Ministério da Educação
Escola Técnica Aberta do Brasil
Universidade Tecnológica Federal do Paraná

Lógica de Programação

Everton Coimbra de Araújo

EdUFMT
Cuiabá, 2009

Ministério da Educação

Comissão Editorial Profª Drª Maria Lucia Cavalli Neder - UFMT
Profª Drª Ana Arlinda de Oliveira - UFMT
Profª Drª Lucia Helena Vendrusculo Possari - UFMT
Profª Drª Gleyva Maria Simões de Oliveira - UFMT
Prof. Dr. Henrique Oliveira da Silva - UTFPR
Prof. M. Sc. Oreste Preti - UAB/UFMT

Designer Educacional Oreste Preti e Gleyva Maria S. de Oliveira

Ficha Catalográfica

A663l Araújo, Everton Coimbra de
Lógica de programação / Everton Coimbra de Araújo.-
Cuiabá : EdUFMT, 2009.
132 p. : il. ; color.

Bibliografia: p. 132.
ISBN 978-85-61819-62-0

1.Informática. 2. Lógica de programação. I. Título

CDU - 004.422.612

Revisão Germano Aleixo Filho
Capa (lay out) Marcelo Velasco
Ilustração Marcelo Velasco
Diagramação Terencio Francisco de Oliveira

EdUFMT
Cuiabá, 2009

PROGRAMA e-TEC BRASIL

Amigo(a) estudante:

O Ministério da Educação vem desenvolvendo Políticas e Programas para expansão da Educação Básica e do Ensino Superior no País. Um dos caminhos encontrados para que essa expansão se efetive com maior rapidez e eficiência é a modalidade a distância. No mundo inteiro são milhões os estudantes que freqüentam cursos a distância. Aqui no Brasil, são mais de 300 mil os matriculados em cursos regulares de Ensino Médio e Superior a distância, oferecidos por instituições públicas e privadas de ensino.

Em 2005, o MEC implantou o ***Sistema Universidade Aberta do Brasil*** (UAB), hoje, consolidado como o maior programa nacional de formação de professores, em nível superior.

Para expansão e melhoria da educação profissional e fortalecimento do Ensino Médio, o MEC está implementando o ***Programa Escola Técnica Aberta do Brasil*** (e-Tec Brasil). Espera, assim, oferecer aos jovens das periferias dos grandes centros urbanos e dos municípios do interior do País oportunidades para maior escolaridade, melhores condições de inserção no mundo do trabalho e, dessa forma, com elevado potencial para o desenvolvimento produtivo regional.

O e-Tec é resultado de uma parceria entre a Secretaria de Educação Profissional e Tecnológica (SETEC), a Secretaria de Educação a Distância (SEED) do Ministério da Educação, as universidades e escolas técnicas estaduais e federais.

O Programa apoia a oferta de cursos técnicos de nível médio por parte das escolas públicas de educação profissional federais, estaduais, municipais e, por outro lado, a adequação da infra-estrutura de escolas públicas estaduais e municipais.

Do primeiro Edital do e-Tec Brasil participaram 430 proponentes de adequação de escolas e 74 instituições de ensino técnico, as quais propuseram 147 cursos técnicos de nível médio, abrangendo 14 áreas profissionais. O resultado deste Edital contemplou 193 escolas em 20 unidades federativas. A perspectiva do Programa é que sejam ofertadas 10.000 vagas, em 250 pólos, até 2010.

Assim, a modalidade de Educação a Distância oferece nova interface para a mais expressiva expansão da rede federal de educação tecnológica dos últimos anos: a construção dos novos centros federais (CEFETs), a organização dos Institutos Federais de Educação Tecnológica (IFETs) e de seus *câmpus*.

O Programa e-Tec Brasil vai sendo desenhado na construção coletiva e participação ativa nas ações de democratização e expansão da educação profissional no País, valendo-se dos pilares da educação a distância, sustentados pela formação continuada de professores e pela utilização dos recursos tecnológicos disponíveis.

A equipe que coordena o Programa e-Tec Brasil lhe deseja sucesso na sua formação profissional e na sua caminhada no curso a distância em que está matriculado(a).

Brasília, Ministério da Educação – setembro de 2009.

CURSO TÉCNICO EM INFORMÁTICA

Caro estudante:

Os avanços tecnológicos na área de informática e comunicação, associados a modelos pedagógicos que são apoiados pelo uso de tecnologia, deram origem à modalidade de ensino chamada Ensino a Distância ou, simplesmente, EaD.

A característica desse modelo é a separação física entre aluno e professor. Para suprir a distância, a interação, entre o aluno e o professor, é mediada tanto por recursos tecnológicos quanto pelo material impresso. Nessa modalidade de ensino, o material impresso, juntamente com recursos de vídeo, videoconferência e um Ambiente Virtual de Aprendizagem, são as bases tecnológicas, às quais você terá acesso durante sua formação.

Todos esses recursos são meios de comunicação entre professor e aluno. Cada recurso possui uma característica própria e necessita de um canal específico de comunicação. Para assistir aos vídeos, participar de videoconferência ou realizar as atividades do Ambiente Virtual de Aprendizagem, você precisará ter acesso a computadores e a internet. Porém, tais recursos tecnológicos nem sempre estão disponíveis em tempo integral, por isso a importância do material impresso, que permitirá a você ter acesso ao conhecimento independentemente de possuir a sua disposição as tecnologias de informática e comunicação.

Aliado às atividades presenciais e às atividades a distância, o material impresso irá, também, apoiá-lo na realização das atividades de estudos, estimulando-o a participar de forma mais ativa em seu processo de ensino-aprendizagem, construindo progressivamente o conhecimento de maneira interativa. Assim, o professor deixa de ser a única fonte de informação. O distanciamento físico não será impedimento para o processo de cooperação e interação entre você e o professor. O educador criará oportunidades para que você participe de forma ativa durante seu processo de aprendizagem. Dessa maneira, você será parte essencial na construção de seu próprio conhecimento.

O material foi elaborado visando à formação de Técnicos em Informática, segundo os parâmetros do Catálogo Nacional de Cursos Técnicos. O profissional formado deverá ter qualificação para atender à demanda regional em consonância com as tendências tecnológicas. Além disso, deve estar ancorado em um conhecimento científico-tecnológico, de relacionamento interpessoal e comunicação oral. Deve também ter pensamento crítico e racional, capacidade para resolver problemas de ordem técnica, capacidade

criativa e inovadora, capacidade de gestão e visão estratégica. Essa base lhe tornará competitivo no mercado de trabalho.

Mas isso somente não é suficiente. Você deve demonstrar: honestidade, responsabilidade, adaptabilidade, capacidade de planejamento, ser ágil e ter capacidade de decisão. Além de ser possuidor de um espírito crítico, uma formação tecnológica generalista e uma cultura geral sólida e consistente.

Foi pensando nessa formação que equipes de professores da rede pública federal de educação elaboraram seu material. Professores que atuam tanto no ensino médio quanto no ensino superior. Todos profissionais conceituados em suas respectivas áreas de atuação. O objetivo desses profissionais é auxiliar você em sua formação profissional.

Tanto os recursos didáticos pedagógicos, quanto os profissionais envolvidos fazem parte do projeto Escola Técnica Aberta do Brasil, e-Tec Brasil. Um projeto que estabelece parceria entre Instituições de Ensino Público Federal, no papel de formadores, e município, ou Estado, que disponibilizam os pólos que receberão os cursos oferecidos na modalidade de EaD.

Mas lembre-se: simplesmente ter acesso aos recursos didáticos e tecnológicos, além de ter a disposição uma equipe especializada de profissionais, não é suficiente. É necessário que esse material seja utilizado intensamente, de forma a tornar-se fonte de conhecimento que lhe auxiliará em todos os momentos de sua formação.

Cientes de que esse também é o seu desejo, a equipe do e-Tec Brasil deseja a todos ótimo processo de aprendizagem.

Atenciosamente,

Equipe de formadores do curso de Informática da
Universidade Tecnológica Federal do Paraná.

SUMÁRIO

CONVERSA INICIAL

Caro estudante:

Neste módulo, você terá condições de desenvolver seus conhecimentos sobre Algoritmo, pois trataremos aqui dos fundamentos e da prática desse conteúdo.

Você terá oportunidade, ao longo da leitura das unidades, de perceber que a utilização da linguagem algorítmica faz parte de seu cotidiano e é fundamental para o conhecimento da área de informática.

Nesse sentido, elaboramos uma dinâmica de organização do conteúdo, partindo de uma introdução sobre a temática, passando pelos conceitos fundamentais dela e buscando, por meio de exemplos, sua compreensão sobre algoritmo e a importância deste na formação do técnico em Informática.

Assim, na **unidade I** você será introduzido nos conceitos de lógica, algoritmo, dados, informação, processamento de dados e lógica de programação.

Daremos continuidade a essa iniciação, na **unidade II,** ao tratarmos de conceitos que devem ser conhecidos antes do estudo da programação com o uso de pseudocódigos, tais como tipos de dados, variáveis e expressões.

Na **unidade III,** você conhecerá a programação hipotética muito usada em todo o mundo, ou seja, a pseudocódigo, e entrará em contato com a representação de algoritmos através dessa linguagem e sua abordagem com estruturas condicionais.

Na **unidade IV,** haverá um aprofundamento sobre pseudocódigos, fazendo uso de estruturas de repetição.

Na **unidade V** serão abordados conjuntos, strings e métodos de classificação e pesquisa.

Finalmente, na **Unidade VI,** você terá oportunidade de conhecer subalgoritmos e registro.

Assim, esperamos que, ao final desta disciplina, você esteja habilitado a:

- Desenvolver algoritmos através de divisão modular e refinamentos sucessivos;
- Interpretar algoritmos nas formas de pseudocódigo;
- Desenvolver algoritmos e representá-los por meio de linguagem de programação procedural;

- Avaliar resultados de testes dos algoritmos desenvolvidos;
- Detectar e corrigir erros em algoritmos;
- Integrar módulos desenvolvidos separadamente.

Sugerimos que você dedique tempo suficiente para fazer a leitura, realizar as atividades e retirar suas dúvidas. Sempre que considerar necessário, volte ao texto, refaça as atividades! Não se limite a este material, faça pesquisas, converse com professores e colegas. Você verá que aprender é uma interessante aventura!

Bom estudo!

Everton Coimbra de Araújo

UNIDADE I

PROGRAMAÇÃO: ANTES DO INÍCIO

Nesta unidade, você estudará alguns conceitos relativos à Lógica, Algoritmo, Dados, Informação, Processamento de Dados e Lógica de Programação. Estes conceitos e observações objetivam uma sustentação científica aos temas propostos.

Organizamos esse conteúdo a fim de que você perceba a importância de cada tema aqui trabalhado para a aprendizagem sobre programação e de como efetuar o levantamento e a identificação de componentes necessários para a resolução de um problema. Além disso, fizemos uma introdução às técnicas de programação.

Os problemas que utilizamos nesta unidade você encontra em seu cotidiano, o que poderá colaborar para iniciarmos um processo de raciocínio, buscando a identificação do problema e de todos os componentes necessários para sua resolução.

Inicialmente, a representação de resolução de algoritmos será a Descrição Narrativa, na qual os passos de resolução são expressos em linguagem natural, como se fossem regras para utilização de determinado aparelho, um manual.

Assim, ao final desta leitura esperamos que você tenha condições de:

- Definir os fundamentos necessários para a lógica de programação;
- Identificar que, para um processamento de dados, é preciso que dados sejam informados;
- Verificar que, para chegar a um resultado e fornecer informações, um processamento de dados é necessário.

1. CONCEITOS BÁSICOS

Nesta seção, você irá estudar conceitos que são fundamentais para sua formação em Informática: Lógica, Algoritmo, Dados e Informação, Dados de Entrada e Saída, Processamento de Dados e Lógica de Programação.

1.1. LÓGICA

- Você já ouviu falar em Lógica? Não? Mas, certamente já ouviu alguém responder algo do tipo: 'é lógico que eu sei", ou "é lógico que isso não daria certo". Não é?
- Mas será que, quando damos uma resposta assim, estamos tendo pensamento lógico?
- Você saberia dizer o que é Lógica?

Escreva, a seguir, o que você entende por lógica.

_____

_____

_____

Pois bem, a palavra lógica vem do grego clássico λογικὴ e foi criada pelo filósofo grego Aristóteles no século IV a.C. para estudar o pensamento humano e distinguir interferências e argumentos certos e errados.

A Lógica é uma ciência de índole matemática, fortemente ligada à Filosofia. É também a designação para o estudo de sistemas prescritivos de raciocínio, ou seja, sistemas que definem como se "deveria" realmente pensar para não errar, usando a razão, dedutivamente e indutivamente.

Assim, um sistema lógico é um conjunto de axiomas e regras de inferência que visam representar formalmente o raciocínio válido, dedutivo ou indutivo.

Você saberia dizer o que é raciocínio ou pensamento dedutivo e indutivo?
Dedutivo: _____

Indutivo: _____

Para que você compreenda melhor o que são esses dois tipos de pensamento e como se diferenciam, acompanhe minha explicação e os exemplos.

O pensamento dedutivo se caracteriza por apresentar conclusões que devem, necessariamente, ser verdadeiras caso todas as premissas sejam verdadeiras.

Exemplo: Todo ser humano é mortal.
O homem é um ser humano
Portanto, ele é mortal.

Lembra-se do Sherlock Holmes, do autor Arthur Conan Doyle? As respostas aos enigmas policiais eram encontradas por Sherlock mediante pensamento dedutivo, que se traduzia na tão conhecida frase: "Elementar, meu caro Watson".

Já o pensamento indutivo significa partir de premissas particulares, na busca de uma lei geral, universal.

Por exemplo: O ferro conduz eletricidade
O ferro é metal
O ouro conduz eletricidade
O ouro é metal
O cobre conduz eletricidade
O cobre é metal
Logo: os metais conduzem eletricidade.

Os indutivistas acreditavam que as explicações para os fenômenos advinham unicamente da observação dos fatos. Por isso eles poderiam produzir resultados falsos.

Por exemplo: O cavalo, o burro e a mula são quadrúpedes.
O cavalo, o burro e a mula são mamíferos.
Logo: Todos os mamíferos são quadrúpedes.

Esta indução é falsa, pois apresenta duas premissas verdadeiras, no entanto a generalização de conclusão é falsa.

Portanto, podemos concluir que

a Lógica é a ciência das formas do pensamento. A Lógica estuda a correção do raciocínio, visto que ele é a forma mais complexa do pensamento. Podemos dizer que a Lógica visa à ordem da razão, isto é, a razão pode funcionar desordenadamente e a lógica estuda e ensina a colocar ordem no pensamento.

Agora, tente você elaborar um raciocínio dedutivo e outro indutivo!
Dedutivo: _____

Indutivo: _____

Muito bem!

Agora que você fez o exercício, prossiga sua leitura da seção 2, que trata dos algoritmos.

Mas, o que a Lógica tem a ver com algoritmos?

1.2. ALGORITMO

Certamente você já ouviu falar em algoritmo, não?
Você consegue se lembrar o que é algoritmo? E automação, você saberia dizer o que é?

Agora, então, você terá oportunidade de relembrar esse assunto e, principalmente, verificar que não há como falar de algoritmos sem antes tecer comentários sobre automação.

Automação é um processo em que uma tarefa deixa de ser desempenhada pelo homem e passa a ser realizada por máquinas, não importando se estas máquinas são mecânicas ou eletrônicas (SALIBA, 1993)

Vamos ver como pode ocorrer uma automação?

Determinada tarefa, para que seja automatizada, deve ter todas as etapas conhecidas, e a máquina ou dispositivo que desempenhará este processo deverá estar apta a garantir sua repetibilidade.

O objetivo maior da automação é que o resultado de uma tarefa possa ser

conhecido ou obtido por várias vezes, no mesmo intervalo de tempo e com a mesma qualidade.

Exemplos:
São exemplos de automação os caixas eletrônicos dos bancos, que hoje realizam tarefas antes realizadas pelos bancários, e as funções mecânicas de montagem na linha de produção automotiva.

Observe que cada situação-problema, ou processo para ser solucionada precisa seguir algumas etapas que podemos chamar de especificação de sequência ordenada de passos. A esta especificação damos o nome ALGORITMO.

PARA REFLETIR

Veja estes exemplos de algoritmo que você pode observar no seu dia a dia:

- resolver uma operação matemática seguindo passos até chegar ao resultado final;
- ligar o forno de micro-ondas seguindo as etapas de conectar à energia, apertar determinados botões, sequencialmente, até que a máquina seja ligada;
- tomar medicamentos seguindo orientação médica;
- fazer receitas culinárias.

ATIVIDADE

E agora, você consegue pensar em outros exemplos? Então escreva no espaço abaixo pelo menos mais dois exemplos de algoritmos que você encontrou em seu dia a dia.

a)_____

b)_____

1.3. DADO E INFORMAÇÃO

Você saberia dizer o que significa "dado" em Informática?

Podemos dizer que o dado é a **matéria-prima** da informação, ou seja, a informação é composta de um conjunto organizado de dados. O dado puro e simples, muitas vezes, é confundido com a própria informação, não carrega obrigatoriamente nenhum caráter informativo.

Imagine que cada letra deste texto estivesse solta na página, ou seja, que as letras não estivessem presas por nenhum processo tipográfico. Ao sacudirmos a página, as letras cairiam no chão, concorda?

Os montinhos de letras espalhadas pelo chão conteriam todos os dados que constituem a informação deste texto, sendo, porém, impossível constituí-lo novamente apenas recolocando as letras sobre o papel. Isso é o que pode acontecer com um dado puro e simples.

Cada tipo de informação está associado a determinado conjunto de dados, criados especificamente para se adequar aos processos de transmissão da informação. Quando falamos, trabalhamos com fonemas que, através do sistema vocal transmitem informações, respeitando regras linguísticas e idiomáticas. Quando escrevemos, trabalhamos com letras que, ao serem ligadas entre si, formam palavras (o que já é uma informação), que formam frases.

Um mesmo conjunto de dados pode ser utilizado para representar informações de natureza distinta.

Veja o exemplo da escrita na língua portuguesa, que utiliza o alfabeto formado por letras, e a escrita na língua japonesa, que utiliza símbolos.

Há também os casos em que a linguagem escrita se socorre do mesmo alfabeto, porém com idiomas distintos. Sendo assim, podemos verificar que, apesar de utilizar o mesmo alfabeto, a língua portuguesa é distinta das demais línguas (francês, inglês, alemão, etc.). Esse é um exemplo de que, quando falamos em dados, podemos encontrar um mesmo veículo e um mesmo conjunto de dados, com formas de informação totalmente distintas, que dependerão apenas das técnicas de união dos dados.

Este material que você está lendo foi redigido em língua portuguesa brasileira. O que você acha que aconteceria se ele tivesse que ser utilizado por um estudante que dominasse apenas a língua francesa? O mesmo conjunto de dados (alfabeto comum a muitas línguas) está sendo utilizado, mas, nesse caso, haveria transmissão da informação?

Com base em suas reflexões, cremos que já é possível enunciar uma definição que atenda ao significado de dado e informação. Vamos às definições?

Dado:	símbolo que expressa a unidade mínima da informação (átomo da informação).
Informação:	conjunto de dados reunidos com regras específicas à natureza da informação.

1.4. DADOS DE ENTRADA E SAÍDA

Quando um problema surge em nosso dia a dia, geralmente ele ocupa totalmente nossa atenção, e isso pode interferir em nossa capacidade de percepção.

Ai... Tirei zero...
Ai... Tirei zero...
Ai... Tirei zero...

Na ilustração acima, há uma preocupação fundamental: a média baixa na avaliação. O estudante tem como foco principal o problema. Não há uma

preocupação com os dados que levaram àquela situação.

Em sua opinião, o que levou o aluno a tirar uma nota abaixo da média? Que ações ou atitudes anteriores colaboraram para que obtivesse aquele resultado?

Visualizando o problema e os componentes causadores dele, podemos identificar dados que auxiliarão em sua resolução. Há situações em que esta identificação é imediata, apesar de não percebermos. Porém, em situações mais complexas, um exaustivo trabalho de identificação de probabilidades e condições se faz necessário.

A estes componentes, probabilidades e condições damos o nome de dados de entrada, pois é por meio do processamento destes dados, com o uso de um algoritmo, que chegamos à solução do problema, que gerará uma saída de dado, ou seja, a resposta, informação ou resultado desejado.

1.5. PROCESSAMENTO DE DADOS

Você saberia dizer qual a relação entre informação e comunicação? Qual o sentido da informação se ela não puder ser comunicada? E qual o sentido da comunicação se não houver informação?

A informação pura e simples nada significa se não for transmitida. Ao mesmo tempo, só podemos afirmar a existência de uma comunicação se houver informação para ser enviada.

Então, observe os conceitos de comunicação e informação:

Comunicação pode ser considerada o intercâmbio de informação entre sujeitos ou objetos.

Informação é o resultado do processamento, manipulação e organização de dados de tal forma que represente uma modificação (quantitativa ou qualitativa) no conhecimento do sistema (pessoa, animal ou máquina) que a recebe.

Por isso, embora sejam distintas em suas definições, só podemos justificar uma com a existência da outra.

