizada à medida que os programas crescessem em tamanho. Além disso, seria praticamente impossível trabalhar em equipe, e a reutilização de código também seria inimaginável, ou seja, toda vez que escrevêssemos nossos programas, teríamos que fazer tudo a partir do zero.

A programação estruturada é uma forma de programação de computadores que estabelece uma disciplina de desenvolvimento de algoritmos, independentemente da sua complexidade e da linguagem de programação na qual será codificado, que facilita a compreensão da solução através de um número restrito de mecanismos de codificação.

É comum encontrar-se nas linguagens de programação, várias funções embutidas, por exemplo, sin (seno), cos (co-seno), abs (valor absoluto), sqrt (raiz quadrada).

Funções embutidas podem ser utilizadas diretamente em expressões, essas funções são utilizadas em expressões como se fossem simplesmente variáveis comuns como variáveis comuns, as funções têm (ou retornam) um único valor

Já em algumas situações desejamos especificar uma operação que não é convenientemente determinada como parte de uma expressão Nesses casos, utilizamos outra forma de sub-algoritmo: o procedimento.

Veremos como constitui-se uma função/procedimento, as diferenças de um para o outro, o que é um argumento e um parâmetro, diferenças entre um e outro, o que é um retorno.

Para começarmos a falar sobre procedimentos e funções, vamos dar uma olhada no conceito da **máquina de von Neumann**, onde os sistemas computacionais são constituídos de:

ENTRADA -> **PROCESSAMENTO** -> **SAÍDA**

Através desse conceito, podemos generalizar que todo programa recebe uma entrada de **Dados**, realiza seus cálculos no processamento e, como saída, temos a **Informação** que é o dado processado. Da mesma forma, os procedimentos e funções recebem dados de entrada, processam-os e retornam algo para quem os chamou. Podemos começar diferenciando um procedimento de uma função pela maneira como é feito o **retorno**.

11.1. Funções

As **funções** (**functions**), também conhecidas como sub-rotinas, são muito utilizadas em programação. Um dos grandes benefícios é não precisar copiar o código todas as vezes que precisar executar aquela operação, além de deixar a leitura do código mais intuitiva.

Na **Função**, o valor de retorno é **Explícito**, de maneira simplificada, podemos dizer que uma função recebe de quem a chamou, em seus **Parâmetros**, os dados a serem processados.

Já quando chamamos uma função, passamos **Argumentos** para que ela execute o seu processamento. Dessa forma, os **Parâmetros** são os dados aguardados pela função, enquanto que os **Argumentos** são os dados que passamos à função.

Podemos dizer que as funções são **pequenos programas**. Ou seja, a partir de agora, podemos "quebrar" o nosso programa em partes menores. Dessa forma, podemos, dentre outras coisas, reutilizar funções já escritas em programas novos, dividir trabalho entre uma equipe, onde

cada um escreve um trecho do código, ou seja, cada um escreve suas funções e depois junta-se em um único programa.

11.2. Procedimentos

Os **procedimentos** (**procedures**) diferem das funções apenas por não retornarem resultado, imagine um procedimento que envia e-mail. Precisa retornar resultado?

Diferente das funções no **Procedimento**, o retorno é realizado de maneira **Implícita**. De resto, é praticamente igual a uma função.

Antes de nos aprofundarmos melhor, em funções e procedimentos, devemos relembra um pouco sobre o conceito de **Variáveis Globais** e **Variáveis Locais**.

11.3. Variáveis Globais e Locais

O conceito de Variáveis, já foi explorado neste curso (variável é uma célula de memória que armazena um valor, toda variável precisa de um tipo, etc).

Uma **Variável Global** é uma variável que pode ser acessada, alterada, enfim, ela pode ser vista por todo o programa. Como assim? As variáveis são todas globais então? Bom, nos programas que fizemos até agora, sim. A partir de agora, veremos que os procedimentos e funções também possuem suas próprias variáveis, que só são conhecidas dentro de seu escopo (dentro do procedimento ou função). Ou seja, uma **Variável Local** é uma variável que só é conhecida por um trecho do programa, não podendo ser acessada fora dele.

Por exemplo, declaramos uma variável chamada **X** dentro de uma função. Se você mencioná-la fora da função onde foi declarada, o compilador em questão irá acusar um erro, dizendo que a variável não foi declarada. Com isso, você pode perceber que podemos ter variáveis de mesmo nome em nosso programa, porém cada uma dentro de seu escopo, sem nenhuma ligação uma com a outra.

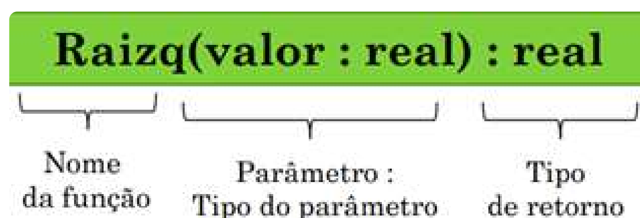
Então vamos ver alguns exemplos de como fazê-los.

11.4. Funções Pré-Definidas

A linguagem do Visualg possui diversas funções predefinidas que podem ser usadas na construção de algoritmos.

Exemplo:

Assinatura da função para cálculo da Raiz Quadrada:



Exemplo de algoritmo usando uma função pré-definida:

- **Funções predefinidas**
  - Como utilizar

```
algoritmo "Uso_de_Funcao_Predefinida"
var
   numero, raiz: real
inicio
   escreva("Digite um número: ")
   leia(numero)
   raiz <- Raizq(numero)
   escreva("A raiz quadrada do número digitado é: ", raiz)
fimalgoritmo
```

O VisuAlg possui uma lista extensa de funções pré-definidas em sua biblioteca.

11.5. Funções Numéricas, Algébricas e Trigonométricas

Abs(expressão) - Retorna o valor absoluto de uma expressão do tipo inteiro ou real. Equivale a **| expressão |** na álgebra.

ArcCos(expressão) - Retorna o ângulo (em radianos) cujo co-seno é representado por expressão.

ArcSen(expressão) - Retorna o ângulo (em radianos) cujo seno é representado por

expressão

ArcTan(expressão) - Retorna o ângulo (em radianos) cuja tangente é representada por expressão.

Cos(expressão) - Retorna o co-seno do ângulo (em radianos) representado por expressão.

CoTan(expressão) - Retorna a co-tangente do ângulo (em radianos) representado por expressão.

Exp(base, expoente) - Retorna o valor de base elevado a expoente, sendo ambos expressões do tipo real.

GraupRad(expressão) - Retorna o valor em radianos correspondente ao valor em graus representado por expressão.

Int(expressão) - Retorna a parte inteira do valor representado por expressão.

Log(expressão) - Retorna o logaritmo na base 10 do valor representado por expressão.

LogN(expressão) - Retorna o logaritmo neperiano (base e) do valor representado por expressão.

Pi - Retorna o valor 3.141592.

Quad(expressão) - Retorna quadrado do valor representado por expressão.

RadpGrau(expressão) - Retorna o valor em graus correspondente ao valor em radianos representado por expressão.

RaizQ(expressão) - Retorna a raiz quadrada do valor representado por expressão.

Rand - Retorna um número real gerado aleatoriamente, maior ou igual a zero e menor que um.

RandI(limite) - Retorna um número inteiro gerado aleatoriamente, maior ou igual a zero e menor que limite.

Sen(expressão) - Retorna o seno do ângulo (em radianos) representado por expressão.

Tan(expressão) - Retorna a tangente do ângulo (em radianos) representado por expressão.

Os valores que estão entre parênteses, representados pelas palavras como *expressão*, *base* e *expoente*, são os parâmetros, ou como dizem alguns autores, os argumentos que passamos para a função para que realize seus cálculos e retorne um outro, que usaremos no programa. Algumas funções, como Pi e Rand, não precisam de parâmetros, mas a maioria tem um ou mais. O valor dos parâmetros naturalmente altera o valor retornado pela função.

11.6. Funções para manipulação de cadeias de caracteres (Strings)

Asc (s : caracter) : Retorna um inteiro com o código ASCII do primeiro caracter da expressão.

Carac (c : inteiro) : Retorna o caracter cujo código ASCII corresponde à expressão.

Caracpnum (c : caracter) : Retorna o inteiro ou real representado pela expressão. Corresponde a StrToInt() ou StrToFloat() do Delphi, Val() do Basic ou Clipper, etc.

Compr (c : caracter) : Retorna um inteiro contendo o comprimento (quantidade de caracteres) da expressão.

Copia (c : caracter ; p, n : inteiro) : Retorna um valor do tipo caracter contendo uma cópia parcial da expressão, a partir do caracter p, contendo n caracteres. Os caracteres são numerados da esquerda para a direita, começando de 1. Corresponde a Copy() do Delphi, Mid$() do Basic ou Substr() do Clipper.

Maiusc (c : caracter) : Retorna um valor caracter contendo a expressão em maiúsculas.

Minusc (c : caracter) : Retorna um valor caracter contendo a expressão em minúsculas.

Numpcarac (n : inteiro ou real) : Retorna um

valor caracter contendo a representação de n como uma cadeia de caracteres. Corresponde a IntToStr() ou FloatToStr() do Delphi, Str() do Basic ou Clipper.

Pos (subc, c : caracter) : Retorna um inteiro que indica a posição em que a cadeia subc se encontra em c, ou zero se subc não estiver contida em c. Corresponde funcionalmente a Pos() do Delphi, Instr() do Basic ou At() do Clipper, embora a ordem dos parâmetros possa ser diferente em algumas destas linguagens.

11.7. Criando Funções

A criação de uma função deve ser realizada dentro da seção de variáveis, esse tipo de subalgoritmo sempre retorna apenas um valor para o algoritmo que o chamou. As funções possuem um tipo de retorno associado. Uma função pode possui 0, 1 ou mais parâmetros.

Sintaxe:

```
algoritmo <nome do algoritmo>
var
      <declaração das variáveis globais>
      <definição das funções>
inicio
      <lista de comandos>
fimalgoritmo
```

Como exemplo, vamos criar um programa que lê dois números e os somam. Para isso iremos criar uma função chamada **somar**, que receberá dois números inteiros como parâmetros e devolverá também um inteiro contendo o resultado da soma.

```
Algoritmo Somar_com_Funcoes

[Declaração de Variáveis]
   valor1, valor2, resultado : inteiro

Função somar(n1, n2 : inteiro) : inteiro
[Início]

[Declaração de Variáveis Locais]
   resultado : inteiro

[Processamento]
   resultado ← n1 + n2
   retorne(resultado)

[Fim]

[Processamento Principal - Início]
   escreva("Digite o primeiro membro da soma: ")
   leia(valor1)
   escreva("Digite o segundo valor da soma: ")
   leia(valor2)
   resultado ← somar(valor1, valor2)
   escreva("O resultado é ", resultado)
[Fim]
```

O exemplo acima, não está codificada especificamente para o VisuAlg. Você pode estar usando este exemplo posteriormente como exercício para aplicar no programa, o que acha?

Bom, vimos acima um exemplo bastante simples, onde declaramos uma função chamada **somar** que recebe dois valores inteiros (**n1** e **n2**) e retorna um valor inteiro, contendo a soma dos dois valores passados como argumento.

Você pode pensar: “que coisa inútil... é muito mais fácil usar o operador + e pronto”. Calma, esse foi só um exemplo. Obviamente não iremos criar funções que já existem. As funções servem pra que possamos **modularizar** o código, dividindo tarefas e deixando o código mais **legível**.