Podemos dizer que a informação só existe após o término de um processo que envolve:

- A elaboração dela mesma;
- A transmissão dos dados que formam a informação;
- A absorção, por outra entidade diferente daquela que originou a informação, de todos os dados organizados de maneira idêntica à informação original.

O universo dos conjuntos de dados associados a um veículo de comunicação é extenso. Para que a transmissão da informação seja realizada com sucesso é necessário:

- que a entidade geradora saiba codificar a informação com o conjunto de dados escolhidos;
- que a entidade receptora saiba decodificar a informação, por meio do conjunto de dados escolhidos para a transmissão.

1.6. LÓGICA DE PROGRAMAÇÃO

Quando falamos em lógica de programação, nós nos remetemos à sequência de instruções.

Você sabe o que é instrução?

Instrução pode ser considerada um conjunto de regras ou normas definidas para a realização de uma tarefa. Em informática, instrução é a informação que indica a um computador uma ação elementar a executar.

Nas disciplinas de programação, quando nos é dado um problema, o primeiro passo é determinar uma sequência de instruções tal que, fornecidos os dados de entrada, por meio da execução da sequência de instruções alcancemos como saída a solução do problema.

Agora, para que você exercite seus conhecimentos sobre lógica de programação, vamos lhe propor um desafio, por meio de um estória muito conhecida:

Um senhor está numa das margens de um rio com uma raposa, uma dúzia de galinhas e um saco de milho. Ele pretende atravessar o rio com sua carga, num barco que só comporta ele e parte de sua carga. Ele não pode deixar a raposa com as galinhas, nem as galinhas com o milho. O que fazer para atravessar o rio e chegar à outra margem com a raposa, as galinhas e o milho?

Ajude o senhor a resolver esse empasse! Porém, tente encontrar a resposta antes de prosseguir a leitura.

Descreva-a no quadro abaixo:

Você conseguiu chegar à resposta? Então, parabéns! Caso contrário, acompanhe o caminho por mim percorrido para solucionar o problema.

Geralmente, os estudantes tentam partir para a resolução imediata, alguns chegam a desenhar duas margens separadas por um rio e os demais personagens tentando encontrar a solução. Poucas são as situações em que ocorre a preocupação em avaliar a essência do problema, ou seja, as características de cada personagem, os dados de entrada.

A identificação do problema é relativamente simples, o próprio texto a retrata muito bem, sem necessidade de informação complementar.

Os componentes fornecidos também são suficientes, bastando uma análise detalhada às características de cada um deles.

Já os dados de saída, neste caso, não são informações, e sim uma maneira de resolver o problema proposto.

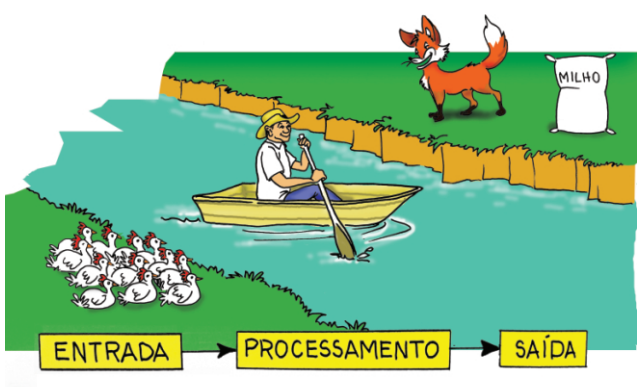
Avaliando as características de cada componente, identificamos que, evidentemente, o senhor não pode deixar em uma das margens, sozinhos, a raposa e as galinhas. Fatalmente, a raposa comeria as galinhas. Nem as galinhas e o milho, pois as galinhas comeriam os grãos.

A questão é escrever uma sequência de passos (um algoritmo) que processe estes dados e oriente o senhor a realizar seu intento, sem que haja perda.

É óbvio que, na primeira viagem, ele não pode levar a raposa, pois as galinhas comeriam o milho.

Também não pode levar o milho, pois a raposa comeria as galinhas.

Como resolver esse problema?



Veja o algoritmo (sequência de passos) identificado (dados de saída) para a resolução do problema:

Atravesse as galinhas
Retorne sozinho
Atravesse a raposa
Retorne com as galinhas
Atravesse o milho
Retorne sozinho
Atravesse as galinhas.

2. DESCRIÇÃO NARRATIVA

O que você imagina que seja "descrição narrativa"?

Ao depararmos com um problema, cuja necessidade de automação é iminente,

precisamos ter certeza de que o compreendemos e de que sabemos o que se espera da solução para depois, sim, iniciarmos o trabalho de programação.

Não há como aceitar uma resposta sem sabermos se é ela que se deseja. Para isso, quando essa situação ocorre, uma ampla análise do problema em questão se faz necessária.

Por meio desta análise, é possível identificar alguns dos passos que devem ser seguidos e respeitados para que seja possível a resolução da situação-problema. Esses passos compõem a tabela usada para estudo de um problema e consequentemente servirão como base para a construção da **Descrição Narrativa** de como este problema deve ser resolvido.

Esta tabela e estes passos serão amplamente discutidos e exemplificados na sequência.

Entretanto, antes disso, vale a pena observarmos que uma das maiores dificuldades encontradas pelos estudantes na disciplina de algoritmos é a interpretação de texto.

Não conseguir identificar aquilo que se pede (problema), o que se espera obter (dados de saída), o que será necessário para o processamento (dados de entrada) e a maneira de processar esses componentes dificulta muito o processo de resolução.

Portanto, aproveitamos para destacar a importância do exercício diário da leitura, seja no contexto escolar, seja no profissional, seja no lar. Mediante o hábito da leitura, firmamos ainda mais a compreensão ou interpretação do que lemos e essas competências são fundamentais para todas as áreas do conhecimento, não somente quando tratamos da Língua Portuguesa, como equivocadamente pensam muitos estudantes.

Nos casos descritos a seguir nos próximos tópicos, por exemplo, você precisará interpretar o texto para identificar os elementos necessários à sua resolução.

Portanto, exercite a leitura e verá como isso poderá ajudá-lo não somente em informática, mas em todas as áreas do conhecimento, ok?

2.1. RESOLUÇÕES SEQUENCIAIS

Agora, observando o exemplo abaixo, vamos verificar se o texto que traz o problema (enunciado) teve uma correta interpretação. Para isso, propomos o preenchimento de uma tabela como mostrada a seguir.

PROBLEMA I

De três prisioneiros que estavam em cárcere, um tinha visão normal, o segundo tinha apenas um olho e o terceiro era cego. Os três eram, pelo menos, de inteligência média.

O carcereiro disse aos prisioneiros que, de um jogo de três chapéus brancos e dois vermelhos, escolheria três e os colocaria na cabeça deles. Cada um deles

estava proibido de ver a cor do chapéu em sua própria cabeça.

Reunindo-os, o carcereiro ofereceu a liberdade ao prisioneiro com visão normal se ele fosse capaz de dizer a cor do chapéu que tinha na cabeça.

O prisioneiro confessou que não podia dizer e se retirou.

A seguir, o carcereiro ofereceu a liberdade ao prisioneiro que só tinha um olho na condição de que ele dissesse a cor de seu chapéu.

O prisioneiro confessou que também não sabia dizê-lo e também se retirou.

O carcereiro não se deu o trabalho de fazer a idêntica proposta ao prisioneiro cego, mas à insistência deste, concordou em dar-lhe a mesma oportunidade.

O prisioneiro cego abriu um amplo sorriso e disse: "Não necessito da minha visão. Pelo que meus amigos disseram, vejo claramente que meu chapéu é branco.

Tabela I: Interpretação do problema I

O Problema	De três prisioneiros, verificar qual ou quais identificam a cor do chapéu posto em sua cabeça, sem olhá-lo. **Observações complementares** Existem cinco chapéus: três brancos e dois vermelhos.
Solução esperada	Cada preso deve identificar a cor do chapéu que tem em sua cabeça.
Dados de entrada	**Detalhamento dos Dados de Entrada** Chapéus *Existem duas cores possíveis entre os cinco existentes*
Dados de saída	Neste problema em particular, não se obtêm dados ou informações ao seu término, apenas a resolução do problema, que é a identificação da cor do chapéu sobre a cabeça do prisioneiro.
Etapas encontradas	- Colocar na cabeça de cada preso um chapéu; - Escolher um dentre os três prisioneiros para perguntar a cor do chapéu sobre sua cabeça; - Efetuar a pergunta para todos.
Descrição Narrativa da solução encontrada	1. Ponha os chapéus na cabeça dos presos; 2. Pergunte ao primeiro preso a cor de seu chapéu; 3. Pergunte ao segundo preso a cor de seu chapéu; 4. Pergunte ao terceiro preso a cor de seu chapéu;

Você percebeu que o preenchimento desta tabela facilita a compreensão dos elementos que compõem o problema? E você já identificou que dados levaram o prisioneiro cego a identificar qual a cor do chapéu lhe foi colocado sobre a cabeça?

A tabela apresentada não é regra, nem pré-requisito para resolução de algoritmos, porém pode ser vista como uma ótima ferramenta para apoio ao início de um aprendizado tão exigente como Programação.

Atente bem a cada um de seus itens, tente identificar a real importância de eles existirem. Caso identifique novos itens relevantes, inclua-os. Caso acredite que nem todos são necessários para a resolução de seus algoritmos, retire-os. Lembre-se de que uma ferramenta deve ter o propósito de auxiliar, e não dificultar. No entanto, ela só poderá auxiliar se você souber como utilizá-la.

Observe, no problema anterior, que a escolha de quem será interrogado antes faz parte do problema, até de forma racional, pois primeiro se pergunta para quem tem mais condições visuais, assumindo-se que talvez, por ter visão total, terá condições de identificação.

Poderá ser notado que a resolução deste algoritmo não está diretamente ligada aos passos que devem ser percorridos, mas sim ao raciocínio lógico do terceiro preso.

Sendo assim, vejamos o raciocínio utilizado por ele para saber seguramente a cor do chapéu em sua cabeça.

a) O primeiro prisioneiro só poderia ter visto na cabeça dos outros dois as seguintes combinações de chapéus:

2° Prisioneiro	3° Prisioneiro
Vermelho	Vermelho
Branco	Branco
Branco	Vermelho
Vermelho	Branco

A primeira possibilidade está descartada, pois ele teria acertado, uma vez que só existiam dois chapéus vermelhos. Sendo assim, restaram as três últimas.

b) O segundo prisioneiro, tendo em vista as possibilidades que restaram, só poderia ter visto:

3° Prisioneiro
Branco
Vermelho
Branco

Como a primeira e a terceira são iguais, ele só pode ter visto vermelho ou branco. Se ele tivesse visto vermelho, teria certeza de que seu chapéu era branco, pois se fosse vermelho, o primeiro teria acertado. Logo, ele só poderia ter visto chapéu branco no terceiro prisioneiro; daí sua dúvida: o dele seria branco ou vermelho?

c) Foram essas as conclusões que o terceiro tirou para dizer que o chapéu dele era ***branco***.

Você deve ter percebido que o terceiro prisioneiro usou a lógica, conceito já estudado nesta unidade, para responder acertadamente sobre a cor de seu chapéu, ou seja, baseando-se nos dados que tinha, ordenou o raciocínio e chegou à resposta.

A situação exposta neste exemplo reflete o que você precisará sempre ter em mente:

"É necessário saber resolver o problema antes de usar ferramentas para fazê-lo."

O programador tem que ter total domínio do problema e do ambiente, pois só desta maneira ele poderá aplicar linguagens de programação e outras ferramentas que o auxiliem a resolvê-los.

Nos problemas resolvidos, há grande possibilidade de maior detalhamento.

Para isso, é necessário sempre um estudo pormenorizado do ambiente em que se deseja promover a automação. Os exercícios descritos e resolvidos tratam os problemas de uma forma generalizada, porém, se durante a leitura e aplicação você encontrar particularidades que devam ser tratadas em decorrência de novo ambiente, faça-o como exercício de abstração.

O exemplo anterior apresentou um pequeno problema, com uma única possibilidade de solução, não havendo condições ou pré-requisitos para tomar um caminho ou outro, ou ainda encontrar uma situação em que inviabializaria a resolução do problema. A este tipo de resolução, dá-se a característica de sequencial, pois cada passo ocorre um após o outro, sem desvio.

2.2. RESOLUÇÕES CONDICIONAIS

Não é difícil imaginar o que são resoluções condicionais, sim?

O próprio nome nos remete ao conceito. Resoluções condicionais são necessárias quando, diante de uma nova situação, temos que lidar com "condições" para que determinada tarefa seja desempenhada.

A identificação destas condições faz parte da identificação do problema e, por isso, muitas vezes deparamos com a dificuldade de encontrá-las e de como resolvê-las.

O exemplo a seguir tem a característica de uma estrutura condicional, veja:

PROBLEMA II

O governo federal implantou um plano de apoio às famílias de baixa renda, que consiste na entrega de cesta básica. Foi determinado um local para distribuição destas cestas, onde se encontra uma pessoa que solicita algumas informações para cada cidadão que para lá se dirige. Sendo identificada a veracidade das informações, é entregue uma cesta básica a cada cidadão.

Tabela II: Interpretação do problema II

O Problema	Entregar cestas básicas à população com baixa renda. **Observações complementares** - Não são todos que têm direito à cesta básica; - Deve-se, através de perguntas, identificar se o cidadão tem direito, ou não, à cesta básica.
Solução esperada	Entrega de cesta básica apenas para quem realmente necessite.
Dados de entrada	**Detalhamento dos Dados de Entrada** **Questionário** Conjunto de perguntas que permitirão a identificação da necessidade, ou não, de cesta básica
Dados de saída	Se for identificada a necessidade de cesta básica, deve ser entregue ao cidadão, caso contrário, deve ser dito a ele que não possui direito.
Etapas encontradas	- Solicitar ao cidadão o preenchimento do questionário; - Avaliar o questionário e identificar a necessidade; - Proceder com o cidadão de acordo a avaliação obtida pelo questionário.

Descrição Narrativa da solução encontrada	1. Entregar questionário para cidadão preencher; 2. Avaliar o questionário preenchido; 3. **Se** o cidadão necessitar de cesta básica, **então** proceda à entrega; 4. **Se** o cidadão não necessitar de cesta básica, **então** informe que não tem direito.

No problema exposto acima, o enunciado praticamente esclarece todas as dúvidas e expõe todos os passos a serem cumpridos para chegarmos à solução, bastando apenas uma especial atenção ao texto, para, assim, a interpretação ser perfeita.

O objetivo é entregar cesta básica às famílias de baixa renda. Mas qual família é de baixa renda? Como saber disso?

Neste caso, não interessa as particularidades, pois o texto diz que as informações serão solicitadas por um responsável e a ele compete esta identificação, cabendo a seu programa apenas trabalhar com a informação passada pelo responsável. Uma vez identificado que a família é de baixa renda, entrega-se a cesta. Caso contrário, apenas informa quais famílias não têm direito à cesta básica.

2.3. RESOLUÇÕES COM PONTOS DE REPETIÇÃO

a) Repetição contada

Sem dúvida, pudemos perceber situações condicionais no exemplo anterior, mas ainda existem ocasiões em que determinado problema, para ser resolvido, necessita de um processo que pode se repetir por várias vezes, podendo esta quantidade ser predeterminada ou condicional, e isso também deve ser trabalhado. A esta característica é dado o nome de repetição.

Veja o exemplo:

PROBLEMA III

Em um instituto de pesquisa voltado à criação de pinguins, foi levantada a temperatura de todos os dias do mês de novembro. Encontre a quantidade de dias com temperatura positiva, a quantidade de dias com temperaturas negativa, além da média da temperatura nos dias quentes.

Tabela III: Interpretação do problema III

O Problema	Encontrar a quantidade de dias de novembro com temperatura positiva e negativa, além da média de temperatura nos dias em que esta foi positiva. **Observações complementares** - O mês de novembro tem 30 dias; - Nem todas as temperaturas obrigatoriamente são positivas ou negativas.

Solução esperada	Apresentação da quantidade de dias do mês de novembro com temperatura positiva e negativa, além da média de temperatura nos dias em que a mesma foi positiva.

Dados de entrada	**Detalhamento dos Dados de Entrada** **Temperatura** Deve ser verificado se é positiva ou negativa.

Dados de saída	- Quantidade de dias com temperatura positiva; - Quantidade de dias com temperatura negativa; - Média da temperatura nos dias positivos.

Etapas encontradas	- Solicitar temperatura para cada dia; - Identificar se a temperatura é positiva ou negativa; - Somar as médias positivas e efetuar o cálculo da média.

Descrição Narrativa da solução encontrada	1. Solicitar temperatura **para cada dia** do mês de novembro; 2. **Se** a temperatura informada for positiva, deve-se somá-la acumulando esta soma, e também somar, acumulando, a quantidade de dias com temperatura positiva; 3. **Se** a temperatura informada for negativa, deve-se somar e acumular a quantidade de dias com temperatura negativa; 4. Obtidas todas as temperaturas, deve-se informar a média delas em dias de temperatura positiva.

Este problema é interessante, pois trata uma situação que se repete um número de vezes conhecido, o que chamamos de repetição contada.

Nem sempre o problema traz a quantidade de vezes que um processo se repetirá. Normalmente essa informação vem de forma implícita, como neste texto.

Você observou que não foi dito quantas vezes é preciso solicitar a temperatura?

Mas você viu que foi dito que a situação se passa em novembro, ficando mais fácil definir a quantidade de repetições, não é?

Um outro detalhe que não pode passar despercebido é que a média desejada não é de todas as temperaturas, apenas das positivas. Logo, é necessário saber quando a temperatura é positiva e, quando o for, somar estes valores.

Observe que não podemos, além disso, dividir este valor pela quantidade de temperatura, apenas pela quantidade identificada como positiva.

Outra solicitação é a quantidade de dias negativos. Esta é a mais fácil, pois já identificamos a quantidade de dias positivos para chegar à média e temos o total

dos dias. Basta deduzirmos que os negativos são o resultado de uma subtração entre o total de dias e os dias com temperatura positiva.

b) Repetição contada e condicional - aninhadas

Para você verificar como ocorre uma repetição contada e condicional "aninhada", acompanhe o problema a seguir:

PROBLEMA IV

Uma vinícola deseja saber o percentual de vinho branco seco vendido durante determinado mês. São vários os tipos de vinho comercializados, assim como são vários os clientes da empresa. A quantidade desejada se refere à venda total da vinícola no mês em referência.

Por questões da estrutura de como os dados são utilizados neste algoritmo, deve-se, para cada dia, verificar cliente por cliente, os tipos de vinho e respectiva quantidade comprada.

Tabela IV: Interpretação do problema IV

O Problema	Identificar o percentual de vinho branco seco vendido em determinado mês. **Observações complementares** - Não se sabe qual o mês se deseja fazer a verificação; - Cada cliente pode comprar vários tipos de vinho.
Solução esperada	Apresentação do percentual que representa a venda de vinho branco seco, perante toda a venda de determinado mês.
Dados de entrada	**Detalhamento dos Dados de Entrada** - Mês - Cliente - Tipo do vinho e quantidade comprada por cada cliente Deve ser solicitado antes do cliente, o tipo de vinho e a quantidade comprada por cliente, pois é o mês que identifica a quantidade de vezes que deve ser verificada a venda.
Dados de saída	- Percentual de vinho branco seco vendido no mês

Etapas encontradas	- Identificar o mês em que se deseja fazer esta pesquisa; - Identificar para cada dia os clientes que compraram vinhos; - Identificar a quantidade e tipo de vinho comprados por cliente; - Saber que, para cada tipo de vinho comprado, se deve verificar se é o desejado para a pesquisa: **Vinho Branco Seco**. Caso seja, deve-se somar a quantidade informada, para, ao final, fazer verificação do percentual; - Devem-se somar todos os vinhos comprados por todos os clientes, pois o percentual deverá ser encontrado sobre o total comercializado; - Calcular e informar o percentual desejado.

Descrição Narrativa da solução encontrada	1. Solicitar o **mês** em que se deseja fazer a pesquisa 2. Solicitar, **para cada dia do mês,** os clientes que compraram vinhos na vinícola. 3. Solicitar, **para cada cliente,** o **tipo de vinho** e a **quantidade comprada**. 4. **Se** o tipo de vinho for **Branco e Seco**, deve-se **acumular** a quantidade informada. 5. **Para cada tipo de vinho informado**, independentemente de ele ser do tipo desejado para pesquisa ou não, deve-se acumular a quantidade. 6. Após a informação do tipo e quantidade comprada pelo cliente, deve-se verificar **se o cliente comprou mais vinho**. 7. Caso a resposta seja positiva, deve-se retornar ao **item 3** Caso contrário, deve-se verificar **se há mais clientes que compraram vinho neste dia**. 8. Caso a resposta seja positiva, deve-se retornar ao **item 2.** Caso contrário, deve-se prosseguir. 9. Uma vez encontrado todos os valores, deve-se efetuar o cálculo do percentual referente às vendas de **Vinho Branco Seco** e informá-lo ao usuário.

Este exercício nos mostra uma situação um pouco mais curiosa que os anteriores: há processos repetitivos dentro de um processo repetitivo. A esta característica damos o título de repetição aninhada.

É interessante observamos que, neste caso, o único laço repetitivo que se pode chamar de repetição contada é o mais externo, o que se referencia aos dias do mês utilizado para a pesquisa.

Aos laços internos denominamos de laços condicionais, pois sua continuidade depende de uma situação que nem sempre se conhece, mas se pode prever. Há casos em que esta condição é alcançada por meio de processos do próprio algoritmo (como em uma repetição contada).

Há casos em que esta condição é alcançada em decorrência de uma interação com o usuário, como ocorre no exemplo, em que se pergunta se o cliente comprou ou não mais vinho. Isso após a avaliação total da primeira interação do laço.

Chegamos ao final dessa unidade, em que você teve a oportunidade, na seção de Conceitos Básicos, de identificar o que é Lógica e a importância desse conceito para aprender algoritmo. Você estudou o conceito de algoritmo e automação. Pôde verificar o que é dado e informação e a relação entre esses dois conceitos.

Você estudou sobre dado de entrada e de saída e verificou como é importante não observar o problema em si, mas os componentes, as probabilidades e as condições para sua resolução. No item Processamento de Dados, você observou a relação entre comunicação e informação. Em Lógica de Programação você viu o que é instrução e como a execução da sequência de instruções permite chegar à resolução do problema.

Na seção 2 - Descrição Narrativa - você pôde conhecer um pouco sobre linguagem e a importância da leitura para identificar um problema. No item Resoluções Sequenciais, por meio de exemplos, você pôde verificar os processos para chegar à solução de um problema.

No item Resoluções Condicionais, você observou que a identificação das condições faz parte da identificação do problema e que a resolução deste depende de verificar quais são essas condições.

No item Resoluções com Pontos de Repetição, você verificou que, além das situações condicionais, um processo pode se repetir várias vezes e esta quantidade pode ser predeterminada ou condicional.

No item Repetição Aninhada, você teve oportunidade de verificar que processos repetitivos podem ocorrer dentro de um processo repetitivo.

Você deve ter percebido, pelo estudo dessa unidade, que falamos um pouco sobre os conceitos básicos que envolvem as técnicas de programação. São noções que precisam ser compreendidas de modo significativo para uma futura compreensão de toda a lógica envolvida nos processos de programação na área da Informática.

Espero que você tenha tido condições de observar que, por exemplo, a descrição narrativa é uma maneira simples de se programar, por isso você deve ter identificado que, se você faz descrições narrativas todos os dias, você é capaz de programar, sim?

Bem, agora que você já teve contato com as primeiras conceituações, podemos dar um passo adiante!

Realmente, espero que nossa conversa nesta unidade tenha fornecido a você os subsídios mínimos para continuidade do trabalho! Mas, se ainda restam dúvidas, leia esta unidade quantas vezes julgar necessário.

Lembre-se sempre que você é capaz!

Até a Unidade II!

UNIDADE II

TIPOS DE DADOS, VARIÁVEIS, EXPRESSÕES, SINTAXE E SEMÂNTICA

Na unidade anterior, buscamos introduzir você no campo do raciocínio lógico e de algumas ferramentas para resolução de problemas.

Nesta segunda etapa, você terá contato com alguns conceitos que deve conhecer antes de iniciar a programação com uso de pseudocódigos: **tipos de dados**, **variáveis** e **expressões.** Depois disso, passaremos a estudar conceitos, exemplos e a prática de programar, usando **pseudocódigo** para resolução de algoritmos.

Assim, por meio do estudo desta unidade esperamos que você seja capaz de:

- Compreender o que são e quais são os tipos de dados básicos;
- Definir e compreender o uso de variáveis;
- Conhecer a definição e o correto uso de expressões e operadores;
- Compreender, identificar e aplicar sintaxe e semântica.

2.1. TIPOS DE DADOS

Salvetti & Barbosa (1998) ressaltam que um programa de computador é descrito em uma linguagem de programação. Geralmente, cada linguagem de programação tem seus próprios tipos de dados, isto é, conjunto de valores, operações e relações já implementadas (disponíveis para uso). Na implementação surge o conceito de domínio, isto é, da limitação do conjunto de valores dos elementos representados.

Os tipos de dados que serão tratados nesta sessão são classificados de acordo com o tipo de informação contida neles. Lembramos que a classificação apresentada aqui não se aplica a nenhuma linguagem de programação específica, pois a idéia é mostrar, de forma sintetizada, os padrões utilizados na maioria das linguagens.

2.1.1. DADOS NUMÉRICOS

Os dados numéricos são divididos, basicamente, em dois grandes conjuntos: **inteiros** e **reais**. Veremos nas definições que tudo que foi aprendido nas aulas básicas de matemática, é totalmente utilizado no conceito destas duas classes de tipos de dados numéricos.

Números inteiros são aqueles que não possuem componentes decimais ou fracionários, podendo ser positivos ou negativos.

Esse é fácil! Quer ver alguns exemplos?

Vamos lá!

36 é um número inteiro positivo
0 é um número inteiro
- 8 é um número inteiro negativo

Os dados de tipo **Real** são aqueles que podem possuir componentes decimais ou fracionários, podendo também ser positivos ou negativos.

Bem, agora vejamos...

Podemos citar como exemplos de dados de tipo real:

36.01	é um número real positivo com duas casas decimais
166.	é um número real positivo com zero (nenhuma) casa decimal
- 18.8	é um número real negativo com uma casa decimal
0.0	é um número real com uma casa decimal
0.	é um número real com zero (nenhuma) casa decimal.

É importante observar que há uma diferença entre **0**, que é um dado do tipo inteiro, e **0.** ou **0.0,** que são dados do tipo real. Portanto, a simples existência do ponto decimal serve para diferenciar um dado numérico do tipo **inteiro** de um tipo **real**.

2.1.2. DADOS LITERAIS

O que é um dado literal? Você saberia dizer?

O tipo de dados **literal**, conforme Saliba (1993), pode ser definido como constituído por uma sequência de caracteres com letras, dígitos e/ou símbolos especiais. Este tipo de dado é também muitas vezes chamado de ***alfanumérico***, ***cadeia de caracteres*** ou, ainda, ***String***.

Usualmente, os dados literais são representados nos algoritmos pela coleção de caracteres, delimitada em seu início e término com o caractere aspas (“). É comum, em algumas linguagens, a diferenciação entre a representação de um único dado literal, que é chamado de caractere (por exemplo: **'A'**) e um conjunto de caracteres, chamado de **String** (por exemplo: **“Olá, Mundo”**). Note que, no exemplo de caractere, foram utilizadas *aspas simples (apóstrofo)*. Já no exemplo de string, *aspas duplas*. É interessante esta diferenciação, pois, para algumas linguagens, ela é necessária (por exemplo C e Java).

O dado do tipo literal possui um comprimento dado pelo número de caracteres nele contido. Veja os exemplos:

"QUEM ?"	- Literal de comprimento 5
" "	- Literal de comprimento 1
"cOmO !?#" -	- Literal de comprimento 8
"AbcDEFghi" -	- Literal de comprimento 9
"4+5-1="	- Literal de comprimento 6
"1"	- Literal de comprimento 1

Atenção aos detalhes!

Perceba que **“1.2”** representa um dado do tipo **LITERAL** de comprimento 3, constituído pelos caracteres “1”, “.” e “2”, diferente de **1.2** que é um dado do tipo **REAL**.

2.1.3. DADOS LÓGICOS

São caracterizados, como tipos lógicos, os dados com valor verdadeiro e falso, ressaltando que este tipo de dado poderá representar apenas um dos dois valores. Ele é chamado por alguns de tipo *booleano*, devido à contribuição do filósofo e matemático inglês George Boole na área da lógica matemática.

Para facilitar a citação de um dado do tipo lógico e diferenciação entre nomes de variáveis, alguns autores/professores apresentam estes valores delimitados pelo caractere ponto (**.**).

Como exemplo deste tipo de dados, temos os valores: **.Falso.** (para o valor lógico: falso) e **.Verdadeiro.** (para o valor lógico: verdadeiro). Observe que isso não é regra, apenas convenção para alguns.

2.2. VARIÁVEIS

Para Lopes & Garcia (2002), uma *variável* é um local na *memória principal*, isto é, um endereço que armazena um conteúdo.

Para facilitar a programação, é permitido que demos nome a esse endereço. O conteúdo de uma variável pode ser de um dos vários tipos apresentados em passo anterior.

Para Forbellone & Eberspacher (2000), um dado é classificado como variável quando tem a possibilidade de ser alterado em algum instante no decorrer do tempo, ou seja, durante a execução do algoritmo, em que é utilizado, o valor do dado sofre alteração ou o dado é dependente da execução em certo momento ou circunstância.

Uma vez definidos o nome e o tipo de uma variável, não podemos alterá-los no decorrer de um algoritmo. Por outro lado, o conteúdo da variável é um objeto de constante modificação no decorrer do programa, de acordo com o fluxo de execução deste.

Quando formos dar nome às variáveis, faz-se necessário seguirmos algumas regras. É bom ressaltar que estas regras irão variar de acordo com a linguagem escolhida como ferramenta, mas a grande maioria adota as seguintes regras genéricas:

- O primeiro caractere é uma **letra**;
- Se houver mais de um caractere, só poderemos usar: **letra** ou **algarismo**;
- Nomes de variáveis escritas com letras maiúsculas serão diferentes de letras minúsculas;
- Nenhuma palavra reservada à ferramenta (linguagem de programação) poderá ser usada como nome de uma variável;
- Procure dar nomes representativos para a variável. Lembre-se de que ao ler seu nome, é importante saber o que ela contém.

As variáveis são definidas no início, pois isso permite a alocação (reserva) de uma área na memória (endereço) para a variável. Outro objetivo da declaração de variáveis é que, após a declaração, o algoritmo sabe os tipos de operação que cada variável pode realizar. Algumas operações só podem ser realizadas com variáveis do tipo inteiro. Outras só podem ser realizadas com variáveis dos tipos inteiro ou real, e outras só com variáveis de caractere, entre outras que serão vistas neste livro.

2.2.1. DECLARAÇÃO DE VARIÁVEIS

Para ilustrar uma declaração de variáveis, é necessário que sejam nominados os tipos de dados que encontramos nos problemas. Vejamos alguns tipos já vistos:

Inteiro – ***int*** ou ***integer***
Real – ***real, float*** ou ***double***
Literal – ***char*** (*um caractere*) ou ***string*** (*cadeia de caracteres*).
Lógico – ***boolean*** ou ***lógico***.

Neste caso, para nomear os tipos de dados de modo válido, precisamos retomar a premissa de que interpretá-los de modo correto é fundamental.

Que tal observarmos alguns exemplos para tornar ainda mais clara essa ideia? Vamos lá...

Sendo dada uma lista de compras com o código, quantidade e preço de oito produtos, faça um algoritmo que escreva o valor total da compra.
string CODIGO ***int*** QUANTIDADE ***float*** PRECO, VALOR TOTAL

Ao serem fornecidos um valor a ser pago e uma taxa para multa, pois o pagamento está sendo feito após o vencimento, calcule o valor da multa e o valor total a ser pago.
float VLRCONTA, TAXAMULTA, VLRMULTA, VLRTOTAL

É solicitada a um motorista, recém-chegado de uma viagem, a quantidade de quilômetros por ele percorrida. O motorista informa o solicitado, e você deverá informar a ele a quantos metros se refere a quantidade de quilômetros.
int QUILOMETROS, METROS

Observe que o trabalho de identificação de variáveis e de seus tipos é, na maioria das vezes, muito fácil, pois basta identificá-los no enunciado. O que ocorre também com grande frequência é a identificação de algumas variáveis

necessárias apenas durante a resolução do algoritmo, mas isso não é problema. Basta relacionarmos estas variáveis à lista já existente.

2.3. EXPRESSÕES E OPERADORES

O conceito de **expressão** em termos computacionais, segundo nos informa Lopes & Garcia (2002), está intimamente ligado ao conceito de expressão (ou fórmula) matemática, no qual um conjunto de variáveis e constantes numéricas se relaciona por meio de operadores, compondo uma fórmula que, uma vez avaliada, resulta em um valor.

Para Saliba (1993), o conceito de **expressão** aplicado à computação assume uma conotação mais ampla: uma expressão é uma combinação de variáveis, constantes e operadores, que, uma vez avaliada, resulta em um valor.

Aqui constam algumas definições. É importante que você leia com atenção esses conceitos e busque outras fontes de informação sobre isso. Também se faz necessário que você procure o significado das palavras que não conhece.

No conceito de Saliba (1993), descrito acima, por exemplo, você saberia conceituar "operadores"?

"Operadores", ainda segundo esse mesmo autor, são elementos funcionais que atuam sobre operandos e produzem determinado resultado.

Ah! Essa esta é bem simples...
Vejamos o exemplo abaixo:

A expressão **5 – 2** relaciona dois operandos (os números 5 e 2) por meio do operador (-) que representa a operação de subtração.

Os operadores podem ser classificados em binários, unários e ternários. Esta classificação é atribuída de acordo com o número de operandos sobre os quais o operador atua. Neste momento, serão abordados somente os operadores binários e unários, mas é importante destacar que os ternários estão presentes em várias linguagens de programação e que sua forma de uso depende exclusivamente da linguagem que está sendo utilizada. Observe, a seguir, exemplos de operadores binários e unários.

- binários: operações matemáticas simples nas quais o operador envolve dois algarismos.
  1+3
  4-1
- unários: o operador somente se encarrega de informar se o número é positivo ou negativo.
  +2

Outra classificação feita aos operadores é em consideração ao tipo de dado de seus operandos e do valor resultante de sua avaliação. Esta classificação é de operadores aritméticos, lógicos, relacionais e literais. Vejamos de que se trata.

2.3.1. EXPRESSÕES ARITMÉTICAS

São aquelas cujo resultado da avaliação é do tipo numérico, seja ele inteiro seja real. Somente o uso de operadores aritméticos e de variáveis numéricas é permitido em expressões deste tipo.

Operador	Tipo	Operação	Prioridade	Exemplo
-	Unário	Inversão de sinal	1	-(-1) = 1
+	Unário	Manutenção de sinal	1	+1 = 1
**	Binário	Exponenciação	2	9**2 = 81
*	Binário	Multiplicação	3	6 * 7 = 42
/	Binário	Divisão	3	8 / 2 = 4
+	Binário	Adição	4	1 + 2 = 3
-	Binário	Subtração	4	4 – 5 = -1

A prioridade entre operadores define a ordem em que eles devem ser avaliados dentro de uma mesma expressão.

Quando há dois ou mais operadores de mesma prioridade em uma expressão, a execução se dá da esquerda para a direita.

É óbvio que, se utilizarmos o conceito de agrupamento de operações através de parênteses, isso poderia ser facilmente visto.

O caractere ***** é adotado na maioria das linguagens de programação para representar a operação de multiplicação, ao invés do caractere **x**, por força da possibilidade de ocorrência do mesmo nome de variável. Pela mesma razão, o símbolo ****** é adotado para representar a operação de exponenciação. Algumas linguagens de programação adotam o símbolo **^,** e outras adotam **função** para resolução deste problema.

Por exemplo: em **Java,** utiliza-se **Math.pow(3, 2)** para elevar o número 3 ao quadrado.

2.3.2. EXPRESSÕES LÓGICAS

São aquelas cujo resultado da avaliação é um valor lógico (**.Verdadeiro.** ou **.Falso.**). Veja, a seguir, operadores lógicos para expressões lógicas.

Para tratar expressões lógicas, vamos recorrer à lógica matemática, na qual Filho (2000) define **proposição** como todo o conjunto de palavras ou símbolos que exprimem um pensamento de sentido completo.

As proposições transmitem pensamentos, isto é, **afirmam fatos** ou **exprimem juízos** que formamos a respeito de determinados entes.

Podemos citar como exemplo de proposição:

a) A Lua é um satélite da terra.
b) Recife é a capital de Pernambuco.
c) 2^3 é 8.

A lógica matemática (assim como a que veremos para algoritmos) adota como regras fundamentais do pensamento os dois seguintes princípios:

a) Princípio da não contradição
Uma proposição não pode ser verdadeira e falsa ao mesmo tempo.
b) Princípio do terceiro excluído
Toda a proposição ou é verdadeira ou é falsa, isto é, verifica-se sempre um destes casos, e nunca um terceiro.

As proposições vistas no exemplo anterior são ditas **simples** ou ***atômicas***, pois não contêm nenhuma outra proposição como parte integrante de si mesma. Há somente a ocorrência de uma informação sendo repassada.

Chama-se **proposição composta** ou ***proposição molecular*** aquela formada pela combinação de duas ou mais proposições, como nos exemplos:

a) Carlos é careca **e** Pedro é estudante.
b) Carlos é careca **ou** Pedro é estudante.
c) **Se** Carlos é careca, **então** é infeliz.

Quando pensamos, efetuamos, muitas vezes, certas operações sobre proposições, chamadas **operações lógicas**.

a) Negação
Chamamos **negação de uma proposição p** a proposição representada por ***não p***, cujo valor lógico é a **verdade** quando **p** é falsa, e a **falsidade** quando **p** é **verdadeira**. Assim ***não p*** tem o valor lógico oposto daquele de ***p***.
b) Conjunção
Chamamos **conjunção de duas proposições p e q** a proposição representada por ***p E q***, cujo valor lógico é **verdade** quando as proposições **p** e **q** são ambas verdadeiras, e **falsa** nos demais casos.
c) Disjunção
Chama-se **disjunção de duas proposições p e q** a proposição representada por **p OU q** cujo valor lógico é a **verdade** quando ao menos uma das proposições **p** e **q** é verdadeira, e a **falsidade** quando as proposições **p** e **q** são ambas falsas.

Os operadores lógicos também são chamados de operadores booleanos.

Operador	Tipo	Operação	Prioridade
.OU.	Binário	Disjunção	3
.E.	Binário	Conjunção	2
.NÃO.	Unário	Negação	1

Suponha duas perguntas feitas a quatro pessoas que se candidataram a uma entrevista de emprego de programador. As respostas às perguntas serão **Sim** ou **Não**. Suponha também que só será chamado para a entrevista o candidato que responder **Sim** às duas perguntas.

Tabela verdade do operador .E.			
Candidato	Você conhece a linguagem C ?	Você conhece a Linguagem Pascal ?	Candidato aprovado para a entrevista ?
César	**Não**	**Não**	**Não**
Itamar	**Não**	**Sim**	**Não**
Dudu	**Sim**	**Não**	**Não**
Neusa	**Sim**	**Sim**	**Sim**

Neste caso, apenas a candidata **Neusa** seria chamado para a entrevista, pois o operador **.E.** só considera a expressão como verdadeira se todas as expressões testadas forem verdadeiras.

Vejamos, a seguir, **operadores relacionais** para expressões lógicas.

Operador	Comparação
=	Igual
<>	Diferente
<	Menor
=<	Menor ou igual
>	Maior
=>	Maior ou igual

Estes operadores são somente usados quando se deseja efetuar comparações.

Vale destacar que as comparações só podem ser feitas entre objetos de mesma natureza, isto é, variáveis do mesmo tipo de dado. O resultado de uma comparação é sempre um valor lógico.

Toda expressão respeita uma ordem de execução de seus operadores, sempre da esquerda para a direita, assim como na matemática. Para isso, segue uma tabela com as prioridades.

TABELA COM PRIORIDADES DOS OPERADORES	
Prioridade	Pseudocódigo
Parênteses e funções	Parênteses e funções
Potência e resto	Exp( ) e resto( )
Multiplicação e divisão	* e /
Adição e subtração	+ e -
Operadores relacionais	E, OU, NÃO
Operadores lógicos	= ,>< ,=> ,=< ,< ,>

Exemplos do uso de operadores relacionais			
Exemplo	**Valores**	**Questionamento ?**	**Resultado**
A <> B	A = 5 e B = 6	A é diferente de B ?	*Verdadeiro*
A <> B	A = 6 e B = 6	A é diferente de B ?	*Falso*
X == 1	X = 2	X é igual a 1 ?	*Falso*
X == 1	X = 1	X é igual a 1 ?	*Verdadeiro*
7 > 6		7 é maior que 6 ?	*Verdadeiro*
8 < 9		8 é menor que 9 ?	*Verdadeiro*
1 <= Y	Y = 1	1 é menor ou igual a Y ?	*Verdadeiro*
1 <= Y	Y = 2	1 é menor ou igual a Y ?	*Verdadeiro*
1 <= Y	Y = 0	1 é menor ou igual a Y ?	*Falso*
4 >= W	W = 4	4 é maior ou igual a W ?	*Verdadeiro*
4 >= W	W = 3	4 é maior ou igual a W ?	*Verdadeiro*
4 >= W	W = 5	4 é maior ou igual a W ?	*Falso*

2.4. SINTAXE E SEMÂNTICA

O conceito de linguagem está associado a um *objeto de comunicação*: indivíduos que partilham uma mesma linguagem são capazes de se comunicar. As línguas naturais são utilizadas como meio *formal* de se estabelecer uma linguagem de comunicação. Para tanto, são necessários um vocabulário, ou *léxico (dicionário/glossário)*, e um conjunto de regras gramaticais ou *sintaxe*: para construir um objeto de comunicação nessa língua, a sintaxe permite associar e manipular os itens do léxico. A esse objeto, construído segundo as *normas* da língua, dá-se o nome de *asserção válida*, ou *gramatical*. Asserções válidas com diversos graus de complexidade podem ser construídas: em grau crescente de complexidade, podemos ter orações, sentenças, conjunto de sentenças (ou parágrafos) e conjuntos de parágrafos (ou textos). A essas asserções, é possível também associar um *significado* e, assim, estaremos trabalhando no campo da *semântica*.