Por exemplo, podemos criar uma função que

calcule o imposto de renda, que calcule as raízes de uma equação de 2° grau, dentre outros. Com isso, se tivermos outro programa que também necessite de tais cálculos, podemos reutilizar o código!

Podemos perceber em nosso código que declaramos algumas variáveis globais e locais. Nesse caso, declaramos as variáveis **globais**: valor1, valor2 e resultado, enquanto que declaramos as variáveis **locais**: n1, n2 e resultado. Epa! Surgiram duas dúvidas agora! **Primeiro:** n1 e n2 são parâmetros e não variáveis locais, certo? **Errado**.

Os **parâmetros** que declaramos em nossas funções e procedimentos **também são variáveis locais**, apesar de não terem sido declarados na área de variáveis locais. A segunda dúvida é: você declarou **duas variáveis de nome 'resultado'!** Por que o compilador não acusou erro?

Quando temos duas variáveis de mesmo nome, sendo uma global e uma local, o programa **sempre** vai levar em conta a variável local. Ele acusaria erro se declarássemos suas variáveis de mesmo nome dentro de **um mesmo escopo**.

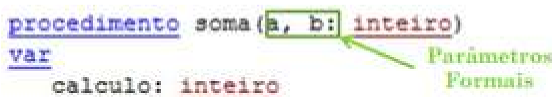
No algoritmo, quando vamos **retornar** o valor que a função calculou, utilizamos simplesmente **retorne()**.

11.8. Passagens de Parâmetros em Procedimentos

Parâmetros são canais por onde os dados são transferidos pelo algoritmo chamador a um subalgoritmo.

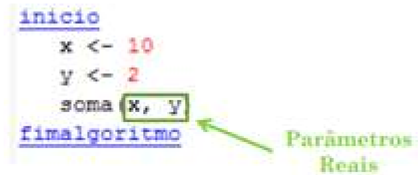
Parâmetros formais

São os nomes simbólicos usados na definição dos parâmetros de um subalgoritmo.



Parâmetros Reais

São aqueles que substituem os parâmetros formais quando da chamada de um subalgoritmo.



A substituição dos parâmetros formais pelos parâmetros reais no ato da invocação de um subalgoritmo é denominada de passagem de parâmetros. Pode se dar dois mecanismos distintos:

- Passagem por valor (ou por cópia)
- Passagem por referência

Exemplo de passagem por Valor:

PASSAGEM POR VALOR

```
algoritmo "PassagemPorValor"
var
   x, y: inteiro

   procedimento soma(a, b: inteiro)
   var
      calculo: inteiro
   inicio
      calculo <- a + b
      escreval("O valor da soma é ", calculo)
      a <- 3
      b <- 4
   fimprocedimento

inicio
   x <- 1
   y <- 2
   soma(x, y)
   escreval("Os valores de x e y são: ", x, y)
fimalgoritmo
```

Exemplo de passagem por Referência:

Passagem por Referência

```
algoritmo "PassagemPorReferencia"
var
   x, y: inteiro

   procedimento soma(var a, b: inteiro)
   var
      calculo: inteiro
   inicio
      calculo <- a + b
      escreval("O valor da soma é ", calculo)
      a <- 3
      b <- 4
   fimprocedimento

inicio
   x <- 1
   y <- 2
   soma(x, y)
   escreval("Os valores de x e y são: ", x, y)
fimalgoritmo
```

Funções e **procedimentos** são utilizados com muita frequência em desenvolvimento de softwares. São vários benefícios como: evita duplicação de código quando precisamos executar a mesma operação várias vezes, deixa o entendimento do algoritmo mais intuitivo, pois tiramos a parte complexa do código do fluxo principal do algoritmo, etc.

11.9. Exercícios Passo a Passo

1. No VisuAlg, insira as variáveis de valor inteiro.

2. Insira uma nova área de variável e determine a variável total de tipo de dados inteiro.

3. Insira uma nova área de início e então aplique os comandos para realizar o cálculo do dobro.

4. Finalize o comando procedimento.

5. Na área início, insira o comando que determine ao usuário digitar um número.

6. Insira o comando que determine ao VisuAlg que leia o número digitado e execute o procedimento.

7. Teste o algoritmo criado.

11.10. Exercícios de Fixação

1. Desenvolva um algoritmo que, dado o nome de uma pessoa (Nome + Sobrenome), retorna apenas o sobrenome. Ex: “Pedro Ferreira”, retorna somente “Ferreira”.

2. Crie uma função que leia um número não determinado de valores positivos e calcule a soma desses números. A função deve parar de ler números quando um número negativo for digitado e retornar a soma dos números lidos.

3. Crie um procedimento que receba um número como parâmetro e escreva a tabuada desse número.

4. Crie um procedimento que, dado um número N por parâmetro, desenhe o seguinte padrão na tela:

anotações

12. Modularização

Os algoritmos que temos construído até então são muito simples, pois resolvem problemas simples e apresentam apenas os componentes mais elementares dos algoritmos: constantes, variáveis, expressões condicionais e estruturas de controle. Entretanto, a maioria dos algoritmos resolve problemas complicados, cuja solução pode ser vista como formada de várias subtarefas ou módulos, cada qual resolvendo uma parte específica do problema.

Nesta aula, veremos como escrever um algoritmo constituído de vários módulos e como estes módulos trabalham em conjunto para resolver um determinado problema algorítmico.

12.1. Modularização ou Sub-rotinas

É o processo de dividir o algoritmo em partes ou módulos que executam tarefas específicas.

A divisão dum sistema em módulos tem várias vantagens. Para o fabricante, por um lado, a modularização tem a vantagem de reduzir a complexidade do problema, dividindo-o em sub-problemas mais simples, que podem inclusivamente ser resolvidos por equipas independentes.