Bem, de acordo com o que foi exposto acima, podemos, de modo conciso, dizer:

SINTAXE : São regras gramaticais de formação de sentenças/asserções válidas ou gramaticalmente corretas;

SEMÂNTICA : É a associação das asserções ao significado, permitindo sua *interpretação*.

Tendo estas definições sido trazidas do idioma humano, pode-se afirmar que não há diferença ao encontrado na área tecnológica, pois podemos dizer que a **sintaxe** de uma linguagem expressa as regras que devem ser obedecidas para atingir determinado resultado, fazendo uso dela. E a **semântica** representa o conteúdo das palavras da linguagem, permitindo assim uma interpretação correta do escrito com determinada linguagem.

Esta unidade tem grande importância no aprendizado de programação. Foram aqui apresentados conceitos e exemplos de tipos de dados possíveis de uso em nossos pseudocódigos, conceitos e regras para uso de variáveis, além de uma introdução sobre expressões e operações. Na parte de expressões lógicas, trouxemos uma pequena demonstração sobre suas operações, as quais serão fortemente tratadas mais adiante. Já nos conceitos de sintaxe e semântica, que foram tratados também de forma breve, foi comentada a grande importância delas, principalmente da sintaxe. Todos estes conceitos serão utilizados nas próximas unidades. Daí a relevância de uma releitura, caso ainda restem dúvidas.

Encerramos esta unidade por aqui, esperando, é claro, que esses conceitos e exemplos tenham ajudado você a compreender ainda mais as etapas da programação.

Mas ainda temos um longo caminho. Vamos em frente!

UNIDADE III

INTRODUÇÃO AOS PSEUDOCÓDIGOS

Nesta unidade, iremos introduzir você na linguagem de programação hipotética, muito usada em todo o mundo. É chamada de pseudocódigo. Utiliza o idioma nativo e algumas regras sintáticas, mas a facilidade de passar para algoritmos regras verbais, por nós conhecida e usada, é mais simples e facilita depois a introdução a uma linguagem de programação.

Por meio de exemplos você terá contato com a representação de algoritmos recorrendo a pseudocódigos. Abordaremos também os pseudocódigos com estruturas condicionais e faremos breve comentário sobre funções.

Assim, esperamos que, ao final da leitura desta unidade, você possa:

- Conhecer e entender pseudocódigos;
- Compreender a importância da endentação;
- Conhecer estruturas de decisão;
- Resolver problemas com estrutura sequencial e condicional por meio de pseudocódigos;
- Reconhecer e utilizar algumas funções predefinidas para auxílio na resolução dos problemas.

3.1. PSEUDOCÓDIGOS

O que é pseudocódigo?

Para Saliba (1993), ***pseudocódigo*** é uma forma para representação de algoritmos rica em detalhes, como a definição dos tipos das variáveis usadas no algoritmo. Por assemelhar-se bastante à forma em que programas são escritos, encontra muita aceitação.

Para nós, a aplicação de pseudocódigos aos algoritmos, a partir deste ponto, será mera tradução. Como um texto em português passado para outro idioma. Necessitamos saber apenas a ortografia (palavras) e regras gramaticais (sintaxe). Mas, qual a vantagem do uso de pseudocódigos?

3.1.1 VANTAGENS

O pseudocódigo é a escrita, por meio de regra predefinida, dos passos a serem seguidos para a resolução de um problema. É o resultado da análise e resolução de um problema exposto em um enunciado.

A passagem de um algoritmo para uma linguagem de programação tem um trânsito fácil, bastando, para isso, o conhecimento do vocabulário e regras sintáticas da linguagem/idioma desejado.

3.1.2 ESTRUTURA BÁSICA PARA UM PSEUDOCÓDIGO

A ferramenta para representação de algoritmos utilizada até agora (Descrição

Narrativa), possui suas regras, embora sejam bem flexíveis.

No caso do **pseudocódigo**, as regras são mais arbitrárias, pois trazemos de nosso idioma palavras que representarão comportamentos e ações que deverão ser tomadas pelo algoritmo.

A estas palavras, dá-se o nome de **Palavras Reservadas à Linguagem**, ou seja, você NÃO pode utilizar tais palavras para outros fins senão àquele que ela representa. Começaremos agora, pouco a pouco, a conhecer estas palavras. Veja a estrutura básica que um pseudocódigo deve ter e seu comparativo para as linguagens de implementação propostas.

E atenção! As palavras reservadas estão em **negrito** e o texto em *itálico* deverá ser preenchido pelo programador, desconsiderando os sinais < > . Não se assuste, é muito simples. Para cada uma das estruturas existem considerações logo após aquelas.

Estrutura básica para um pseudocódigo
Algoritmo <nome> **Variáveis** *<lista de variáveis>* **Início** *<bloco de variáveis>* **Fim**

Algoritmo

Representa apenas o ínicio de um novo programa, e o *<nome>* deve ser fornecido pelo programador. É interessante que este nome seja condizente com a finalidade do programa.

Variáveis

Informa ao pseudocódigo que as próximas linhas se referem à declaração das variáveis que serão utilizadas no programa. Todas as variáveis que serão utilizadas devem estar relacionadas nesta parte do código, cada uma com seu tipo de dado predeterminado.

Início

Declaração de que, neste ponto, é dado início ao processamento do programa. Tudo que estiver no *<bloco de instruções>* se refere à solução encontrada para resolução do problema.

Fim

Obrigatoriamente deve ser informado o final do programa, assim como foi informado o início.

A seguir, um exemplo clássico encontrado em todos os livros de iniciação à programação, o famoso ***Olá, Mundo!***

Olá, Mundo em pseudocódigo
Algoritmo OlaMundo **Variáveis** **Início**   **Escreva** "Olá Mundo!" **Fim**

3.2. ENDENTAÇÃO

A endentação está diretamente ligada à **formatação de programas,** tendo como finalidades principais:

- melhorar a legibilidade deles;
- facilitar a manutenibilidade posterior;
- facilitar a identificação do erro lógico durante a execução.

Endentação deve ser utilizada para indicar que as instruções endentadas estão **sob controle da instrução anterior não endentada**. O uso ***consistente*** de endentação é essencial para a legibilidade do programa.

Use sempre endentação para instruções dentro de blocos ou para instruções que fazem parte de alguma estrutura de controle ou seguindo um rótulo.

Observe, no pseudocódigo anterior, que a instrução **Escreva** está *endentada* e, visivelmente, nota-se que ela está sob o controle de **Início** e **Fim**.

3.3. DECLARAÇÃO DE VARIÁVEIS

A declaração das variáveis a serem manipuladas no algoritmo devem estar entre as palavras reservadas **Variáveis** e **Início**. Toda variável utilizada deve ter um nome e um tipo. Na unidade anterior, foram tratados os tipos de dados possíveis para uma variável e as regras para atribuir um nome a ela.

Declaração de variáveis em pseudocódigo

Algoritmo variaveis
Variáveis
  a : Inteiro
  b : Real
  c : Caractere
  d : Literal
Início
...
Fim

3.4. ATRIBUIÇÃO DE VARIÁVEIS

A atribuição de variável se dá quando precisamos *armazenar* algum valor para ser usado posteriormente em nosso algoritmo, seja este uso em operações ou na simples informação para o usuário. Quando utilizamos pseudocódigo para representar algoritmos, devemos convencionar um símbolo para representar esta operação. Normalmente utilizamos **:=** (*dois pontos seguido de um igual – sem espaço*), um simples sinal de **=** (*igual*) ou ainda os símbolos gráficos ⇐ ou ← para representar que a parte que está à direita do operador é atribúida à variável a seu lado esquerdo. Como na matemática, sem segredos.

Atribuição de valores à variáveis em pseudocódigo

```
Algoritmo variaveis
Variáveis
  a : Inteiro
Início
  a ← 1
Fim
```

3.5. ENTRADA DE DADOS

Uma operação de entrada de dados se refere a qualquer valor que seja informado (digitado) pelo usuário. Este valor pode ser numérico ou literal, e será armazenado, obrigatoriamente, em uma variável de seu respectivo tipo. Ou seja, um valor literal não pode ser armazenado em uma variável numérica, porém um valor numérico pode ser armazenado em uma variável literal, mas este será tratado como literal. Quando representamos um algoritmo por meio de pseudocódigos, várias palavras de nosso idioma podem ser utilizadas, tais como ***Leia, Entre, Pegue*** *ou* ***Receba***.

Perceba que, como são várias as opções, faz-se necessário que seja estabelecida uma convenção. Aqui será definida como **Leia**.

É interessante saber que o valor informado pelo usuário será guardado (armazenado) na variável imediatamente após a instrução **Leia**. Apesar de algumas literaturas trazerem várias variáveis em uma única instrução **Leia**, separadas apenas por vírgulas (,), é recomendado que cada instrução **Leia** tenha apenas uma variável ligada a ela.

Entrada de dados em pseudocódigo

```
Algoritmo EntradaDeValores
Variáveis
 idade : Inteiro
 salario : Real
 nome : Literal
```

```
Início
  Escreva "Digite seu Nome"
  Leia nome
  Escreva "Digite sua idade"
  Leia idade
  Escreva "Digite seu salário"
  Leia salario
  Escreva "Olá ",nome," você tem ", idade, " e recebe ", salario
Fim
```

3.6. SAÍDA DE DADOS

Uma operação de saída de dados se refere a qualquer valor exibido ou retornado ao usuário. Este valor pode ser uma mensagem de orientação, o resultado de uma expressão, um valor constante ou uma variável (é lógico que será exibido o valor contido na variável). Quando representamos um algoritmo por meio de pseudocódigos, várias palavras de nosso idioma podem ser utilizadas, tais como ***Escreva, Informe, Mostre*** ou ***Exiba***. Novamente, há a ocorrência de várias possibilidades, o que sugere a escolha de um dos termos. No caso desta unidade optamos por ***Escreva.***

É interessante saber que, quando desejarmos exibir ao usuário uma frase, esta deve estar entre aspas (estas aspas, podem ser a simples ou as duplas, mas usaremos as duplas). Se quisermos exibir um valor numérico constante, basta informá-lo após a instrução. Se o que desejarmos exibir ao usuário for o conteúdo de uma variável, basta informarmos tal variável após a instrução **Escreva**.

Saída de dados em pseudocódigo

```
Algoritmo saida
Variáveis
  a : Inteiro
Início
  Escreva "Olá Mundo"
  Escreva 10
  Escreva 11.20
  a ← 30
  Escreva a
  Escreva "a tem ", a
Fim
```

3.7. ALGUMAS SITUAÇÕES

3.7.1. ESTRUTURA SEQUENCIAL

Com os conceitos e as instruções primitivas passadas, podemos iniciar o desenvolvimento e o aprendizado necessário para a resolução e representação

de algoritmos em sua forma de pseudocódigos.

Sobre o salário bruto de um funcionário, são descontados 8% de INSS, 10% de IR (imposto de renda) e sobre o restante 0,5% referente à filiação sindical. Para cada dependente (filhos), o funcionário ganha R$ 50,00. Ao ser fornecido o valor do salário bruto do funcionário e a quantidade de dependentes, calcule :

a) O total dos descontos
b) Total de acréscimo
c) Salário líquido.

```
Algoritmo CalculoDeFolha
Variáveis
    SalarioBruto, INSS, IR, FS, TotalDeDescontos, TotalDeAcrescimos,
            SalarioLiquido : Real
    Dependentes : Inteiro
Início
    Escreva "Informe o Salário Bruto"
    Leia SalarioBruto
    Escreva "Informe a Quantidade de Dependentes"
    Leia Dependentes
    INSS ← SalarioBruto*0.08
    IR ← SalarioBruto * 0.10
    FS ← (SalarioBruto – (INSS+IR)) * 0.005
    TotalDeDescontos ← INSS + IR + FS
    TotalDeAcrescimos ← (Dependentes * 50)
    SalarioLiquido ← SalarioBruto – TotalDeDescontos + TotalDeAcrescimos
    Escreva "O Total de Descontos é : ", TotalDeDescontos
    Escreva "O Total de Acréscimos é : ", TotalDeAcrescimos
    Escreva "O Salário Líquido é : ", SalarioLiquido
Fim
```

O primeiro passo na codificação de um pseudocódigo é a identificação das variáveis que se farão necessárias para a resolução. Como fazer isso? Bom, o começo deve ser com o enunciado. Devemos trabalhar o texto e nele identificar o que deve ser recebido pelo pseudocódigo para que solucione o problema proposto e também aquilo que será informado ao usuário, normalmente a resolução.

Mas isso não é tudo! Podem surgir alguns problemas durante a resolução, e aí emerge a necessidade de outras variáveis. Atenção a esta próxima etapa.

Observe, no quadro a seguir, o que será fornecido ao algoritmo para que processe o desejado.

Sobre o **salário bruto** de um funcionário, são descontados 8% de INSS, 10% de IR (imposto de renda) e sobre o restante 0,5% referente à filiação sindical. Para cada **dependente** (filhos) o funcionário ganha R$ 50,00. **Ao ser fornecido o valor do salário bruto do funcionário e a quantidade de dependentes**, calcule:

...

```
Algoritmo CalculoDeFolha
Variáveis
        SalarioBruto, INSS, IR, FS, TotalDeDescontos, TotalDeAcrescimos,
                SalarioLiquido : Real
        Dependentes : Inteiro
Início
        Escreva "Informe o Salário Bruto"
        Leia SalarioBruto
        Escreva "Informe a Quantidade de Dependentes"
        Leia Dependentes
        ...
Fim
```

Veja a ligação entre as declarações destas variáveis e seu real uso dentro do pseudocódigo. Realmente são dados de entrada.

Agora, é identificada na figura a declaração do que é informado ao usuário, ou seja, aquilo que deve ser resolvido pelo algoritmo: o problema.

..., calcule :

a) O Total dos Descontos
b) Total de Acréscimo
c) Salário Líquido

```
Algoritmo CalculoDeFolha
Variáveis
        SalarioBruto, INSS, IR, FS, TotalDeDescontos, TotalDeAcrescimos,
                SalarioLiquido : Real
        Dependentes : Inteiro
Início
        ...
        Escreva "O Total de Descontos é : ", TotalDeDescontos
        Escreva "O Total de Acréscimos é : ", TotalDeAcrescimos
        Escreva "O Salário Líquido é : ", SalarioLiquido
Fim
```

Note agora a existência de variáveis que não são nem de entrada, nem de saída. Podemos caracterizá-las como **variáveis auxiliares**, pois elas realmente auxiliam a resolução do problema.

```
...
Algoritmo CalculoDeFolha
Variáveis
        SalarioBruto, INSS, IR, FS, TotalDeDescontos, TotalDeAcrescimos,
                SalarioLiquido : Real
        Dependentes : Inteiro
Início
        ...
        INSS ← SalarioBruto*0.08
        IR ← SalarioBruto * 0.10
        FS ← (SalarioBruto – (INSS+IR)) * 0.005
        TotalDeDescontos ← INSS + IR + FS
        TotalDeAcrescimos ← (Dependentes * 50)
        SalarioLiquido ← SalarioBruto – TotalDeDescontos + TotalDeAcrescimos
        ...
        Fim
```

No caso específico deste problema, estas variáveis são desnecessárias, pois o resultado pode ser obtido diretamente, mediante operações matemáticas. Vejamos como poderia ficar esta resolução.

```
...
Algoritmo CalculoDeFolha
Variáveis
        SalarioBruto, TotalDeDescontos, TotalDeAcrescimos, SalarioLiquido : Real
        Dependentes : Inteiro
Início
        ...
        TotalDeDescontos ← (SalarioBruto * 0.08) + (SalarioBruto * 0.10)
        TotalDeDescontos ← (SalarioBruto – TotalDeDescontos) * 0.005
        ...
Fim
```

Observe que, para a implementação, optou-se por duas linhas para o cálculo do desconto, pois a *Filiação Sindical* é sobre o *Salário Bruto* já descontado o *INSS* e *IR*. É óbvio que, matematicamente, daria para incluir tudo isso em uma única linha, porém poderia ficar "poluída", dificultando a compreensão da referida linha de operação.

Outra situação seria a resolução, inclusive sem o uso das variáveis de saída, porém, no início do aprendizado, este uso **excessivo** de variáveis é benéfico, mas com o decorrer deve ser evitado, otimizando o desempenho e os recursos utilizados.

```
...
Algoritmo CalculoDeFolha
Variáveis
        SalarioBruto : Real
        Dependentes : Inteiro
Início
        ...
        Escreva "O Total de Descontos é : ", (SalarioBruto * 0.08) + (SalarioBruto *
                0.10) + ((SalarioBruto – ((SalarioBruto * 0.08) + (SalarioBruto * 0.10))) *
                0.05
        Escreva "O Total de Acréscimos é : ", (Dependentes * 50)
        Escreva "O Salário Líquido é : ", SalarioBruto – ((SalarioBruto * 0.08) +
                (SalarioBruto * 0.10) + ((SalarioBruto – ((SalarioBruto * 0.08) +
                (SalarioBruto * 0.10))) * 0.05) + (Dependentes * 50)
Fim
```

Note a expressão acima. Há casos em que isso não é recomendado, tampouco deverá ser feito, pois, em uma necessidade de alteração, dificultará a compreensão.

Desenvolva um programa que receba o salário-base de um funcionário, calcule e mostre o salário a receber, sabendo-se que esse funcionário tem gratificação de 5% sobre o salário-base e paga imposto de 7% sobre o salário-base.

```
Algoritmo CalculoDeFolha
Variáveis
        SalarioBase, SalarioLiquido  : Real
Início
        Escreva "Informe o Salário Base"
        Leia SalarioBase
        SalarioLiquido ← SalarioBase + (SalarioBase *  0.05) –
                (SalarioBase * 0.07)
        Escreva "O Salário Líquido é : ",SalarioLiquido
Fim
```

O problema a ser solucionado neste enunciado é relativamente simples. Será informado pelo usuário um valor qualquer (**SalarioBase**) e sobre este valor deverão ser aplicados dois percentuais: um como crédito (**gratificação**) ao valor informado, e outro como débito (**imposto**). Ou seja, o primeiro valor aumentará o valor inicial, e o segundo diminuirá. Os valores a serem aplicados estão explícitos no enunciado, devendo apenas ser aplicados em uma fórmula/expressão.

Perceba que, neste pseudocódigo, foram utilizadas apenas as variáveis necessárias: a de entrada (**SalarioBase**) e a de saída (**SalarioLiquido**). Poderia optar-se por um desenvolvimento mais detalhado e fazer uso de variáveis

auxiliares, mas não seria nada otimizado e muito menos útil neste caso. Caso se opte por este uso, a resolução seria à semelhança do mostrado a seguir.

```
...
Algoritmo CalculoDeFolha
Variáveis
        SalarioBase, SalarioLiquido, Credito, Debito : Real
Início
        ...
        Credito ← (SalarioBase * 0.05)
        Debito ← (SalarioBase * 0.07)
        SalarioLiquido ← SalarioBase + Credito – Debito
        ...
Fim
```

Observe que, como foram informados os valores em percentual a serem descontados, estes puderam ser aplicados diretamente na fórmula, sem necessidade de aplicar a regra de três.

3.8. PSEUDOCÓDIGOS – ESTRUTURA CONDICIONAL

3.8.1. ESTRUTURA CONDICIONAL

Se ... Então
Senão Se ... Então
Senão ...
Fim se

Leia dois valores numéricos e inteiros, e efetue a soma destes valores. Caso o valor encontrado na soma

- seja maior ou igual a 10, este valor deverá ser somado de 5;
- não seja maior ou igual a 10, este valor deverá ser subtraído de 5.
- exiba o valor final.

```
Algoritmo Soma
Variáveis
        A, B, Soma : Inteiro
Início
        Escreva "Informe o primeiro valor"
        Leia A
        Escreva "Informe o segundo valor"
        Leia B
        Soma ← (A + B)
        Se (Soma >= 10) então
```

```
            Soma ← Soma + 5
        Senão
            Soma ← Soma - 5
        FimSe
        Escreva "O resultado de toda a operação é ", Soma
Fim
```

O problema apresentado neste enunciado é identificar qual operação deverá ser feita sobre determinado resultado, obtido mediante a soma de dois valores, os quais serão fornecidos pelo usuário. Note que a operação que deverá ser feita não é de grande importância, mas a identificação de quando deverá ser feita, sim.

Neste exemplo, o uso de uma variável auxiliar é bem visto (**Soma**), pois é por meio do que ela recebe que o problema poderá ser resolvido, sendo feita uma outra operação, a depender do resultado da expressão condicional. Tendo realizado a avaliação da soma dos dois valores, a operação pertinente, mostra-se então o resultado. Note que o resultado é mostrado apenas após todas as possibilidades (apenas duas) terem sido avaliadas, e não a cada avaliação.

Sendo dados três números inteiros distintos (assume-se que o serão), faça um algoritmo que escreva o maior número digitado.

```
Algoritmo MaiorNumero
Variáveis
        A, B, C : Inteiro
Início
        Escreva "Informe o primeiro número"
        Leia A
        Escreva "Informe o segundo número"
        Leia B
        Escreva "Informe o terceiro número"
        Leia C
        Se (A > B) então
            Se (A > C) então
                Escreva "O maior número é ", A
            Senão
                Escreva "O maior número é ", C
            FimSe
        Senão
            Se (B > C) então
```

```
                Escreva "O maior número é ", B
            Senão
                Escreva "O maior número é ", C
            FimSe
        FimSe
Fim
```

Sempre que temos um problema em que a solução dependa de comparação ou avaliação, quer sejam fornecidas diretamente pelo usuário, quer seja ela o resultado de uma operação, encontramos uma solução condicional, que não sabemos, pois a sequência de comandos depende de valores variáveis.

Como pode ser constatado no enunciado acima, serão fornecidos três números, que não se sabe quais são. A cada vez, podem ser números diferentes e talvez, em uma destas execuções, o primeiro número fornecido possa ser o maior, mas em outra não. Em outra situação, o terceiro número poderá ser o maior. A solução deve trabalhar todas as hipóteses.

Execução	1° Número	2° Número	3° Número
1ª	1	2	3
2ª	4	6	5
3ª	14	13	15
4ª	17	18	16
5ª	12	10	11
6ª	9	8	7

Registrando que os valores digitados seguem a tabela acima, podemos identificar em que momento o 1° valor é maior dentre os três. Veja abaixo a sinalização.

Execução	1° Número	2° Número	3° Número
...			
5ª	12	10	11
6ª	9	8	7

Contudo, nós não teremos ciência destes valores, apenas dos locais em que estes valores estarão armazenados, ou seja, as variáveis. Sendo assim, vamos trabalhar o problema com as colunas e as chamaremos de **A** (*1° valor*), **B** (*2° valor*) e **C** (*3° valor*). Temos então as situações seguintes que devem ser avaliadas a cada execução.

1. Quando o valor em **A** for maior que **B**, garante que também é maior que **C**? Lógico que não, pois ainda não nos cabe uma dedução. Porém, se **C** for maior que **A,** está claro que **C** é maior que **B** também. Então **A** é o maior número.

2. Caso ocorra de **A** ser menor que **B**, existe garantia de que **B** é maior que **C**? Também não. Porém, se **C** for maior que **B**, seguramente **C** é o maior número, senão **B** o será.

Veja este texto no quadro a seguir :

```
...
                ...
                Se (A > B) então
                Se (A > C) então
                    Escreva "O maior número é ", A
                Senão
                    Escreva "O maior número é ", C
                FimSe
            Senão
                Se (B > C) então
                    Escreva "O maior número é ", B
                Senão
                    Escreva "O maior número é ", C
                FimSe
            FimSe
Fim
```

Outra situação seria na sintaxe das instruções condicionais. Caso uma condição avaliada não seja a verdadeira, e a contradição seja interessante, utilizamos o **senão**, mas há casos em que várias contradições podem ocorrer e, além disso, mais de uma condição pode ser avaliada ao mesmo tempo. Veja a figura.

```
...
                ...
                Se (A > B) E (A > C) então
                Escreva "O maior número é ", A
            Senão Se (A < B) E (B > C) então
                Escreva "O maior número é ", B
            Senão
                Escreva "O maior número é ", C
            FimSe
Fim
```

Observe agora que a expressão que é operada por **E**, só será verdadeira se todas retornarem verdadeiro.

3.8.2. ESTRUTURA CONDICIONAL COM ESCOLHA ... CASO

Nos exemplos anteriores sobre estrutura condicional, tratou-se da estrutura **Se...Senão Se...FimSe**. Porém, esta não é a única. Existe outra estrutura que veremos a seguir e que trabalha com múltiplas escolhas.

Importante!
Esta estrutura não foi tratada ainda, desta forma, muita atenção às explicações que seguem sobre sua estrutura básica, assim como sobre o detalhamento de suas especificidades.

```
Escolha (expressão)
        Caso <rótulo 1>:        <Bloco de comandos>
        Caso <rótulo 2>:        <Bloco de comandos>
        Caso <rótulo 3>:        <Bloco de comandos>
        Senão
                        <Bloco de comandos>
FimEscolha
```

Algumas considerações para esta nova estrutura devem ser vistas:

- A expressão é avaliada, e o valor será comparado com um dos rótulos.
- A opção senão é opcional.
- O rótulo será aqui definido como uma constante caractere, ou uma constante numérica inteira, embora em algumas linguagens possam ser usadas constantes literais.
- A estrutura é muito usada em algoritmos com menus, tornando-os mais claros do que quando usamos a condição **Se** aninhada.

Para ajudar um pouco, quero relembrar com vocês esta conceituação de Aninhamento: é o termo que se refere a estruturas dentro de outras, por exemplo, uma estrutura **Se** completa dentro de uma outra estrutura **Se**. Aninhamento não é privativo a estruturas condicionais, podem ocorrer e ocorrem também em estruturas de repetição.

Escrever um algoritmo que leia um peso na terra e o número de um planeta, e exiba o valor de seu peso neste planeta. A relação de planetas é dada a seguir juntamente com o valor das gravidades relativas à Terra:

Código	Gravidade relativa	Planeta
1	0,37	Mercúrio
2	0,88	Vênus
3	0,38	Marte
4	2,64	Júpiter
5	1,15	Saturno
6	1,17	Urano

Para calcular o peso no planeta utilize a fórmula:

$$\text{PesoNoPlaneta} = \frac{\text{PesoNaTerra}}{10} * \text{Gravidade}$$

```
Algoritmo PesoPlaneta
Variáveis
        Opcao : Inteiro
```

```
        Peso : Real
Início
        Escreva  "Planetas que podem ser analisados"
        Escreva  "1-Mercúrio"
        Escreva  "2-Vênus"
        Escreva  "3-Marte"
        Escreva  "4-Júpiter"
        Escreva  "5-Saturno"
        Escreva  "6-Urano"
        Leia Opcao
        Escreva  "Informe seu peso atual"
        Leia Peso
        Escolha (Opcao)
            Caso 1 :
                Escreva "Seu peso no planeta Mercúrio é : ", (Peso/10)*0.37
            Caso 2 :
                Escreva "Seu peso no planeta Vênus é : ", (Peso/10)*0.88
            Caso 3 :
                Escreva "Seu peso no planeta Marte é : ", (Peso/10)*0.38
            Caso 4 :
                Escreva "Seu peso no planeta Júpiter é : ", (Peso/10)*2.64
            Caso 5 :
                Escreva "Seu peso no planeta Saturno é : ", (Peso/10)*1.15
            Caso 6 :
                Escreva "Seu peso no planeta Urano é : ", (Peso/10)*1.17
            Senão
                Escreva "A opção ", Opcao, " não pode ser avaliada"
        FimEscolha
Fim
```

Será exibido ao usuário um ***menu*** com as opções que ele pode escolher e que serão processadas pelo pseudocódigo. Neste caso, as opções se referem a nomes de planetas associados a um número, como se fosse um cardápio de um restaurante, pois, na realidade, a ideia de um menu é realmente esta. Veja no quadro a seguir esta situação.

```
...
...
        Escreva  "Planetas que podem ser analisados"
        Escreva  "1-Mercúrio"
        Escreva  "2-Vênus"
        Escreva  "3-Marte"
        Escreva  "4-Júpiter"
        Escreva  "5-Saturno"
        Escreva  "6-Urano"
        Leia Opcao
...
```

É preciso agora, após a escolha do usuário, verificarmos qual foi esta escolha. Poderíamos, sem problema algum, fazer uso da estrutura **Se**, mas para este tipo de situação a estrutura **Escolha** é a mais indicada. Quando fazemos uso desta estrutura, temos que informar a qual variável serão feitos os testes e, a cada **caso**, informarmos o valor que está sendo comparado (que deve existir na variável da **Escolha)**. Veja o quadro a seguir.

```
...
...
        Escolha (Opcao)
            Caso 1 :
                Escreva "Seu peso no planeta Mercúrio é : ", (Peso/10)*0.37
            Caso 2 :
                Escreva "Seu peso no planeta Vênus é : ", (Peso/10)*0.88
            Caso 3 :
                Escreva "Seu peso no planeta Marte é : ", (Peso/10)*0.38
            Caso 4 :
                Escreva "Seu peso no planeta Júpiter é : ", (Peso/10)*2.64
            Caso 5 :
                Escreva "Seu peso no planeta Saturno é : ", (Peso/10)*1.15
            Caso 6 :
                Escreva "Seu peso no planeta Urano é : ", (Peso/10)*1.17
            Senão
                Escreva "A opção ", opcao, " não pode ser avaliada"
        FimEscolha
...
```

Observe que, como não foi feita nenhuma validação aos valores digitados no **menu**, precisamos prever que o usuário poderá digitar um valor que não faz parte das opções disponíveis. Sendo assim, temos que informar a ele esta situação. Observe também que o resultado da fórmula interessará apenas para ser informado ao usuário. Nessa situação, a expressão é avaliada diretamente na instrução **Escreva**. Vejamos algumas situações de execução:

1ª	3	95	Seu peso em Marte é 3,61
2ª	1	63	Seu peso em Mercúrio é 2,33
3ª	7	80	A opção 7 não pode ser avaliada

Nesta unidade, apresentamos breve introdução sobre os pseudocódigos, que representam uma linguagem de programação hipotética.

Os algoritmos apresentados nesta unidade se limitaram à estrutura sequencial, sendo trabalhados, em seguida, os

algoritmos condicionais. A lógica foi toda trabalhada nas unidades anteriores. Estamos agora fazendo uso de uma linguagem e não mais narração.

Nesta fase também foram abordados problemas que estão mais próximos da realidade, em sua forma de resolução, pois trabalhamos condições e estamos sujeitos a elas diariamente. Foram também exibidas algumas opções e regras para o desenvolvimento de pseudocódigos com estruturas condicionais.

Estamos avançando significativamente dentro das técnicas e conhecimentos necessários para a programação. Fique atento e, caso seja necessário, refaça as leituras, pesquise em outras fontes, busque sempre mais! Não fique satisfeito somente com as informações aqui registradas. É sempre necessário questionar, testar, fazer, confrontar e refazer, para então construir suas convicções. Isto é aprender!

Parabéns por chegar até aqui... E vamos em frente!

UNIDADE IV

ESTRUTURAS DE REPETIÇÃO COM PSEUDOCÓDIGOS

Nesta unidade continuaremos a abordagem dos pseudocódigos, agora tratando com maior intensidade as especificidades dos pseudocódigos com estruturas de repetição.

A estrutura de repetição é caracterizada pela necessidade de um mesmo conjunto de instruções ocorrer - serem elas executadas - mais de uma vez, da mesma forma que a primeira. O uso destas estruturas possibilita a reutilização de códigos. Ao invés de, por exemplo, solicitar dez vezes para um usuário informar alguma coisa, pede-se uma vez e esta solicitação é inserida em uma estrutura de repetição.

Esperamos, assim, que ao final desta unidade você seja capaz de

- Conhecer as estruturas de repetição existentes;
- Identificar a correta estrutura de repetição para um problema a ser resolvido.

4.1. ESTRUTURAS DE REPETIÇÃO

Uma estrutura de repetição permite ao programador especificar que uma ação será repetida enquanto alguma condição permanecer verdadeira.

A estrutura de repetição também garante que um processo repetitivo será feito sempre da mesma forma, não importando se ele ocorrerá uma, dez, cem ou mil vezes. Existe uma subdivisão para a estrutura de repetição:

- Contada;
- Condicional, com teste no início;
- Condicional, com teste no final.

Vejamos, resumidamente, cada uma dessas estruturas.

4.2. ESTRUTURAS DE REPETIÇÃO CONTADA

Como visto nas unidades anteriores sobre representação de algoritmos, podemos conceituar uma estrutura com repetição contada como aquela em que sabemos, antes de ela ocorrer, quantas vezes deverá ocorrer. Deve ficar claro, entretanto, que ***saber quantas vezes ocorrerá*** não significa saber a quantidade exata, mas que este valor pode estar atribuído a uma variável, informada ou não pelo usuário.

Sendo dado um número, faça um algoritmo que escreva todos os números ímpares menores que esse número. Dica: Comece pelo um.

```
Algoritmo NumerosImpares
Variáveis
        Numero, I : Inteiro
Início
        Escreva "Informe um número"
```

```
        Leia Numero
        Para I = 1 Até Numero Faça
            Se (Resto(I, 2) <> 0) Então
                Escreva I, " é ímpar"
            FimSe
        FimPara
Fim
```

O problema proposto neste enunciado é que deverá ser solicitado ao usuário um número, o algoritmo deverá fazer uma contagem de 1 até o valor informado, e, a cada número contado, identificar se o mesmo é ímpar. Se for, deve ser mostrado seu valor ao usuário.

Agora muita atenção! O texto anterior ao exercício dizia que uma repetição é caracterizada como contada quando *sabemos* quantas vezes ela vai ocorrer, e você pode agora dizer: "*Mas eu não sei, é o usuário que irá informar.*". É verdade, nós realmente não sabemos o valor, mas sabemos onde ele será guardado quando o usuário o informar e, sabendo isso, conseguimos saber o valor. Neste caso, a variável ***Número***.

Nós sabemos contar, porém como o algoritmo pode controlar isso? Primeiramente, ele precisa de uma estrutura que caracterize, imponha e garanta a repetição. Esta estrutura é a **PARA...FAÇA...FIMPARA**. Ela tem esta finalidade.

Supondo que o valor informado na execução do algoritmo tenha sido **50**, a leitura para a instrução seria : ***Para cada ocorrência entre 1 e 50, faça ..... fimpara***.

```
...
...
        ...
        Para I = 1 Até Numero Faça
            Se (Resto(I, 2) <> 0) Então
                Escreva I, " é ímpar"
            FimSe
        FimPara
Fim
```

PARA REFLETIR

Tudo o que estiver entre o **faça** e o **fimpara** será caracterizado como o que deve ser feito para cada ocorrência, neste caso verificar se a variável **I** é **ímpar** e, em caso positivo, informá-la ao usuário. Mas o que é e o que faz esta variável **I** ?

Muita calma! Vamos passo a passo...

Como dito em um passo anterior, é preciso prover recursos para que o algoritmo comece a contar, e isso deve ser feito usando uma variável que tem seu valor inicial, como o primeiro informado na instrução **PARA** e a cada vez que se alcança o **FIMPARA**, ela automaticamente é incrementada em um. Após isso, retorna ao **PARA** e é verificado se não ultrapassou o máximo permitido (o segundo valor informado). Caso tenha ultrapassado, nada entre o **PARA** e o **FIMPARA** é executado, saltando o controle do algoritmo para a primeira linha após o **FIMPARA**.

Teste de mesa para a digitação do número 5				
Número	I	(I > Numero)	(Resto(I, 2) <> 0)	Saída
5	1	Não	Sim	1
5	2	Não	Não	
5	3	Não	Sim	3
5	4	Não	Não	
5	5	Não	Sim	5
5	6	Sim		

Faça um algoritmo que leia 10 números e obtenha:

a) A soma dos números;
b) A média dos números;
c) O maior número;
d) O menor número.

```
Algoritmo MenuNumeros
Variáveis
        Numero, Soma, Maior, Menor, I : Inteiro
        Media : Real
Início
        Soma = 0
        Para I = 1 Até 10 Faça
                Escreva "Informe um número ", I, " de 10."
                Leia Numero
                Soma = Soma + Numero
                Se (I=1) ou (Numero > Maior) Então
                   Maior = Numero
                FimSe
                Se (I=1) ou (Numero < Menor) Então
                   Menor = Numero
                FimSe
        FimPara
        Media = (Soma / 10)
        Escreva "A soma dos 10 números digitados é : ", Soma
        Escreva "A média aritmética entre os 10 números digitados é : ", Media
        Escreva "O maior número dentre os 10 digitados é ", Maior
        Escreva "O menor número dentre os 10 digitados é : ", Menor
Fim
```

O que se repete neste algoritmo? A solicitação de 10 números, porém, a cada número digitado, devem ser efetuados alguns passos com objetivos de atender às quatro solicitações expressas no enunciado.

A ***primeira solicitação*** diz respeito à ***soma de todos os números***. Para isso se faz necessária uma variável que guarde os valores acumulados a cada número informado. Note que esta variável se chama ***SOMA***, mas poderia ter qualquer nome. A principal informação a ser feita é que ela recebe o valor 0 (zero) antes da estrutura de repetição começar.

Este procedimento se faz necessário, pois como poderia ter um total se, a cada execução do laço, zerássemos a ***variável acumuladora***? E por que o 0 (zero) ? Ele não afetará em nada o resultado final.

```
...
...
        Soma = 0
        Para I = 1 Até 10 Faça
            ...
            Soma = Soma + Numero
            ...
        FimPara
        ...
Fim
```

A ***segunda solicitação*** pede a média dos números informados. Como o texto não diz a que tipo de média se refere, assumimos a aritmética, e a fórmula diz que a obtemos através da soma de N valores que devem ser divididos por N. Bem, a soma nós já temos, pois foi obtida para a resolução do primeiro item. Basta então dividirmos pelo total de números informados, neste caso explicitamente o 10. Note que esta operação é feita após ter a soma garantida, ou seja, após a entrada de todos os números (após a repetição).

```
...
        ...
        Para I = 1 Até 10 Faça
            ...
            Soma = Soma + Numero
            ...
        FimPara
        Media = (Soma / 10)
        ...
Fim
```

A ***terceira solicitação*** quer o maior número de todos os 10 informados. Como fazer isso? Se fossemos fazer esta verificação em uma folha com 10 números, seria fácil, pois compararíamos os números entre si e então identificaríamos o maior. Mas como fazermos isso, quando os números nos serão informados um a um? É mais fácil, você verá.

Se temos um único número, existe, nestes valores que temos (um único número), algum maior que ele? Não, pois ele é o único. Sendo assim, podemos afirmar que, em qualquer situação de comparação, o primeiro valor informado ou obtido será sempre o maior, o menor, o mais baixo, o mais magro, o mais gordo, ...

```
...
        ...
        Para I = 1 Até 10 Faça
            ...
            Se (I = 1) ou (Numero > Maior) Então
               Maior = Numero
            FimSe
            ...
        FimPara
        ...
Fim
```

É isso que fizemos ao comparar nossa variável de controle a **1** (um), como pode ser visto na figura acima. Mas existem outros valores, como fazer? Bem, na primeira vez em que a estrutura ocorre, existe a garantia da variável maior receber um valor. Sendo assim, é só comparar o valor atual com aquele que temos como maior. Veja a figura anterior. O operador **OU** garante que a condição será satisfeita em uma ou outra situação.

É interessante ressaltar que esta é uma das muitas maneiras de resolver este problema.

Quanto à ***quarta solicitação***, podemos dizer que a regra é a mesma para a resolução da terceira opção.

Veja abaixo o teste de mesa para a resolução apresentada.

Teste de mesa para os 10 números solicitados								
I	Número	Soma	(I=1)	(Número>Maior)	Maior	(Número<Menor)	Menor	Média
		0						
1	1	1	Sim	*Nem entra aqui*	1	*Nem entra aqui*	1	
2	100	101	Não	Sim	100	Não		
3	2	103	Não	Não		Não		
4	50	153	Não	Não		Não		
5	-2	151	Não	Não		Sim	-2	
6	200	351	Não	Sim	200	Não		
7	4	355	Não	Não		Não		
8	25	380	Não	Não		Não		
9	-4	376	Não	Não		Sim	-4	
10	400	776	Não	Sim	400	Não		
								77,6

4.3. ESTRUTURAS DE REPETIÇÃO CONDICIONAL, COM TESTE NO INÍCIO

Da mesma forma que visto nos textos anteriores que trataram as formas de representação de um algoritmo, a repetição condicional em pseudocódigos ocorrerá de duas formas: o teste condicional (pergunta) feito antes (início) de a estrutura de comandos ocorrer ou ao final (após) da estrutura de comandos.

Calcule o imposto de renda de um grupo de contribuintes, considerando que os dados de cada contribuinte são: número de CPF, número de dependentes e renda mensal. Para cada contribuinte será feito um desconto de 5% de salário mínimo por dependente. Os valores da alíquota para cálculo do imposto são:

- Até 2 salários Mínimos : Isento;
- 2,1 até 3 salários Mínimos : 10%;
- 3,1 até 5 salários Mínimos : 15%;
- 5,1 até 7 salários Mínimos : 20%;
- Acima de 7,1 salários Mínimos: 25%.

O último valor que não será considerado terá o CPF igual a zero. Deve ser fornecido o valor atual do salário mínimo.

```
Algoritmo DescontoIR
Variáveis
        CPF, QtdeDependentes, Aliquota : Inteiro
        Salario, SalarioMinimo, SalarioBase, ValorIR,
                QtdeSalarioMinimo : Real
Início
        Escreva "Informe o Salário Mínimo"
        Leia SalarioMinimo
        Escreva "Informe o CPF"
        Leia CPF
        Enquanto (CPF <> 0) Faça
                Escreva "Informe o salário do funcionário"
                Leia Salario
                Escreva "Informe a quantidade de dependentes"
                Leia QtdeDependentes
                SalarioBase = (Salário – (QtdeDependentes*(SalarioMinimo*0.05))
                QtdeSalarioMinimo = (SalarioBase / SalarioMinimo)
                Se (QtdeSalarioMinimo <= 2) Então
                   Aliquota = 0
                Senão Se (QtdeSalarioMinimo <= 3) Então
                   Aliquota = 10
                Senão Se (QtdeSalarioMinimo <= 5) Então
                   Aliquota = 15
                Senão Se (QtdeSalarioMinimo <= 7) Então
                   Aliquota = 20
```

```
                Senão
                   Aliquota = 25
                FimSe
                ValorIR = (SalarioBase * Aliquota) / 100
                Escreva "O desconto do IR para o CPF ", CPF, " é de ", ValorIR
                Escreva "Informe o CPF"
                Leia CPF
        FimEnquanto
Fim
```

Este exercício traz uma situação em que não sabemos quantas vezes, num determinado conjunto de instruções, estas devem ser executadas, porém sabemos quando elas deixam de se repetir.