Até sob o ponto de vista do fabrico é mais simples alterar a composição de um módulo, por exemplo porque se desenvolveram melhores circuitos para o amplificador, do que alterar a composição de um sistema integrado. Por outro lado, é mais fácil detectar problemas e resolvê-los, pois os módulos são, em princípio, razoavelmente independentes.

Claro que os módulos muitas vezes não são totalmente independentes. Por exemplo, o sistema de controlo à distância duma aparelhagem implica interação com todos os módulos simultaneamente. A arte da modularização está em identificar claramente que módulos devem existir no sistema, de modo a garantir que as ligações entre os módulos são minimizadas e que a sua coesão interna é máxima. Isto significa que, no caso de um bom sistema de alta fidelidade, os cabos entre os módulos são simplificados ao máximo e que os módulos contêm apenas os circuitos que garantem que o módulo faz a sua função.

A coesão tem portanto a ver com as ligações internas a um módulo, que idealmente devem ser maximizadas. Normalmente, um módulo é coeso se tiver uma única função, bem definida.

Para um utilizador, por outro lado, a modularização tem como vantagem principal permitir a alteração de um único módulo sem ter de comprar um sistema novo. Claro que para isso acontecer o novo módulo deve ter a mesma função do módulo substituído e possuir uma interface idêntica (os mesmo tipo de cabos com o mesmo tipo de sinal eléctrico). Isto é, os módulos, do ponto de vista do utilizador, funcionam como "caixas pretas" com uma função bem definida e com interfaces bem conhecidas. Para o utilizador o interior de um módulo é irrelevante. Mas a modularização tem outras vantagens: o amplificador pode, no futuro, ser reutilizado num sistema de vídeo, por exemplo, evitando a duplicação de circuitos com a mesma função (se nunca pensou nisso, lembre-se que a televisão tem o seu próprio amplificador, normalmente de fraca qualidade, servindo apenas para a encarecer). Aliás, o amplificador era já utilizado para amplificar o sinal de vários outros módulos (e.g., o leitor de CD ou o sintonizador).

As vantagens da modularização são muitas, como se viu. A modularização é um dos métodos usados em engenharia da programação para desenvolvimento de programas de grande escala. Mas a modularização é útil mesmo para

pequenos programas, quanto mais não seja pelo treino que proporciona.

As vantagens da modularização para a programação são pelo menos as seguintes:

1. Facilita a detecção de erros, pois é em princípio simples verificar qual é o módulo responsável pelo erro.
2. É mais fácil testar os módulos individualmente do que o programa completo.
3. É mais fácil fazer a manutenção (correcção de erros, melhoramentos, etc.) módulo por módulo do que no programa total. Além disso, a modularização aumenta a probabilidade dessa manutenção não ter consequências nefastas nos outros módulos do programa.
4. Permite o desenvolvimento independente dos módulos. Isto simplifica o trabalho em equipa, pois cada elemento, ou cada sub-equipa, tem a seu cargo apenas alguns módulos do programa.
5. Porventura a mais evidente vantagem da modularização em programas de pequena escala, mas também nos de grande escala, é a possibilidade de reutilização do código* desenvolvido.

Um programador assume, ao longo do desenvolvimento de um programa, os dois papeis descritos acima, por um lado o fabricante, pois é sua responsabilidade desenvolver módulos; por outro é utilizador, pois fará com certeza uso de outros módulos, desenvolvidos por outrem ou por ele próprio no passado. Esta é uma noção muito importante. É de toda a conveniência que um programador possa ser um mero utilizador dos módulos já desenvolvidos, sem se preocupar com o seu funcionamento interno: ele sabe qual a interface do módulo e qual a sua função, e usa-o. Isto permite reduzir substancialmente a complexidade da informação que o programador tem de ter presente na sua memória, conduzindo por isso a substanciais ganhos de produtividade e a uma menor taxa de erros.

* **Dá-se o nome de código a qualquer pedaço de programa numa dada linguagem de programação.**

12.2. Escopo de Dados e Códigos

O escopo de um módulo (ou variável) de um algoritmo é a parte ou partes do algoritmo em que o módulo (ou variável) pode ser referenciado. Quando iniciamos o estudo de modularização é natural nos perguntarmos qual é o escopo de um dado módulo e das constantes ou variáveis nele presentes. Em particular, o escopo de um módulo determina quais são os demais módulos do algoritmo que podem chamar-lhe e quais ele pode chamar.

Os módulos de um algoritmo são organizados por níveis. No primeiro nível, temos apenas o algoritmo principal. Aqueles módulos que devem ser acessados pelo algoritmo principal devem ser escritos dentro dele e, nesta condição, são ditos pertencerem ao segundo nível. Os módulos escritos dentro de módulos de segundo nível são ditos módulos de terceiro nível. E assim sucessivamente.

A regra para determinar o escopo de um módulo é bastante simples: um módulo X escrito dentro de um módulo A qualquer pode ser acessado apenas pelo módulo A ou por qualquer módulo escrito dentro de A ou descendente (direto ou não) de algum módulo dentro de A.

Como exemplo, considere um algoritmo no qual o módulo principal contém outros quatro módulos, S1, S2, F1, F2. Por sua vez, S1 contém mais dois módulos, S3 e F3; e F2 contém os módulos S4 e F4. De acordo com as regras de escopo descritas anteriormente, o módulo F3 só pode ser chamado por S1 e S3, enquanto o módulo F1 pode ser acessado por todos os módulos.

12.3. Alguns comandos no VisuAlg

Nas próximas duas aulas, o foco será no desenvolvimento de dois projetos específicos e práticos, logo o conteúdo conceitual do curso, se completa nesta aula.

Para referendar esta aula, e lhe auxiliar com os projetos a serem desenvolvidos nas duas aulas seguintes, vamos dar exemplos de alguns

comandos específicos no VisuAlg, para que você possa incrementar seus algoritmos.