O primeiro passo é descobrir qual conjunto de instruções deve ser repetido no algoritmo. Identificamos a solicitação do CPF (inclusive devido ao fato de ele ser nosso *flag*). Para cada CPF deve ser solicitado também a quantidade de dependentes que o contribuinte tem e seu salário-base. É importante lembrar também a necessidade do salário mínimo, porém este não se repete, pois tem o mesmo valor para todo o algoritmo. Outro ponto a ser destacado é que, a cada grupo de informações recebidas, a identificação da alíquota de imposto em que se enquadra cada contribuinte deve ser feita.

```
...
Algoritmo DescontoIR
                ...
Início
        Escreva "Informe o Salário Mínimo"
        Leia SalarioMinimo
        Escreva "Informe o CPF"
        Leia CPF
        Enquanto (CPF <> 0) Faça
                ...
        FimEnquanto
Fim
```

Note acima que o Salário Mínimo está sendo informado antes da estrutura de repetição (***Enquanto***). Isso garante que o valor informado pode ser válido para toda a estrutura, mas não quer dizer que não possa ser solicitado dentro dela, como é o caso do CPF, que é solicitado também antes de a estrutura começar, porém é solicitado novamente dentro da estrutura, pois ele é nosso ***flag***.

FLAG é uma condição predefinida que determina o fim de uma estrutura de repetição, seja ela contada ou não contada. Na contada, o *FLAG* é implícito na quantidade de vezes que a estrutura se repete. Na não contada, ela depende de uma interação do usuário ou um comportamento específico do algoritmo.

Com a estrutura exibida acima, pode o usuário digitar o CPF 0 (zero) antes mesmo de a estrutura começar? Sim, pode. Isso faria com que a estrutura de repetição não ocorresse nenhuma vez. Não há problemas nisso. Imagine a situação em que quer fazer algo, mas, na hora de começar, algo ocorre e então você opta por não fazer.

Vamos explicar melhor, por meio de uma analogia bem simples:

Imagine que você tenha recebido várias folhas, todas com várias linhas, em que cada linha represente informações referentes a um CPF. Esta linha tem o número do CPF, a quantidade de dependentes e salário básico para o contribuinte do CPF da linha.

São muitas linhas, muitas folhas... Aff... Você não contará quantos são para poder usar uma estrutura de repetição contada! Mas, também não pode garantir que informará todas as linhas de uma única vez e, quando 'cansar' ou tiver que parar, ao invés de informar um CPF válido, informará o valor 0 (zero), e quando quiser, volta e recomeça. Simples, não é?

```
...
Algoritmo DescontoIR
Variáveis
        ...
Início
        ...
        Enquanto (CPF <> 0) Faça
                Escreva "Informe o salário do funcionário"
                Leia Salario
                Escreva "Informe a quantidade de dependentes"
                Leia QtdeDependentes
                SalarioBase = (Salário – (QtdeDependentes*(SalarioMinimo*0.05))
                QtdeSalarioMinimo = (SalarioBase / SalarioMinimo)
                ...
        FimEnquanto
Fim
```

Tão logo a estrutura de repetição ocorra (o CPF é diferente de zero), deve ser solicitado o salário do contribuinte e quantos dependentes ele tem. Um esclarecimento aqui deve ser dado em relação ao salário-base para cálculo de imposto de renda. Do salário utilizado para cálculo do imposto já são deduzidos os descontos, em nosso caso em relação aos dependentes que ele tem. Para isso, fazemos a operação de produto da quantidade de dependentes com o valor a que tem direito por salário mínimo. Após este cálculo, temos o salário-base para cálculo do imposto.

A identificação da faixa em que o contribuinte se enquadra é feita em relação à quantidade de salários mínimos que apresenta para cálculo. Sendo assim, fazemos uma divisão do salário identificado como base, pelo valor do salário mínimo.

```
...
Algoritmo DescontoIR
Variáveis
        ...
Início
        ...
        Enquanto (CPF <> 0) Faça
        ...
        Se (QtdeSalarioMinimo <= 2) Então
                Alíquota = 0
            Senão Se (QtdeSalarioMinimo <= 3)) Então
                Alíquota = 10
            Senão Se (QtdeSalarioMinimo <= 5)) Então
                Alíquota = 15
            Senão Se (QtdeSalarioMinimo <= 7)) Então
                Alíquota = 20
            Senão
                Alíquota = 25
            FimSe
            ValorIR = (SalarioBase * Alíquota) / 100
            Escreva "O desconto do IR para o CPF ", CPF, " é de ", ValorIR
            ...
            FimEnquanto
Fim
```

Uma vez obtida a quantidade de salários mínimos que o contribuinte recebe, fazemos a verificação da faixa em que ele se encontra, para então identificar a alíquota que será incidida em seu salário. Note que a preocupação na estrutura condicional (**Se...Senão Se...Senão...FimSe**) é identificar e atribuir a uma variável a alíquota, e não fazer o cálculo, pois o cálculo só será feito após esta identificação, e a cada contribuinte apenas uma única alíquota será encontrada, não justificando fazer o processo a cada verificação de alíquota.

Uma vez identificada a alíquota (ao final da estrutura condicional), efetuamos o processo de cálculo do valor do imposto de renda para o contribuinte e consequente exibição do valor encontrado.

```
...
Algoritmo DescontoIR
Variáveis
        ...
Início
        ...
        Enquanto (CPF <> 0) Faça
        ...
        Escreva "Informe o CPF"
            Leia CPF
        FimEnquanto
Fim
```

Observe que, dentro da estrutura de repetição, é solicitado novamente, ao usuário, o CPF do contribuinte. Por quê? Se o CPF for solicitado apenas fora da estrutura de repetição, caracteriza-se que ele seria comum a todos os contribuintes, e isso não é verdade, além de ele ser nosso *flag*, ou seja, ele precisa ser informado a cada repetição, pois por meio dele se verifica quando o usuário deseja interromper a estrutura.

Verifique na tabela abaixo a representação do teste de mesa para a resolução deste enunciado. Para efeito de exemplos, os CPFs informados não receberam nenhuma validação, tampouco existem.

Teste para 5 CPFs válidos							
Salário Mínimo	CPF	Salário	Qtde Dependentes	Salário Base	Qtde Salário Mínimo	Alíquota	Valor IR
240,00	123	1.000,00	1	988,00	4,12	15	148,20
	456	900,00	2	876,00	3,65	15	131,40
	789	1.100,00	3	1.064,00	4,43	15	159,60
	012	800,00	4	752,00	3,13	15	112,80
	345	1.200,00	3	1.164,00	4,85	15	174,60
	0						

Uma empresa deseja fazer um levantamento de seu estoque. Para isto possui, para cada tipo de produto, seu nome, valor unitário e sua quantidade em estoque. A empresa deseja saber o Valor Contábil em estoque de cada produto, quantos tipos de produtos existem em estoque, o saldo de todos os produtos e Total Contábil Geral. A digitação terminará quando o nome do produto for igual a "FIM".

Como você procederia?

```
Algoritmo Estoque
Variáveis
        Nome : String
        ValorUnitario, Quantidade, SubTotal, TotalGeral, Estoque : Real
        QtdeTipos : Inteiro
Início
        Estoque = 0
        QtdeTipos = 0
        TotalGeral = 0
        Escreva "Informe o nome do produto"
        Leia Nome
        Enquanto (Nome <> "FIM") Faça
                QtdeTipos = QtdeTipos + 1
                Escreva "Informe o valor Unitário"
                Leia ValorUnitario
                Escreva "Informe a quantidade em estoque"
                Leia Quantidade
```

```
            SubTotal = (Quantidade * ValorUnitario)
            Estoque = Estoque + Quantidade
            TotalGeral = TotalGeral + SubTotal
            Escreva "O valor total do produto ", Nome, " é ",  SubTotal
            Escreva "Informe o nome do produto"
            Leia Nome
        FimEnquanto
        Escreva "Existem ", QtdeTipos, " tipos de produtos"
        Escreva "O estoque em volumes é ", Estoque
        Escreva "O valor comercial do estoque é ", TotalGeral
Fim
```

Este algoritmo se assemelha a um levantamento de estoque (inventário). De modo idêntico ao exercício anterior, não se sabe quantas vezes ocorrerá a repetição, ou seja, quantos itens existem na suposta empresa.

```
...
Algoritmo Estoque
Variáveis
        ...
Início
        Estoque = 0
        QtdeTipos = 0
        TotalGeral = 0
        ...
        Enquanto (Nome <> "FIM") Faça
            ...
        FimEnquanto
        ...
Fim
```

Observe a inicialização das variáveis. São elas que exibirão ao usuário aquilo que ele pede, são todas acumuladoras. Sendo assim, precisam inicializar com zero. A variável **Estoque** terá a quantidade de volumes, **QtdeTipos** se refere a quantidade itens, e a **TotalGeral** trará o **Estoque Contábil**, ou seja, o valor financeiro do estoque da empresa. Estas três varíaveis sofrerão alterações a cada produto informado pelo usuário.

```
...
Algoritmo Estoque
Variáveis
        ...
Início
        ...
        Escreva "Informe o nome do produto"
        Leia Nome
        Enquanto (Nome <> "FIM") Faça
```

```
            ...
        FimEnquanto
        ...
Fim
```

Igualmente ao exercício anterior, temos aqui nosso *flag*, que é solicitado antes de a estrutura começar e depois, dentro da estrutura, antes que esta termine. Note que apenas houve alteração no tipo de dado utilizado, pois usamos agora um **literal**.

```
...
Algoritmo Estoque
Variáveis
        ...
Início
        ...
        Enquanto (Nome <> "FIM") Faça
                QtdeTipos = QtdeTipos + 1
                Escreva "Informe o valor Unitário"
                Leia ValorUnitario
                Escreva "Informe a quantidade em estoque"
                Leia Quantidade
                ...
        FimEnquanto
        ...
Fim
```

Sempre que a estrutura ocorre, significa que um novo item terá seus dados informados. Sendo assim, devemos incrementar nossa variável responsável pela identificação final da quantidade de itens informados, e logo então, solicitar ao usuário o valor unitário de tal produto e a quantidade existente dele no estoque.

```
...
Algoritmo Estoque
Variáveis
        ...
Início
        ...
        Enquanto (Nome <> "FIM") Faça
                ...
                SubTotal = (Quantidade * ValorUnitario)
                Estoque = Estoque + Quantidade
                TotalGeral = TotalGeral + SubTotal
                Escreva "O valor total do produto ", Nome, " é ",  SubTotal
                ...
        FimEnquanto
        ...
Fim
```

Uma vez tendo o usuário informado os valores solicitados e necessários, devem ser calculados (acumulados) os valores desejados pelo usuário. Um deles é exibido na própria estrutura, pois se refere a cada produto, seu subtotal.

```
...
Algoritmo Estoque
Variáveis
        ...
Início
        ...
        Enquanto (Nome <> "FIM") Faça
                ...
                Escreva "Informe o nome do produto"
                Leia Nome
        FimEnquanto
        ...
Fim
```

Ao se chegar no final da estrutura de repetição, para que ela reinicie (ou termine), deve ser informado pelo usuário o nome do próximo produto, que pode inclusive ser o valor que terminará a estrutura (**"FIM"**).

```
...
Algoritmo Estoque
Variáveis
        ...
Início
        ...
        Enquanto (Nome <> "FIM") Faça
                ...
        FimEnquanto
        Escreva "Existem ", QtdeTipos, " tipos de produtos"
        Escreva "O estoque em volumes é ", Estoque
        Escreva "O valor comercial do estoque é ", TotalGeral
Fim
```

Ao terminar o processo de levantamento de estoque com todos os produtos da empresa, os totais já foram obtidos. Sendo assim, basta informá-los ao usuário.

Veja abaixo o teste de mesa para a resolução do algoritmo para o problema apresentado.

Teste para 5 produtos informados							
Nome	(Nome<>"FIM")	Qtde Tipos	Valor Unitário	Quantidade	Sub Total	Estoque	Total Geral
		0				0	0
"Parafuso"	Sim	1	1,00	2	2,00	2	2,00

"Prego"	Sim	2	2,00	4	8,00	4	10,00
"Porca"	Sim	3	4,00	8	32,00	12	42,00
"Fio"	Sim	4	8,00	1	8,00	13	50,00
"Bucha"	Sim	5	3,00	3	9,00	16	59,00
"FIM"	Não						

Procure observar com atenção as especificidades contidas em cada exemplo colocado até aqui, pois estas situações descritas são práticas e muito relacionadas com ações do cotidiano. Se permanecer qualquer dúvida, releia os textos explicativos, refaça o percurso e pesquise!

4.4. ESTRUTURAS DE REPETIÇÃO CONDICIONAL, COM TESTE NO FINAL

Como visto nos exemplos de teste no início, pode ocorrer a situação de a repetição nunca ocorrer, uma vez que, para ela ocorrer a primeira vez, passa antes pelo teste condicional. Esta situação já não se dá nos algoritmos com testes condicionais ao final da repetição, pois toda a estrutura deverá ocorrer ao menos uma vez, para assim poder ser feito o teste condicional.

Um comerciante deseja pagar todas as suas contas do dia. Faça um algoritmo que solicite o valor a ser pago e a Taxa de Juros, caso a conta esteja em atraso. Calcule o Total dos valores de multa e o valor total a ser pago. O algoritmo deverá também solicitar quantas contas serão pagas no dia. Informe também o total de contas em atraso e o total de contas pagas em dia. Atenção que não há possibilidade de não haver contas a pagar, ou seja, haverá no mínimo uma conta a ser paga no dia.

```
Algoritmo Contas
Variáveis
        QtdeContasAPagar, QtdeContasPagas, QtdeContasEmAtraso : Inteiro
        ValorConta, TaxaDeJuros, TotalVlrJuros, TotalAPagar : Real
        PagoEmAtraso : String
Início
        QtdeContasPagas = 0
        QtdeContasEmAtraso = 0
        TotalVlrJuros = 0
        TotalAPagar = 0
        Escreva "Informe quantas contas serão pagas"
        Leia QtdeContasAPagar
        Repita
                Escreva "Informe o valor da conta"
                Leia ValorConta
```

```
            Escreva "Esta conta está sendo paga em atraso (S/N) ?"
            Leia PagoEmAtraso
            Se (PagoEmAtraso = "S") Então
               Escreva "Informe o índice da taxa de juros (0-100)"
               Leia TaxaDeJuros
               QtdeContasEmAtraso = QtdeContasEmAtraso + 1
            Senão
               TaxaDeJuros = 0
            FimSe
            TotalVlrJuros = TotalVlrJuros + (ValorConta * TaxaDeJuros / 100)
            TotalAPagar = TotalAPagar + ValorConta
            QtdeContasPagas = QtdeContasPagas + 1
         Até (QtdeContasPagas = QtdeContasAPagar)
         Escreva "O Valor total pago por multa é ", TotalVlrJuros
         Escreva "O Valor total pago é ", (TotalAPagar + TotalVlrJuros)
         Escreva "Contas em atraso : ", QtdeContasEmAtraso
         Escreva "Contas em dia : ", (QtdeContasAPagar-
         QtdeContasEmAtraso)
Fim
```

Esta situação traz um levantamento financeiro, para previsão de pagamento. Não sabemos quantas contas serão pagas, mas o usuário as informará. O enunciado ainda diz que haverá sempre o mínimo de uma conta. Isso caracteriza que a repetição ocorrerá no mínimo uma vez.

Releia o parágrafo anterior! Está na última frase a dica para fazer o teste no fim, após a estrutura ocorrer, ou seja, antes de acabar.

```
...
Algoritmo Contas
Variáveis
      ...
Início
      QtdeContasPagas = 0
      QtdeContasEmAtraso = 0
      TotalValorJuros = 0
      TotalAPagar = 0
      Escreva "Informe quantas contas serão pagas"
      Leia QtdeContasAPagar
      Repita
            ...
      Até (QtdeContasPagas = QtdeContasAPagar)
      ...
Fim
```

Veja a inicialização em 0 (zero) das variáveis acumuladoras, diretamente ligadas ao que o usuário deseja saber ao final. A **QtdeContasPagas** é a responsável pela

verificação de até quando a estrutura de repetição ocorrerá. Ela garantirá que a quantidade de contas a pagar não seja inferior à quantidade de contas pagas. **QtdeContasEmAtraso** será responsável por informar ao usuário a quantidade de contas em atraso, não havendo necessidade de existir alguma. **TotalValorJuros** e **TotalAPagar**, se referem aos valores totais solicitados pelo usuário. Note também que, antes de a estrutura começar (por meio do **Repita**), é solicitada ao usuário a quantidade de contas que serão pagas.

```
...
Algoritmo Contas
Variáveis
        ...
Início
        ...
        Repita
                Escreva "Informe o valor da conta"
                Leia ValorConta
                Escreva "Esta conta está sendo paga em atraso (S/N) ?"
                Leia PagoEmAtraso
                ...
        Até(QtdeContasPagas = QtdeContasAPagar)
        ...
Fim
```

Para cada ocorrência da estrutura de repetição deverá ser solicitado ao usuário o valor da conta e se ela está sendo paga em atraso ou não.

```
...
Algoritmo Contas
Variáveis
        ...
Início
        ...
        Repita
                ...
                Se (PagoEmAtraso = "S") Então
                   Escreva "Informe o índice da taxa de juros (0-100)"
                   Leia TaxaDeJuros
                   QtdeContasEmAtraso = QtdeContasEmAtraso + 1
                Senão
                   TaxaDeJuros = 0
                FimSe
                ...
        Até (QtdeContasPagas = QtdeContasAPagar)
        ...
Fim
```

Uma vez informado pelo usuário se foi ou não pago em atraso a conta, deve-se executar os procedimentos necessários para cada situação. Quando for pago em

atraso, existe uma taxa de juros que deve ser utilizada, além da necessidade de acumular a quantidade de contas pagas em atraso, pois é uma informação desejada pelo usuário. Note que, no caso de a conta ter sido paga em dia, é atribuído uma taxa de 0 (zero) . Isso facilita a operação de cálculo do valor de juros e não causa nenhum problema, pois zero, multiplicado por qualquer valor, resulta em zero.

```
...
Algoritmo Contas
Variáveis
        ...
Início
        ...
        Repita
                ...
                TotalVlrJuros = TotalVlrJuros + (ValorConta * TaxaDeJuros / 100)
                TotalAPagar = TotalAPagar + ValorConta
                QtdeContasPagas = QtdeContasPagas + 1
        Até (QtdeContasPagas = QtdeContasAPagar)
        ...
Fim
```

Tendo sido solicitado ao usuário o valor da conta, se esta foi paga em atraso, ou não, e identificada a taxa de juros, efetuam-se os cálculos (acumuladores) desejados pelo usuário ao final do algoritmo. Também é incrementada a variável contadora (*flag*). Perceba que, na estrutura com teste no início (**Enquanto...FimEnquanto**), quando retorna **verdadeiro**, ela é executada, porém na estrutura de repetição com teste no final (**Repita...Até**), quando for retornado **verdadeiro**, indica o término da estrutura de repetição.

```
...
Algoritmo Contas
Variáveis
        ...
Início
        ...
        Repita
                ...
        Até (QtdeContasPagas = QtdeContasAPagar)
        Escreva "O Valor total pago por multa é ", TotalVlrJuros
        Escreva "O Valor total pago é ", (TotalAPagar + TotalVlrJuros)
        Escreva "Contas em atraso : ", QtdeContasEmAtraso
        Escreva "Contas em dia : ", (QtdeContasAPagar-
        QtdeContasEmAtraso)
Fim
```

Tendo terminado a estrutura de repetição, e isso ocorre quando todas as contas foram informadas pelo usuário, os resultados solicitados por ele são exibidos. Note que ele solicitou também a quantidade de contas pagas em dia e não foi criada

nenhuma variável para esta informação, e ela realmente não se faz necessária, pois, se tem o total de contas pagas, obteve-se a quantidade de contas pagas em atraso. Sendo assim, basta apenas uma operação de subtração nos valores conhecidos para identificação de quantas contas foram pagas em dia.

Teste para 5 contas a serem pagas								
Qtde Contas A Pagar	Valor Conta	Pago Em Atraso	(Pago Em Atraso ="S")	Taxa De Juros	Total Valor Juros	Total A Pagar	Qtde Contas Em Atraso	Qtde Contas Pagas
					0	0	0	0
5	100,00	"N"	Não	0	0	100,00		1
	200,00	"S"	Sim	10	20,00	200,00	1	2
	400,00	"N"	Não	0		600,00		3
	800,00	"N"	Não	0		1.400,00		4
	1.600,00	"S"	Sim	20	340,00	3.000,00	2	5

Vamos a uma atividade?