12.4. Parâmetros Interrompa

Causa uma saída imediata do laço. Embora esta técnica esteja de certa forma em desacordo com os princípios da programação estruturada, o comando interrompa foi incluído no VisuAlg por ser encontrado na literatura de introdução à programação e mesmo em linguagens como o Object Pascal (Delphi/Kylix), Clipper, VB, etc. Seu uso é exemplificado a seguir:

```
algoritmo "Números de 1 a 10 (com interrompa)"
var x: inteiro
inicio
x <- 0
repita
   x <- x + 1
   escreva (x:3)
   se x = 10 entao
      interrompa
   fimse
ate falso
fimalgoritmo
```

12.5. Comando Aleatório

Muitas vezes a digitação de dados para o teste de um programa torna-se uma tarefa entediante. Com o uso do comando aleatório do VisuAlg, sempre que um comando leia for encontrado, a digitação de valores numéricos e/ou caracteres é substituída por uma geração aleatória. Este comando não afeta a leitura de variáveis lógicas: com certeza, uma coisa pouco usual em programação.

Este comando tem as seguintes sintaxes:

aleatorio [on]

Ativa a geração de valores aleatórios que substituem a digitação de dados. A palavra-chave on é opcional. A faixa padrão de valores gerados é de 0 a 100 inclusive. Para a geração de dados do tipo caractere, não há uma faixa préestabelecida: os dados gerados serão sempre strings de 5 letras maiúsculas

aleatorio [,]

Ativa a geração de dados numéricos aleatórios estabelecendo uma faixa de valores mínimos e máximos. Se apenas < valor1> for fornecido, a faixa será de 0 a inclusive; caso contrário, a faixa será de a inclusive. Se for menor que, o VisuAlg os trocará para que a faixa fique correta. Importante: e devem ser constantes numéricas, e não expressões.

aleatorio off

Desativa a geração de valores aleatórios. A palavra-chave off é obrigatória.

12.6. Comando Arquivo

Muitas vezes é necessário repetir os testes de um programa com uma série igual de dados. Para casos como este, o VisuAlg permite o armazenamento de dados em um arquivo-texto, obtendo deles os dados ao executar os comandos leia.

Esta característica funciona da seguinte maneira:

- Se não existir o arquivo com nome especificado, o VisuAlg fará uma leitura de dados através da digitação, armazenando os dados lidos neste arquivo, na ordem em que forem fornecidos.

- Se o arquivo existir, o VisuAlg obterá os dados deste arquivo até chegar ao seu fim. Daí em diante, fará as leituras de dados através da digitação.

- Somente um comando arquivo pode ser empregado em cada pseudocódigo, e ele deverá estar na seção de declarações (dependendo do "sucesso" desta característica, em futuras versões ela poderá ser melhorada...).

- Caso não seja fornecido um caminho, o VisuAlg irá procurar este arquivo na pasta de trabalho corrente (geralmente, é a pasta onde o programa VISUALG.EXE está). Este comando não prevê uma extensão padrão; portanto, a especificação do nome do arquivo deve

ser completa, inclusive com sua extensão (por exemplo, .txt, .dat, etc.).

Exemplo:

```
algoritmo "lendo do arquivo"
arquivo "teste.txt"
var x,y: inteiro
inicio
para x de 1 ate 5 faca
   leia (y)
fimpara
fimalgoritmo
```

12.7. Comando Timer

Embora o VisuAlg seja um interpretador de pseudocódigo, seu desempenho é muito bom: o tempo gasto para interpretar cada linha digitada é apenas uma fração de segundo. Entretanto, por motivos educacionais, pode ser conveniente exibir o fluxo de execução do pseudocódigo comando por comando, em "câmera lenta". O comando timer serve para este propósito: insere um atraso (que pode ser especificado) antes da execução de cada linha. Além disso, realça em fundo azul o comando que está sendo executado, da mesma forma que na execução passo a passo.

Sua sintaxe é a seguinte:

timer on

Ativa o timer.

timer

Ativa o timer estabelecendo seu tempo de atraso em milissegundos. O valor padrão é 500, que equivale a meio segundo. O argumento deve ser uma constante inteira com valor entre 0 e 10000. Valores menores que

0 são corrigidos para 0, e maiores que 10000 para 10000.

timer off

Desativa o timer.

Ao longo do pseudocódigo, pode haver vários comandos timer. Todos eles devem estar na seção de comandos. Uma vez ativado, o atraso na execução dos comandos será mantido até se chegar ao final do pseudocódigo ou até ser encontrado um comando timer off.

12.8. Comandos de Depuração

Nenhum ambiente de desenvolvimento está completo se não houver a possibilidade de se inserir pontos de interrupção (breakpoints) no pseudocódigo para fins de depuração. VisuAlg implementa dois comandos que auxiliam a depuração ou análise de um pseudocódigo: o comando **pausa** e o comando **debug**.

12.9. Comando Pausa

Sua sintaxe é simplesmente:

Pausa

Este comando insere uma interrupção incondicional no pseudocódigo. Quando ele é encontrado, o VisuAlg pára a execução do pseudocódigo e espera alguma ação do programador. Neste momento, é possível: analisar os valores das variáveis ou das saídas produzidas até o momento; executar o pseudocódigo passo a passo (com F8); prosseguir sua execução normalmente (com F9); ou simplesmente terminá-lo (com Ctrl-F2). Com exceção da alteração do texto do pseudocódigo, todas as funções do VisuAlg estão disponíveis.

12.10. Comando Debug

Sua sintaxe é:

debug

Se a avaliação de resultar em valor VERDADEIRO, a execução do pseudocódigo será interrompida como no comando pausa. Dessa forma, é possível a inserção de um breakpoint condicional no pseudocódigo.