Faça um algoritmo que solicite números inteiros até que um número ímpar seja digitado. O algoritmo deverá informar, ao final, quantos números pares foram informados.

```
Algoritmo NumeroPares
Variáveis
        Numero, QtdeNumerosPares : Inteiro
Início
        QtdeNumerosPares = 0
        Repita
                Escreva "Informe um número"
                Leia Numero
                QtdeNumerosPares = QtdeNumerosPares + 1
        Até (Resto(Numero, 2) <> 0)
        QtdeNumerosPares = QtdeNumerosPares - 1
        Escreva "Números pares informados ", QtdeNumerosPares
Fim
```

Este exercício é simples, talvez sinta uma dificuldade apenas em saber como identificar quando um número é ímpar ou par, mas para resolver isso podemos abstrair a existência de uma função chamada **Resto( ).** Esta recebe dois argumentos: o número a ser dividido e o número pelo qual este número será dividido (numerador e denominador).

O usuário deseja saber quantos números pares foram informados. Identificamos a variável contadora e temos que presumir que o primeiro número digitado já seja

um ímpar, então ela começa com 0 (zero). Bem, para verificar se um número é par ou ímpar, também é preciso tê-lo antes. Lembrando explicações anteriores, é esta característica que nos identifica quando usar o teste no início ou no final, mas não há nada que impeça o uso de uma ou outra estrutura para resolução deste exercício.

No momento em que for identificado como ímpar o número digitado pelo usuário, a estrutura de repetição se finda e é exibida ao usuário a quantidade de números pares informados. No entanto, antes é feita uma subtração deste valor, pois quando a estrutura acaba, foi lido um número ímpar.

Teste para digitação de dois números pares		
Número	Qtde Números Pares	(Resto(Número, 2) <> 0)
	0	
20	1	Não
40	2	Não
45	3	Sim
	2	

Mostramos nesta unidade problemas que trabalham sobre uma grande variedade de dados, sendo possível saber quantos são estes dados antes do processamento, como também se esta quantidade pode estar ligada diretamente à interação com o usuário. À primeira situação, demos o nome de repetição contada. Já à segunda, de repetição condicional com teste no início ou ao término da estrutura.

Caracterizamos como repetição uma série de instruções que devem ser executadas da mesma forma, por mais de uma vez, mas nada impede que seja executada uma única vez.

Esperamos que os exemplos tenham auxiliado na compreensão de mais esses conceitos! Vamos em frente!

Chegamos ao final da leitura desta unidade e não poderíamos deixar de parabenizá-lo por estar conosco até aqui e cheio de vontade para continuar o estudo da próxima unidade.

UNIDADE V

CONJUNTOS

Nesta nova unidade, abordaremos conjuntos e métodos de classificação e pesquisa, que são noções importantes para:

- resolução de problemas que trabalham com um conjunto de dados que normalmente são ligados a um problema, como a idade dos alunos de uma única turma;
- classificar, em forma ascendente ou descendente, os dados de um determinado conjunto;
- pesquisar por um determinado dado dentre os elementos em um conjunto.

Portanto, no término desta etapa de estudo é fundamental que você seja capaz de:

- reconhecer a necessidade de um conjunto de dados como tipo de uma variável;
- identificar, popular e varrer uma matriz, unidimensional e bidimensional.

5.1. CONJUNTOS

Você deve lembrar que definimos variável como uma entidade criada para permitir o acesso a uma posição de memória, onde se armazena uma informação de um determinado tipo de dado pela simples referência a um nome simbólico.

Compreender o que é uma variável e sua função no campo da informática é importante, mas devemos lembrar que existem diversos tipos de variáveis.

De fato, são comuns situações práticas em que se necessita referenciar um grupo de variáveis do mesmo tipo pelo mesmo nome simbólico.

Saliba (1993) conceitua como **variáveis indexadas** o conjunto de variáveis do mesmo tipo, referenciáveis pelo mesmo nome e individualizadas entre si, por meio de sua posição dentro desse conjunto.

Um conjunto, na linguagem matemática, pode ser definido como uma coleção de elementos e, quando em um mesmo conjunto, dizemos que esses elementos são do mesmo domínio.

Por exemplo, se temos um conjunto de vogais, qual será o domínio do conjunto? Só podem ser letras vogais (a, e, i, o, u); e ainda, tendo um conjunto de elementos numéricos, qual será o domínio deste conjunto? Serão números...

Daí podemos entender domínio, de maneira simplificada, como o tipo de dados ou elementos que determinado conjunto pode conter.

Lopes e Garcia (2002) definem como **vetor** um arranjo de elementos armazenados, um após o outro, todos com o mesmo nome. Lembra, de forma análoga à matriz-linha da matemática, várias colunas para uma única linha.

Ascencio & Campos (2002) definem **vetor** como uma variável composta homogênea unidimensional formada por uma sequência de variáveis, todas do mesmo tipo, com o mesmo identificador (mesmo nome) e alocadas sequencialmente na memória. Ou seja, uma vez que as variáveis tenham o mesmo nome, o que as distinguirá é um índice, que referencia sua localização dentro da estrutura.

5.1.1, VETORES OU MATRIZ DE UMA ÚNICA LINHA

Um vetor de 8 elementos pode ser representado graficamente da seguinte forma :

0	1	2	3	4	5	6	7
2	4	5	8	12	3	56	34

Podemos, de maneira análoga, ter 8 variáveis, todas com o mesmo nome e diferentes por sua posição dentro do vetor, que é indicado por um índice. Quando se tem apenas uma linha, podemos omiti-la e colocar somente a coluna. Assumindo que nossa variável se chame **X**, tem-se : **X_0 =2, X_1=4, X_2=5, X_3=8, X_4=12, X_5=3, X_6=56, X_7=34**. Em pseudocódigo, representamos este vetor da seguinte forma: **X[0]=2, X[1]=4, X[2]=5, X[3]=8, X[4]=12, X[5]=3, X[6]=56, X[7]=34**. Trabalhemos alguns exemplos básicos para fixação deste novo conceito.

Faça um algoritmo que leia N números e, após sua leitura, mostre ao usuário em ordem inversa à digitada por ele.

```
Algoritmo OrdemInversa
Variáveis
        Entrada[10], I, N : Inteiro
Início
        Repita
                Escreva "Informe a quantidade de números que digitará"
                Leia N
        Até ((N > 0) E (N <= 10))
        Para I = 1 Até N Faça
              Escreva "Informe o número ", I, " de ", N
              Leia Entrada[I]
        FimPara
        I = N
        Enquanto ( I > 0) Faça
                Escreva "Número ", I, " de ", N, " é ", Entrada[I]
                I = I – 1
        FimEnquanto
Fim
```

No desenvolvimento deste pseudocódigo, são trabalhados alguns novos conceitos, ligados diretamente a conjuntos, e estes conceitos serão detalhados agora, antes da explicação minuciosa do problema proposto.

5.1.2. TAMANHO FÍSICO E TAMANHO LÓGICO

Quando definimos uma varíavel de conjunto (vetor ou matriz), devemos definir também a capacidade máxima de armazenamento que ela aceita, ou seja, a quantidade de elementos (valores) que ela suportará.

A este tamanho máximo, damos o nome de **Tamanho Físico**. Porém, observem com atenção que a *reserva* deste espaço não implica sua real utilização: podemos optar (necessitar) usar menos (nunca mais).

Ao que realmente foi utilizado do conjunto, ou seja, a esta quantidade de elementos utilizados é dado o nome de **Tamanho Lógico**.

```
...
Algoritmo OrdemInversa
Variáveis
        Entrada[10], I, N : Inteiro
Início
        ...
Fim
```

Tamanho físico

No momento em que seu pseudocódigo define um **Tamanho Físico** para um conjunto e a alimentação (entrada de dados) deste conjunto será feita de forma direta pelo usuário, devemos garantir que este limite não seja ultrapassado.

Veja, na ilustração acima, que o **Tamanho Físico** para a variável **Entrada** é **10**.

```
...
Algoritmo OrdemInversa
Variáveis
        ...
Início
        Repita
                Escreva "Informe a quantidade de números que digitará"
                Leia N
        Até ((N > 0) E (N <= 10))
        ...
Fim
```

Observe que o enunciado não diz uma quantidade determinada de valores que serão informados pelo usuário. Entretanto, deixa claro que esta quantidade será informada pelo usuário antes da digitação. Isso pode ser afirmado devido ao fato

de não estipular nenhuma regra (*flag*) para término da digitação. Sendo então assumido que a responsabilidade de informação de quantidade de valores que serão *entradas* no conjunto é do usuário, seu pseudocódigo **deve** garantir que esta quantidade (**tamanho lógico**) não ultrapasse o máximo imposto por você (**tamanho físico**).

Como garantir isso?

Deixe-me explicar: solicita-se esta quantidade em uma estrutura de repetição (neste caso optado pela condicional com teste no fim), que termina apenas quando o tamanho informado esteja dentro dos limites aceitos. Observe o exemplo a seguir.

```
...
Algoritmo OrdemInversa
Variáveis
        ...
Início
        ...
        Para I = 1 Até N Faça
                Escreva "Informe o número ", I, " de ", N
                Leia Entrada[I]
        FimPara
        ...
Fim
```

Tendo a quantidade de valores que deverão ser informados, podemos assumir que conhecemos a quantidade de vezes em que deverá ser solicitada ao usuário a digitação de um número. Assumimos então que a melhor estrutura para esta situação é a repetição contada. Note que, para informar entrada de valores em um conjunto, utilizamos **[ ]** (colchetes) e, dentro deles um valor que especifica a posição (**índice**) no vetor onde o valor será armazenado.

```
...
Algoritmo OrdemInversa
Variáveis
        ...
Início
        ...
        I = N
        Enquanto ( I > 0) Faça
                Escreva "Número ", I, " de ", N, " é ", Entrada[I]
                I = I – 1
        FimEnquanto
Fim
```

Uma vez que todos os valores tenham sido informados, devemos executar o solicitado no enunciado: a informação destes valores ao usuário, porém de forma inversa à que foi entrada. Para isso, basta começarmos a estrutura de repetição, tendo o contador iniciando com o endereço do último índice e, então, ao invés de incrementar o contador faz seu decremento, seu decréscimo.

Teste para três valores							
N	(N>0) E (N<=10)	I	(I<=10)	Entrada[I]	Entrada[10]	(I>0)	Saída
20	Não						
-5	Não						
3	Sim	1	Sim	10	1 2 3 4 5 6 7 8 9 10 10		
		2	Sim	5	1 2 3 4 5 6 7 8 9 10 10 5		
		3	Sim	15	1 2 3 4 5 6 7 8 9 10 10 5 15		
		4	Não				
		3		15		Sim	15
		2		5		Sim	5
		1		10		Sim	10
		0				Não	

Observe este outro exemplo:

Faça um algoritmo que seja capaz de fazer a soma de duas matrizes (índice por índice) de 5 vetores cada. O resultado deverá estar em uma terceira matriz. Deverão ser solicitados ao usuário os valores das duas matrizes. Exiba ao final o resultado obtido.

```
Algoritmo SomaMatriz
Variáveis
        A[5], B[5], C[5], I : Inteiro
Início
        Para I = 1 Até 5 Faça
                Escreva "Informe o número ", I, " de 5 da primeira matriz"
                Leia A[I]
                Escreva "Informe o número ", I, " de 5 da segunda matriz"
```

```
            Leia B[I]
            C[I] = (A[I] + B[I])
            Escreva "A soma de ", A[I], " e ", B[I], " é ", C[I]
        FimPara
Fim
```

Este problema é relativamente simples, pois solicita ao usuário dois valores (por cinco vezes) e, a cada par de valores informados, efetuamos uma soma e a armazenamos em uma terceira variável, que, neste caso, é um índice de uma matriz. Na resolução apresentada, foi optado por uma única estrutura de repetição, que é a mais otimizada, mas poderia ser apresentada a resolução em várias outras maneiras.

Teste de mesa

I	A[5]	B[5]	C[5]
1	1 2 3 4 5 2	1 2 3 4 5 3	1 2 3 4 5 5
2	1 2 3 4 5 2 6	1 2 3 4 5 3 7	1 2 3 4 5 5 13
3	1 2 3 4 5 2 6 14	1 2 3 4 5 3 7 15	1 2 3 4 5 5 13 29
4	1 2 3 4 5 2 6 14 30	1 2 3 4 5 3 7 15 31	1 2 3 4 5 5 13 29 61
5	1 2 3 4 5 2 6 14 30 62	1 2 3 4 5 3 7 15 31 63	1 2 3 4 5 5 13 29 61 125

5.2. MATRIZ

Uma matriz é uma variável composta homogênea bidimensional, formada por uma sequência de variáveis, todas do mesmo tipo, com o mesmo identificador (mesmo nome) e alocadas sequencialmente na memória. Uma vez que as variáveis têm o mesmo nome, o que as distingue são índices que referenciam sua localização dentro da estrutura. Uma variável do tipo matriz é composta por linhas e colunas.

	1	2	3	4	5
1	X[1, 1]				
2					
3				[X[3, 4]	

Observe a seguinte situação:

Faça um algoritmo que solicite quatro notas e seus respectivos pesos. Estes valores deverão estar armazenados em uma única matriz. Ao final, exiba média ponderada dessas notas. Assuma que as notas são de zero a cem (0 a 100).

```
Algoritmo Boletim
Variáveis
        Numerador, Denominador, Aluno[4, 2], I : Inteiro
        Media : Real
Início
        Numerador = 0
        Denominador = 0
        Para I = 1 Até 4 Faça
                Escreva "Informe a nota ", I, " do aluno"
                Leia Aluno[I, 1]
                Escreva "Informe o peso da nota ",  I
                Leia Aluno [I, 2]
                Numerador = Numerador + (Aluno[I, 1] * Aluno[I, 2])
                Denominador = Denominador + Aluno[I, 2]
        FimPara
        Media = (Numerador / Denominador)
        Escreva "A média do aluno é ", Media
Fim
```

Observe este algoritmo e compare-o com os exemplos anteriores. No que ele difere dos outros?

Note que nesse algoritmo, *duas* informações diferentes serão armazenadas em uma única variável: uma **matriz bidimensional**. A matriz bidimensional pode ser vista como linhas que possuem colunas. Usando esta analogia, podemos dizer ainda que, em relação ao enunciado, cada linha representa um bimestre, e que a primeira coluna terá as notas de cada bimestre e a segunda os pesos de cada nota.

Bem, conforme se sabe, cada nota se refere a um bimestre. Mas, como identificar o bimestre neste algoritmo? Conforme foi tratado, cada posição da matriz possui um índice (um número) que identifica um endereço dentro da matriz onde os dados estão armazenados. Pois bem, este índice indica o bimestre, porém esta informação não é relevante ao problema.

```
...
Algoritmo Boletim
Variáveis
        Numerador, Denominador, Aluno[4, 2], I : Inteiro
        Media : Real
Início
        ..
        Para I = 1 Até 4 Faça
                Escreva "Informe a nota ", I, " do aluno"
                Leia Aluno[I, 1]
                Escreva "Informe o peso da nota ",  I
                Leia Aluno [I, 2]
                ..
        FimPara
        ..
Fim
```

Note que, na entrada de dados, a variável que representa a matriz tem dentro dos colchetes dois valores – **Aluno[I, 2]**. Um deles é uma variável (**I**), pois, como são quatro notas, a cada iteração, a linha tem que mudar. O segundo valor é constante, não muda, e se refere à coluna onde o dado será armazenado. A coluna um terá a nota; e a coluna dois, o peso. Vejamos a simulação representada no teste de mesa.

Teste de mesa				
Numerador	Denominador	I	Aluno[ ]	Média
0	0			
100	1	1	100 | 1 | | | | | |	
120	3	2	100 | 1 10 | 2 | | | |	
390	6	3	100 | 1 10 | 2 90 | 3 | |	
470	10	4	100 | 1 10 | 2 90 | 3 20 | 4	
				4,70

São muitas as situações nas quais a construção de algoritmos relativos a uma matriz assumem novas características. Portanto, é muito importante que sejam observados os diferentes exemplos com bastante atenção. Vejamos esta outra situação:

Um comerciante deseja pagar todas as suas contas do dia. Faça um algoritmo que solicite o valor a ser pago e a taxa de juros para a conta. Lembre-se de que, caso a conta seja paga em dia, a taxa de juros é zero. Calcule o total dos valores de multa e o valor total a ser pago. O algoritmo deverá também solicitar quantas contas serão pagas no dia.

```
Algoritmo Pagamento
Variáveis
        I, QtdeContas : Inteiro
        Contas[2, 5], TotalMultas, TotalContas: Real
Início
        Repita
                Escreva "Informe quantas contas serão pagas"
                Leia QtdeContas
        Até (QtdeContas > 0 E QtdeContas <= 5)
        TotalMultas = 0
        TotalContas = 0
        Para I = 1 Até QtdeContas Faça
                Escreva "Informe o valor da conta ", I, " de ", QtdeContas
                Leia Contas[1, I]
                Escreva "Informe a taxa de juros da conta ", I, " de ", QtdeContas
                Leia Contas[2, I]
                TotalMultas = TotalMultas + (Contas[1, I] * Contas[2, I] / 100)
                TotalContas = TotalContas + Contas[1, I]
        FimPara
        Escreva "O total a ser pago em multa é ", TotalMultas
        Escreva "O valor total a ser pago é ", TotalContas + TotalMultas
Fim
```

Para efeito didático, este exemplo trata a matriz como duas linhas, em que cada linha tem cinco colunas. O que varia neste exemplo é a coluna onde a informação será armazenada, diferentemente do anterior, onde a coluna era constante. Observe o teste de mesa:

Teste de mesa

I	QtdeContas	TotalMultas	TotalContas	Contas[ ]
	3	0	0	
1		10,00	100,00	100,00 / 10,00
2		12,50	150,00	100,00 / 10,00 ; 50,00 / 5,00
3		20,00	175,00	100,00 / 10,00 ; 50,00 / 5,00 ; 25,00 / 30,00

5.3. CLASSIFICAÇÃO E PESQUISA

O conjunto de aplicações possíveis para matrizes é enorme, mas dois são comuns, importantes e necessários. São eles: a classificação de matrizes e a pesquisa.

A classificação consiste em ordenar os elementos da matriz em uma determinada classificação, seguindo o critério necessário para cada problema. Este critério pode ser alfabético, para dados literais; crescente ou decrescente para dados numéricos. Vários métodos de classificação são trabalhados para o desenvolvimento desta tarefa, tais como **Ordenação por Inserção**, **ShellSort**, **Bubblesort**, **QuickSort** e **Heapsort**.

A pesquisa consiste na verificação da existência de um valor dentro de um conjunto de dados (matriz).

Este texto cuidará apenas do método **Bubblesort** para ordenação e trabalhará dois métodos bem difundidos para pesquisa, que são a pesquisa sequencial e a pesquisa binária.

5.3.1. MÉTODO BOLHA DE CLASSIFICAÇÃO – *BUBBLESORT*

Não é o mais eficiente dos métodos existentes, entretanto é considerado o mais popular, devido à sua simplicidade.

A filosofia básica deste método consiste em ler todo o vetor, comparando os elementos vizinhos entre si. Caso estejam fora de ordem, eles trocam de posição entre si. Procede-se assim até o final do vetor. Na primeira leitura, verifica-se que o último elemento do vetor já está em seu devido lugar (no caso de ordenação crescente, ele é o maior de todos). A segunda leitura é análoga à primeira e vai até o penúltimo elemento. Este processo é repetido até que seja feito um número de leituras igual ao número de elementos a serem ordenados, menos um. Ao final deste processo, o vetor estará classificado segundo o critério escolhido.

Classificação de um vetor de cinco elementos reais em ordem crescente.

```
Algoritmo Bubble_Sort
Variáveis
        Numeros [5] : Real
        I, J : Inteiro
        Aux : Real
Início
        Para I = 1 Até 5 Faça
                Escreva "Informe o número ", I, " de 5"
                Leia Numeros[I]
        Fimpara
        J = 5
        Enquanto (J > 1) Faça
                Para I = 1 até (J-1) Faça
                        Se (Numeros[I] > Numeros[I+1]) então
```

```
                                    Aux = Numeros[I]
                                    Numeros[I] = Numeros[I+1]
                                    Numeros[I+1] = Aux
                            FimSe
                    Fimpara
                    J = J –1
            FimEnquanto
            Escreva "Vetor Ordenado"
            Para I = 1 Até 5 Faça
                    Escreva Numeros[I]
            Fimpara
Fim
```

Observe que os valores que serão ordenados são inicialmente solicitados ao usuário, ou seja, o usuário pode, e fatalmente o fará, informar estes valores em uma ordem aleatória, sem classificação. Veja o trecho de código onde esta entrada de dados ocorre.

```
...
Algoritmo Bubble_Sort
Variáveis
        ...
Início
        Para I = 1 Até 5 Faça
                Escreva "Informe o número ", I, " de 5"
                Leia Numeros[I]
        Fimpara
        ...
Fim
```

Podemos aceitar que os valores informados são os existentes na matriz a seguir.