12.11. Comando Eco

Sua sintaxe é:

eco on | off

Este comando ativa (eco on) ou desativa (eco off) a impressão dos dados de entrada na saída-padrão do VisuAlg, ou seja, na área à direita da parte inferior da tela. Esta característica pode ser útil quando houver uma grande quantidade de dados de entrada, e se deseja apenas analisar a saída produzida. Convém utilizá-la também quando os dados de entrada provêm de um arquivo já conhecido.

12.12. Comando Cronômetro

Sua sintaxe é:

cronometro on | off

Este comando ativa (cronometro on) ou desativa (cronometro off) o cronômetro interno do VisuAlg. Quando o comando cronometro on é encontrado, o VisuAlg imprime na saída-padrão a informação "Cronômetro iniciado.", e começa a contar o tempo em milissegundos. Quando o comando cronometro off é encontrado, o VisuAlg imprime na saída-padrão a informação "Cronômetro terminado. Tempo decorrido: xx segundo(s) e xx ms". Este comando é útil na análise de desempenho de algoritmos (ordenação, busca, etc.).

12.13. Comando LimpaTela

Sua sintaxe é:

limpatela

Este comando simplesmente limpa a tela DOS do Visualg (a simulação da tela do computador). Ele não afeta a "tela" que existe na parte inferior direita da janela principal do Visualg.

12.14. Exercícios Passo a Passo

1. Vamos desenvolver na prática um programa simples, que tem o objetivo de informar se o número digitado é par ou ímpar. No VisuAlg, insira a variável, assim como determine que o usuário insira um número inteiro.

2. Faça com que o programa armazene o número digitado e insira o comando responsável por realizar o teste lógico.

3. Insira o comando responsável por retornar na tela o número caso ele seja par.

4. Insira o comando responsável por retornar na tela o número caso ele seja ímpar.

5. Teste o algoritmo desenvolvido.

12.15. Exercícios de Fixação

1. Desenvolva um algoritmo que leia um número e retorne o seu quadrado e o seu cubo.

2. Desenvolva um algoritmo em para ler o ano de nascimento de uma pessoa, calcular e mostrar sua idade e, também, verificar e mostrar se ela já tem idade para votar (16 anos ou mais) e para conseguir a Carteira de Habilitação (18 anos ou mais).

3. Desenvolva um algoritmo para criar um vetor real de 20 posições: as 10 primeiras são informados pelo usuário, e as 10 seguintes são os mesmos números em ordem inversa.

4. Desenvolva um algoritmo que leia 5 valores reais e armazenar em um vetor. Modifique o vetor de modo que os valores das posições impares sejam aumentados em 5%, e os das posições pares sejam aumentados em 2%. Imprima depois o vetor resultante.

13. Prática 1

Nas nossas duas últimas apostilas de Lógica de programação, nosso objetivo será praticar todo o conteúdo visto no curso.

Separamos três exercícios práticos para que você possa realizar. Um deles será acompanhado com o aluno, passo a passo, conforme ocorreu em todas as aulas anteriores.

Os outros dois exercícios, vão se localizar na área de fixação, eles terão apenas o enunciado e o aluno deve desenvolver todo o projeto usando o conhecimento adquirido nas aulas interativas e utilizando da sua apostila.

Nas aulas interativas, demos algumas dicas sobre alguns dos códigos que utilizamos nos exercícios, com o objetivo de dar um norte ao aluno.

Após a conclusão dos projetos o aluno deve realizar o teste da aula normalmente como foi realizado também em todas as demais aulas anteriores.

Para lhe auxiliarmos, resolvemos apresentar aqui na apostila alguns exemplos de algoritmos resolvidos.

Desta forma você pode se basear na linha de códigos usadas, para então realizar os exercícios práticos propostos.

Exemplo 1:

Uma Empresa decidiu fazer um levantamento dos candidatos que se inscreveram para preenchimento de vaga no seu quadro de funcionários, utilizando processamento eletrônico e você foi contratado. Escreva um algoritmo que leia, via teclado, um conjunto de informações para cada candidato, contendo:

- Número de inscrição do candidato
- Idade
- Sexo
- Experiência anterior(S-sim/N-nao)

Calcule:

- Quantidade de candidatos?
- Quantidade de candidatas?
- Média de idade dos homens com experiência?
- Percentagem dos homens com mais de 45 anos?
- Quantidade de mulheres com idade inferior a 35 anos e com experiência?
- Menor idade entre as mulheres que já tem experiência no serviço?

A resolução deste algoritmo, seria a seguinte:

```
algoritmo "Exemplo"

var
Numeroidentificacao,I,n,Homens,Mu_menor35,nh45,he,Si:inteiro
Sx,p,o:caractere
Porc_Hmaior45,media,menor:real
inicio
// Seção de Comandos
O<-"s"
menor<-9999999999
enquanto O="s" faca
escreval("************************************************************")
escreval("Informações sobre o candidato.")
escreval("************************************************************")
escreval()
escreval()
n<-n+1
escreval("Informe o número da matricula do candidato")
leia (Numeroidentificacao)
escreval("Informe o sexo do candidato (m-masculino; f-feminino)")
leia (sx)
escreval("Informe a idade do candidato")
Leia (I)
escreval("Primeiro emprego? (s-sim; n-não)")
Leia (p)
escreval("Inscrever outro candidato. (s-sim; n-não)")
leia (o)
se sx="m" entao
Homens<-Homens+1
se (p="n") entao
He<-He+1
Si<-Si+I
```

Segue abaixo a continuação do código:

```
se sx="m" entao
Homens<-Homens+1
se (p="n") entao
He<-He+1
Si<-Si+I
media<-Si/He
fimse
se I>45 entao
nh45<-nh45+1
Porc_Hmaior45<-(nh45/Homens)*100
fimse
fimse
se (Sx="f") e (p="n") entao
se I menor<-I
fimse
se I<35 entao
Mu_menor35<-Mu_menor35+1
fimse
fimse
limpatela
fimenquanto
escreval("Resultados")
Escreval ("número de candidatos",n)
escreval("O número de candidatos é ",Homens)
escreval("O número de candidatas é ",N-Homens)
escreval("Média de idade dos homens com experiência ",media)
escreval("Porcentagem dos homens maiores que 45 anos ",Porc_Hmaior45)
escreval("mulheres com experiência com idade inferior a 35 anos ",Mu_menor35)
escreval("Mulher mais nova com experiência",menor)
fimalgoritmo
```

Exemplo 2:

Ler uma matriz de 3x3 e dizer a quantidade de valores lidos, a quantidade números pares lidos e a quantidade de números ímpares lidos:

Resolução do algoritmo:

```
algoritmo "semnome"
var
     matriz:vetor[0..2,0..2] de inteiro
     somaPar,somaImpar,somaTodos, linha, coluna:inteiro

inicio

somaPar <-0
somaImpar <-0
somaTodos  <-0

para linha de 0 ate 2 faca
  para coluna de 0 ate 2 faca
       leia(matriz[linha,coluna])
       somaTodos <- somaTodos + 1
          se (matriz[linha,coluna] % 2) = 0 entao
             somaPar <- somaPar + 1
          senao
             somaImpar <- somaImpar +1
          fimse
  fimpara
fimpara

escreval("escrevendo a matriz")
para linha de 0 ate 2 faca
  para coluna de 0 ate 2 faca
     escreval(matriz[linha,coluna])
  fimpara
fimpara

escreval("quantidade de numeros lidos ", somaTodos)
escreval("quantidade de numeros impares lidos ", somaImpar)
escreval("quantidade de numeros pares lidos ", somaPar)

fimalgoritmo
```

13.1. Exercícios Passo a Passo

1. Nosso objetivo é criar um algoritmo que acerte o número que o usuário está pensando. No VisuAlg Insira as variáveis de tipo inteiro.

2. Insira a variável de tipo caractere.

3. Determine o primeiro bloco de comandos do algoritmo, dando valores específicos às variáveis.

4. Determine o valor que a variável palpite deve receber neste momento do algoritmo.

5. Insira o comando responsável por questionar o usuário referente ao seu número.

6. Faça com que o VisuAlg armazene o valor digitado, e então aplique um comando SE, no caso de o programa ter acertado o número.

7. Insira um comando senão, para o caso de o usuário ter digitado “maior”.

8. Insira um comando senão, para o caso de o usuário ter digitado “menor”.

9. Insira um comando senão, para o caso de o usuário ter digitado outra palavra qualquer.

10. Determine uma mensagem que solicite ao usuário que insira uma das palavras pré-determinadas.

11. Complete os comandos SEs usados anteriormente além de aplicar o comando para incrementar a variável tentativas.

12. Conclua mais um laço SE, e então insira o comando necessário para encerrar o programa no caso de o programa acertar o número, ou ultrapassar o número de 10 tentativas.

13. Aplique mais um comando SE, no caso de o programa ultrapassar 10 tentativas.

14. Determine a mensagem que o algoritmo deve retornar, ao ultrapassar mais de 10 tentativas.

15. Conclua mostrando na tela, o número de tentativas.

16. Teste o algoritmo desenvolvido.

13.2. Exercícios de Fixação

1. Desenvolva um algoritmo que simule o jogo da batalha naval.

2. Desenvolva um algoritmo que receba números impares e então retorne um desenho na tela com os respectivos caracteres: +, - e |. Ex: 5,3:

anotações

14. Prática 2

Nesta Apostila, damos continuidade ao desenvolvimento de alguns algoritmos propostos, com o objetivo de colocarmos em prática todo o conteúdo visto durante o curso.

Como ocorreu na aula anterior, são três projetos a serem desenvolvidos, um deles vamos estar desenvolvendo junto com você passo a passo, e os outros três ficaram a disposição dentro do menu fixação, apenas com o enunciado de cada uma das questões.

O Aluno deve utilizar dos seus conhecimentos adquiridos no curso, pra desenvolver os algoritmos, você terá total liberdade para modificar o programa conforme desejar.

Use a sua apostila para tirar duvidas sobre a utilização de códigos e comandos que podem lhe auxiliar no visuAlg.

Assim como na aula anterior, vamos demonstrar dois exemplos de programas prontos aqui, para que possas praticar um pouco mais, e lhe auxiliar com o desenvolvimento dos desafios.

Exemplo 1:

Ao completar o tanque de combustível de automóvel, faça um algoritmo que calcule o consumo efetuado, assim como a autonomia que o carro ainda teria antes do abastecimento. Considere que o veículo sempre seja abastecido até encher o tanque e que são fornecidas apenas a capacidade do tanque, quantidade de litros abastecidos e a quilometragem percorrida desde o último abastecimento.

Segue a resolução do algoritmo:

```
algoritmo "CALCULA QUILOMETRAGEM POR LITRO"

var
capacTanq: inteiro //quantos litros cabem no tanque.
litroAbast: inteiro //quant. de litros abastecidos.
kmpUltAbast:inteiro //km pecorrido desde o último abastecimento.
respCons,sobraTanq,autonFutura:real
inicio
//qual consumo efetuado?
//qual autonomia que ainda teria antes do abastecimento?
//Seção de Comandos
escreva ("Digite a capacidade do tanque em litros: ")
leia(capacTanq)
escreva ("Quantidades de litros abastecidos: ")
leia (litroAbast)
escreva ("Qual quilometragem desde o último abastecimento? ")
leia(kmpUltAbast)
respCons <- litroAbast/kmpUltAbast
//consumo desde último abastecimento Lts
sobraTanq <- capacTanq - litroAbast
//calcula o que sobrou no tanque.
autonFutura <- sobraTanq/respCons
//calcula os kms que percorreria ainda com o restou do tanque.
escreval("Consumo efetuado é de: ",respCons," Lts/km")
escreval("Autonomia que ainda teria é de:",autonFutura, " km")
escreval (" ")
Escreval ("---------------Outras curiosidades--------------------")
//Somente para título
escreval("Quantos litros sobraram no tanque?",sobraTanq," Litros")
escreval("Kilometragem por litro? ",kmpUltAbast/litroAbast:2:1," K/Lts")
fimalgoritmo
```

Exemplo 2:

Desenvolva um algoritmo que mostre ao usuário um menu de votação, neste menu deve o conter o nome das seguintes redes de televisão aberta:

Record, SBT, Band, Globo, RedeTV, Gazeta.

O objetivo do programa é que o usuário possa votar na opção que mais assiste. O usuário pode votar quantas vezes desejar, incluindo votar em branco, ao final quando ele digitar o número zero, o programa deve encerrar e então mostrar na tela o resultado de audiência de cada uma das redes de televisão assim como os votos em branco.

Resolução do algoritmo:

```
algoritmo "VERIFICAR AUDIÊNCIA DE CANAL DE TV"
var
Record,SBT,Band,Globo,redeTv,Gazeta: real
branco, total:real
canal: caractere
inicio
//O laço dá looping até o usuário digitar Zero
enquanto canal <> "0" faca
escreval("Digite uma das opções ou 0 para sair:")
escreval("1 - Record")
escreval("2 - SBT")
escreval("3 - Band")
escreval("4 - Globo")
escreval("5 - rede Tv")
escreval("6 - Gazeta")
escreval("7 - Voto em Branco")
//entrada da escolha do usuário
leia(canal)
//limpa a tela
limpatela
escolha canal
caso "1"
Record <- Record + 1
caso "2"
SBT <- SBT + 1
caso "3"
Band <- Band + 1
caso "4"
Globo <- Globo + 1
caso "5"
redeTv <- redeTv + 1
caso "6"
```


```
redeTv <- redeTv + 1
caso "6"
```


```
Gazeta <- Gazeta + 1
caso "7"
branco <- branco + 1
//zero para sair do laço
caso "0"
//não escreve nada
escreva("")
outrocaso
escreval ("Digite um canal da lista!")
fimescolha
fimenquanto //fim do laço
//soma os valores e aplica em variável total
total<-Record + SBT + Band + Band + redeTv + Gazeta + branco
//Divide variável por total e depois multiplica
//por 100 para obter a porcentagem
Record<-(Record / total)*100
SBT<-(SBT / total)*100
Band<-(Band / total)*100
Globo<-(Globo / total)*100
redeTv<-(redeTv / total)*100
Gazeta<-(Gazeta / total)*100
branco<-(branco / total)*100
(não escreve nada já que não há nada digitado entre parênteses;)
(entretando pula linha já que o comando é escreval e não escreva)
escreval("")
```


```
fimenquanto //fim do laço
//soma os valores e aplica em variável total
total<-Record + SBT + Band + Band + redeTv + Gazeta + branco
//Divide variável por total e depois multiplica
//por 100 para obter a porcentagem
Record<-(Record / total)*100
SBT<-(SBT / total)*100
Band<-(Band / total)*100
Globo<-(Globo / total)*100
redeTv<-(redeTv / total)*100
Gazeta<-(Gazeta / total)*100
branco<-(branco / total)*100
(não escreve nada já que não há nada digitado entre parênteses;)
(entretando pula linha já que o comando é escreval e não escreva)
escreval("")
```


```
(Exibe o resultado)
escreval("A totalização da audiência em % :")
escreval("Record teve ",Record:2:2,"% de audiência.")
escreval("SBT teve ",SBT:2:2,"% de audiência.")
escreval("Band teve ",Band:2:2,"% de audiência.")
escreval("Globo teve ",Globo:2:2,"% de audiência.")
escreval("redeTv teve ",redeTv:2:2,"% de audiência.")
escreval("Gazeta teve ",Gazeta:2:2,"% de audiência.")
escreval("Não assistiram: ",branco:2:2,"%.")
fimalgoritmo
```

14.1. Exercícios Passo a Passo

1. Nosso objetivo é desenvolver um algoritmo que calcule a hora final que o usuário levou para finalizar um jogo de vídeo-game qualquer. No VisuAlg insira as variáveis necessárias.

2. Insira a primeira parte dos comandos, solicitando ao usuário que determine a hora e o minuto do início do jogo.

3. Insira o comando para o sistema determinar que o usuário não possa inserir um número superior a 23 para a hora inicial.

4. Insira o início do próximo comando.

5. Conclua o comando que determina que o usuário não possa inserir um número superior a 59 para os minutos finais.

6. Insira o comando para o sistema determinar que o usuário não possa inserir um número superior a 23 para a hora final.

7. Insira o comando para o sistema determinar que o usuário não possa inserir um número superior a 59 para os minutos finais.

8. Insira o comando que faça o primeiro teste lógico.

9. Insira o comando responsável receber os dados.

10. Insira o segundo teste lógico do

algoritmo assim como o cálculo necessário.

11. Insira o terceiro teste lógico do algoritmo assim como o cálculo necessário.

12. Insira o quarto teste lógico do algoritmo assim como o cálculo necessário.

13. Insira o quinto teste lógico do algoritmo assim como o cálculo necessário.

14. Insira o comando SE que determina os resultados que podem ser apresentados no programa.

15. Determine a mensagem que o sistema deve retornar na tela, com o cálculo realizado.

16. Finalize os laços.

17. Teste o algoritmo.

14.2. Exercícios de Fixação

1. Desenvolva um algoritmo que simule o jogo da forca no VisuAlg.

2. Desenvolva um algoritmo que permita realizar cadastros, sair do sistema, excluir cadastros, consultar cadastros pelo nome, e recarregar o programa.

anotações