	1	2	3	4	5
Numeros[5]	**2**	**256**	**32**	**128**	**1**

Acompanhando o trecho do algoritmo listado abaixo, podemos efetuar o teste de mesa e avaliar as alterações existentes, as quais garantirão a classificação/ordenação da matriz.

```
...
Algoritmo Bubble_Sort
Variáveis
        ...
Início
        ..
        J = 5
        Enquanto (J > 1) Faça
```

```
                Para I = 1 até (J-1) Faça
                        Se (Numeros[I] > Numeros[I+1]) então
                                Aux = Numeros[I]
                                Numeros[I] = Numeros[I+1]
                                Numeros[I+1] = Aux
                        FimSe
                Fimpara
                J = J –1
        FimEnquanto
        ..
Fim
```

Note a variável J sendo inicializada com a quantidade máxima de elementos existentes na matriz. Isso é necessário devido ao fato de cada leitura ir até o penúltimo elemento da matriz. A estrutura Enquanto...FimEnquanto garante a execução da comparação até o momento em que todos os elementos estiverem classificados. A cada comparação de um elemento com todos os outros da matriz, esta variável é decrementada em um, o que garante que os últimos elementos já estejam em ordem, ou seja, os menores valores são sempre levados ao início (topo) da matriz, como se borbulhassem. Daí o nome Ordenação pelo Método da Bolha. Vejamos o teste de mesa:

J	(J > 1)	I	Números[I]	Números[I+1]	Aux	
5	V	1	2	256		1 2 3 4 5 / 2 256 32 128 1
5	V	2	256	32	256	1 2 3 4 5 / 2 256 32 128 1
			32	256		1 2 3 4 5 / 2 32 256 128 1
5	V	3	256	128		1 2 3 4 5 / 2 32 256 128 1
			128	256	256	1 2 3 4 5 / 2 32 128 256 1
5	V	4	256	1		1 2 3 4 5 / 2 32 128 256 1
			1	256	256	1 2 3 4 5 / 2 32 128 1 256
4						

Observe, no teste de mesa, que a variação de posição entre os índices da matriz ocorreu justamente com o maior valor existente entre todos. Originalmente, ele estava no segundo índice e, ao terminar todas as comparações, ele passou para a última posição. É possível constatar que a leitura ocorreu sempre em relação a um elemento e o seu próximo elemento. Agora que existe a garantia de que o último elemento é realmente o maior, ele não será mais usado na próxima leitura. Veja o teste de mesa da segunda leitura.

J	(J > 1)	I	Números[I]	Números[I+1]	Aux	
4	V	1	2	32		1 2 3 4 5 2 32 128 1 256
4	V	2	32	128		1 2 3 4 5 2 32 158 1 256
4	V	3	128	1		1 2 3 4 5 2 32 158 1 256
			1	128	128	1 2 3 4 5 2 32 1 128 256

As alterações nesta segunda leitura diminuíram. Na realidade, ocorreu uma única vez, mas isso não é regra, pois esta situação está intimamente ligada aos números existentes na matriz. Veja agora a terceira leitura.

J	(J > 1)	I	Números[I]	Números[I+1]	Aux	
3	V	1	2	32		1 2 3 4 5 2 32 1 128 256
3	V	1	2	32		1 2 3 4 5 2 32 1 128 256
3	V	2	32	1		1 2 3 4 5 2 32 158 128 256
			1	32	32	1 2 3 4 5 2 1 32 128 256

Segue o teste de mesa para a última leitura para este exemplo.

J	(J > 1)	I	Números[I]	Números[I+1]	Aux	
2	V	1	2	1		1 2 3 4 5 2 1 32 128 256
			1	2	2	1 2 3 4 5 1 2 32 128 256
1	F					

No momento em que a última leitura é feita, a matriz está totalmente classificada.

Classificação de um vetor de dez elementos reais em ordem crescente.
Uma outra versão.

```
Algoritmo Bubble_Sort
Variáveis
        Numeros [10] : Real
        I, J : Inteiro
        Aux : Real
Início
```

```
        Para I = 1 Até 10 Faça
                Escreva "Informe o número ", I, " de 10"
                Leia Numeros[I]
        Fimpara
        Para I = 1 até 9 Faça
                Para J = I até (10-I) Faça
                        Se (Numeros[J] > Numeros[J+1]) então
                                Aux = Numeros[J]
                                Numeros[J] = Numeros[J+1]
                                Numeros[J+1] = Aux
                        FimSe
                Fimpara
        FimPara
        Escreva "Vetor Ordenado"
        Para I = 1 Até 10 Faça
                Escreva Numeros[I]
        Fimpara
Fim
```

A diferença básica desta versão para a anteriormente explicada está em sua estrutura de repetição, em que, agora, são usadas duas estruturas **PARA...FIMPARA** e, com isso, faz-se uso do recurso da *linguagem* eliminando algumas linhas de processamento. O resultado é o mesmo.

Veja a parte da figura que tem a diferença.

```
...
Algoritmo Bubble_Sort
Variáveis
        ...
Início
        ...
        Para I = 1 até 9 Faça
                Para J = I até (10-I) Faça
                        Se (Numeros[J] > Numeros[J+1]) então
                                Aux = Numeros[J]
                                Numeros[J] = Numeros[J+1]
                                Numeros[J+1] = Aux
                        FimSe
                Fimpara
        FimPara
        ...
Fim
```

5.3.2. MÉTODO DE PESQUISA SEQUENCIAL

Conhecida também como pesquisa linear. É o método mais objetivo para encontrar um elemento particular num conjunto não classificado, isto é, cujos

elementos não estão ordenados segundo algum critério.

Envolve a simples verificação de cada componente do conjunto, sequencialmente, até que o elemento desejado seja encontrado (neste caso, diz-se que a pesquisa foi **bem-sucedida**) ou que todos os elementos do conjunto tenham sido verificados sem que o elemento procurado tenha sido encontrado (pesquisa **mal-sucedida**).

Leitura de um nome e verificação se ele se encontra em um conjunto de 10 nomes, utilizando pesquisa sequencial.

```
Algoritmo PesquisaSequencial
Variáveis
        Valor[15], Nomes[10, 15] : String
        I : Inteiro
        Achou : Lógico
Início
        Para I = 1 Até 10 Faça
                Escreva "Informe o nome ", I, " de 10"
                Leia Nomes[I]
        Fimpara
        Escreva "Informe o nome que deseja procurar"
        Leia Valor
        I = 1
        Achou = .Falso.
        Enquanto (I <= 10) E (NÃO Achou) Faça
                Se (Nomes[I] = Valor) então
                        Achou = .Verdadeiro.
                Senão
                        I = I + 1
                FimSe
        FimEnquanto
        Se (Achou) então
                Escreva Valor, " foi encontrado."
        Senão
                Escreva Valor, " não foi encontrado."
        Fimse
Fim
```

Este exemplo de algoritmo solicita ao usuário que informe dez (10) nomes para preenchimento de uma matriz de nomes. Esta matriz será utilizada como fonte de pesquisa dos nomes que o usuário deseja procurar.

...

```
Algoritmo PesquisaSequencial
Variáveis
        ...
Início
```

```
        Para I = 1 Até 10 Faça
                Escreva "Informe o nome ", I, " de 10"
                Leia Nomes[I]
        Fimpara
        Escreva "Informe o nome que deseja procurar"
        Leia Valor
        I = 1
        Achou = .Falso.
        ...
Fim
```

A figura acima exibe a parte do algoritmo em que o usuário informará os dez (10) nomes que comporão a matriz, o nome que deseja procurar na matriz e duas variáveis tidas como auxiliares. A variável **I** é utilizada como contadora e terá a função de identificar qual nome da matriz está sendo utilizado para a comparação atual com o nome digitado pelo usuário. **Achou** terá a responsabilidade de informar se o nome digitado foi ou não foi encontrado na matriz. Note que **I** começa com um (1), pois a primeira comparação será com o primeiro nome e **Achou** começa com **.falso.**, pois devemos assumir que o nome pode não existir e, em caso de ser encontrado, aí sim, recebe um valor que reflita esta situação.

```
...

Algoritmo PesquisaSequencial
Variáveis
        ...
Início
        ...
        Enquanto (I <= 10) E (NÃO Achou) Faça
                Se (Nomes[I] = Valor) então
                        Achou = .Verdadeiro.
                Senão
                        I = I + 1
                FimSe
        FimEnquanto
        ...
Fim
```

Note na estrutura de repetição com teste no início, na figura acima, que existem duas condições que deverão manter a estrutura: a primeira reflete a leitura de toda a matriz; a segunda reflete a situação de que, se for encontrado o nome desejado, pode acabar a estrutura, mesmo que não se leia toda a matriz. O operador **E** garante a estrutura apenas quando as duas operações retornarem **verdadeiro**.

A estrutura condicional, dentro do **Enquanto...FimEnquanto**, trata duas situações:

- a de ter encontrado o nome, então se atribui **.Verdadeiro.** à variável *flag;*
- e a segunda, que, caso o nome digitado seja diferente do que o contador (**I**) aponta na matriz, este contador será incrementado em um (1) e, então, as condições da estrutura de repetição são novamente avaliadas.

```
...
Algoritmo PesquisaSequencial
Variáveis
        ...
Início
        ...
        Se (Achou) então
                Escreva Valor, " foi encontrado."
        Senão
                Escreva Valor, " não foi encontrado."
        Fimse
Fim
```

No momento em que a estrutura de repetição termina, é verificada a situação (valor) atual da variável **Achou**. Se ele retornar **.Verdadeiro.**, significa que o nome foi encontrado e, então, é exibida ao usuário esta confirmação. Caso seu valor continue sendo o inicialmente atribuído a ela, ou seja, **.Falso.**, é informado ao usuário que o nome não foi encontrado na matriz.

5.3.3. MÉTODO DE PESQUISA BINÁRIA

Quando os elementos de uma matriz estão classificados (ordenados), a pesquisa binária é um dos tipos de pesquisa mais eficazes para estes casos. Ela tem a capacidade de eliminar metade dos elementos da matriz que está sendo pesquisada, a cada comparação.

O método de pesquisa binária faz uso de três variáveis auxiliares:

- uma que aponte para o primeiro elemento da pesquisa; no caso de ser a **primeira** comparação é o primeiro elemento (valor 1).
- Outra que aponte para o **último** elemento da matriz de pesquisa; no caso de uma matriz de dez elementos, seria o valor 10.
- A terceira variável, que aponte para o elemento do **meio** da matriz; no exemplo de 10 elementos, seria o quinto elemento.

O elemento que é apontado pela variável **meio** é localizado e comparado com o valor procurado. Se ele for igual ao valor procurado, a pesquisa é dita bem-sucedida e é interrompida. No caso de ser ele maior que o valor procurado, repetimos o processo na primeira metade da matriz. No caso de o elemento central ser menor que o valor procurado, repetimos o processo na segunda metade da matriz. Vamos analisar a seguinte situação:

Leitura de um valor e verificação se este se encontra em um conjunto de 15 valores, utilizando pesquisa binária. Assuma que o conjunto seja informado de forma ordenada.

```
Algoritmo PesquisaBinaria
Variáveis
        I, Valor, Numeros[15], Meio, Alto, Baixo, : Inteiro
        Achou : Lógico
Início
        Para I = 1 Até 15 Faça
                Escreva "Informe o número ", I, " de 15"
                Leia Numeros[I]
        Fimpara
        Escreva "Informe o número que deseja procurar"
        Leia Valor
        Baixo = 1
        Alto = 15
        Achou = .Falso.
        Enquanto (Baixo <= Alto) E (NÃO Achou) Faça
                Médio = (Baixo + Alto) / 2
                Se (Valor < Numeros[Médio]) então
                        Alto = Médio - 1
                Senão se (Valor > Numeros[Médio]) então
                        Baixo = Médio + 1
                Senão Se (Valor = Números[Médio]) então
                        Achou = .Verdadeiro.
                FimSe
        FimEnquanto
        Se (Achou) então
                Escreva Valor, " foi encontrado."
        Senão
                Escreva Valor, " não foi encontrado."
        Fimse
Fim
```

O procedimento segue até que o elemento desejado seja localizado ou, então, até que não reste mais um trecho da matriz a ser pesquisada.

...

```
Algoritmo PesquisaBinaria
Variáveis
        Valor, Numeros[15], Meio, Alto, Baixo, : Inteiro
        Achou : Lógico
Início
        Para I = 1 Até 15 Faça
                Escreva "Informe o número ", I, " de 15"
                Leia Numeros[I]
        Fimpara
```

```
        Escreva "Informe o número que deseja procurar"
        Leia Valor
        Baixo = 1
        Alto = 15
        Achou = .Falso.
        ...
Fim
```

O procedimento inicial se assemelha ao utilizado na pesquisa sequencial: solicitamos os valores para preenchimento da matriz (neste caso, optado por números), pedimos ao usuário o valor que ele quer procurar entre os digitados. Observe que assumimos que os valores informados estão ordenados. O que diferencia este trecho de código é a inclusão de duas variáveis auxiliares, **Baixo** e **Alto**, que são responsáveis por manter a informação do primeiro e do último elemento do subconjunto para a pesquisa.

```
...
Algoritmo PesquisaBinaria
Variáveis
        ...
Início
        ...
        Enquanto (Baixo <= Alto) E (NÃO Achou) Faça
                Médio = (Baixo + Alto) / 2
                Se (Valor < Numeros[Médio]) então
                        Alto = Médio - 1
                Senão se (Valor > Numeros[Médio]) então
                        Baixo = Médio + 1
                Senão se (Valor = Números[Médio]) então
                        Achou = .Verdadeiro.
                FimSe
        FimEnquanto
        ...
Fim
```

Em relação às condições existentes na estrutura de repetição da pesquisa sequencial, houve aqui a alteração da primeira operação: podemos verificar se a variável que aponta para o primeiro elemento é menor ou igual à que aponta para o último elemento.

A primeira instrução identifica o número do elemento que será usado para apontar a metade da matriz. Veja que assumimos a soma do primeiro e do último e, então, dividimos esta soma por dois. No exemplo de 15 elementos e na primeira passada, obtemos a soma 16, que, ao ser dividido por dois, retorna 8. Em caso da divisão retornar um número com decimais, como a variável que receberá esta

divisão é **inteira**, o valor armazenado será a parte inteira do valor.

É feita a comparação entre o valor digitado pelo usuário e o valor apontado na matriz pela variável **Meio**. Caso o valor comparado seja maior que o valor digitado, a variável que aponta para o último elemento recebe o número do elemento central diminuído em 1, mudando assim o tamanho (quantidade) dos elementos a serem comparados na próxima execução da estrutura condicional. Caso o valor digitado seja maior que o valor comparado, a quantidade de elementos utilizados na próxima execução também diminui, mas agora a variável que sofre alteração é a que aponta para o primeiro elemento, que recebe o valor contido na variável, que aponta para o meio, somada de um. Para o caso de ter o valor comparado igual ao digitado, atribuimos **.Verdadeiro.** à variável *flag* **Achou** e então a estrutura termina.

```
...
Algoritmo PesquisaBinaria
Variáveis
        ...
Início
        ...
        Se (Achou) então
                Escreva Valor, " foi encontrado."
        Senão
                Escreva Valor, " não foi encontrado."
        Fimse
Fim
```

Igualmente ao apresentado no exemplo de pesquisa sequencial, quando a estrutura de repetição finda, é verificado o valor da variável *flag* **Achou** e, baseado em seu resultado, uma afirmação de sucesso ou fracasso na pesquisa.

As seguintes figuras mostram um exemplo com uma matriz de quinze (15) elementos e a pesquisa de alguns valores. O elemento sombreado mostra que é o elemento do meio, e cada linha representa a porção da matriz que será utilizada para a pesquisa.

Valor a pesquisar : **25**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	4	6	8	10	12	**14**	16	18	20	22	24	26	28
								16	18	20	**22**	24	26	28
												24	**26**	28
												24		

Neste exemplo, o valor não foi encontrado, mas é possível notar graficamente as porções da matriz que foram utilizadas a cada execução da estrutura de repetição. Veja outra situação.

Valor a pesquisar : **8**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	4	6	8	10	12	**14**	16	18	20	22	24	26	28
0	2	4	**6**	8	10	12								
				8	**10**	12								
				8										

Este exemplo já traz uma situação de êxito na pesquisa: o valor foi encontrado. Porém, podemos notar que foi feita uma pesquisa até que o último elemento fosse assumido como do meio.

Vamos a um último exemplo:

Valor a pesquisar : **6**

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	4	6	8	10	12	**14**	16	18	20	22	24	26	28
0	2	4	**6**	8	10	12								

Este último exemplo já obteve o resultado positivo na segunda execução da estrutura de repetição, ou seja, com apenas duas comparações.

PARA REFLETIR

A seguir, procure refletir um pouco sobre o que acabou de ler. Aproveite o momento para praticar os novos conhecimentos e realize as atividades propostas logo abaixo. Mãos à obra!

Chegamos ao final de mais uma unidade. Esperamos que agora tenha ficado mais claro seu entendimento sobre o estilo de armazenamento de dado trabalhado aqui, pois é um ótimo recurso para receber uma grande quantidade de dados e então trabalhar com eles em um momento posterior. Até agora, quando trabalhávamos com estrutura de repetição para receber estes dados, o processamento necessitava ser no mesmo momento, pois, ao término, não tinhamos mais estes dados. Mas agora, com matrizes e vetores, já é possível trabalhar estes dados em outra etapa do algoritmo.

O uso de matrizes para armazenamento de valores homogêneos teve, dentro dos textos vistos, mais um benefício identificado: a possibilidade de classificação e de pesquisa de valores. Dessa forma, a classificação pode ser feita em forma crescente ou decrescente para números, e alfabética para literais (o que não deixa de ser uma ordenação crescente).

Quanto à pesquisa de valores em uma matriz, foram vistas duas das formas mais usadas: a sequencial, que faz uma leitura em todos os elementos de uma matriz, não importando se ela está ou não classificada; e a pesquisa binária, que implica a necessidade de a matriz estar ordenada. A ordenação se fará sempre necessária quando for identificada uma exibição de valores de uma matriz por determinada ordem, por exemplo uma listagem de produtos por ordem alfabética. A pesquisa tem como exemplo a situação também ligada a produtos. Imagine uma matriz com código e estoque de um produto. Em uma venda você informa o código e quantidade vendida. Após isso, deve ajustar seu estoque. Para isto, precisará pesquisar em sua matriz o produto vendido.

Esta unidade tratou de temas importantes, um pouco mais complexos que os abordados nas unidades anteriores, exigindo leitura mais aprofundada e uma análise mais detalhada dos exemplos mostrados. Por isso, voltamos a convidá-lo a expandir seus conhecimentos a respeito desses assuntos por meio da pesquisa. Relembramos que a inquietação e a busca por novas informações é um grande passo para o sucesso em qualquer área da vida. Mãos à obra!

UNIDADE VI

SUBALGORITMOS E REGISTROS

Nesta nova unidade, abordaremos subalgoritmos e registros, que são noções importantes para:

- particionamento de um problema complexo em pequenos problemas simples;
- construir uma boa estrutura de dados identificados nos problemas, em que determinado dado, como um cadastro de clientes, pode passar a aceitar mais de um tipo de dado, como nome, idade e salário.

Portanto, no término desta etapa de estudo é fundamental que você seja capaz de

- dividir um algoritmo resolvido em pequenos subalgoritmos;
- identificar, criar e popular uma variável de um tipo específico de dado.

6.1. SUBALGORITMOS

A complexidade dos algoritmos está intimamente ligada à da aplicação a que se destinam. Em geral, problemas complicados exigem algoritmos extensos para sua solução.

É sempre possível dividir grandes algoritmos que resolvem grandes e complexos problemas em partes menores em que a solução e os problemas são particionados. Uma vez dividido um grande problema em pequenos problemas, é possível a resolução de cada um desses problemas individualmente, para que, ao final, o problema, em seu todo, tenha sido resolvido.

Subalgoritmos, subrotinas ou subprogramas são blocos de instruções que realizam tarefas específicas. O código de um subalgoritmo é digitado uma vez e pode ser executado quantas vezes forem necessárias. Dessa maneira, os programas tendem a ficar menores e mais organizados, dado que o problema pode ser dividido em pequenas tarefas.

Os programas (em programação estruturada), em geral, são executados linearmente, uma linha após a outra, até o fim. Entretanto, quando são utilizados subalgoritmos, é possível a realização de desvios na execução natural dos programas. Estes desvios são realizados quando uma função é chamada pelo programa principal.

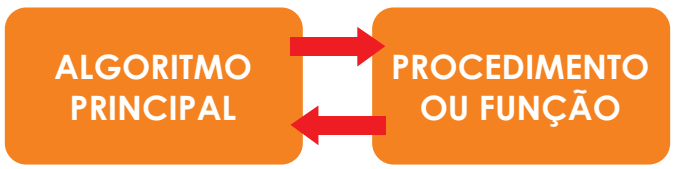
Fique atento!

Um *subalgoritmo* é um nome dado a um trecho de um algoritmo mais complexo e que, em geral, encerra em si próprio um pedaço da solução de um problema maior – o algoritmo a que ele está subordinado.

São importantes na

- subdivisão de algoritmos complexos, facilitando seu entendimento;
- estruturação de algoritmos, facilitando principalmente a detecção de erros e a documentação de sistemas; e na
- modularização de sistemas, que facilita a manutenção de programas e a reutilização de subalgoritmos já implementados.

Um algortimo completo é dividido em um **algoritmo principal** e diversos **subalgoritmos** (a quantidade que for necessária). O ***algoritmo principal*** é aquele por onde a execução do algoritmo sempre se inicia. Este pode invocar os demais subalgoritmos.



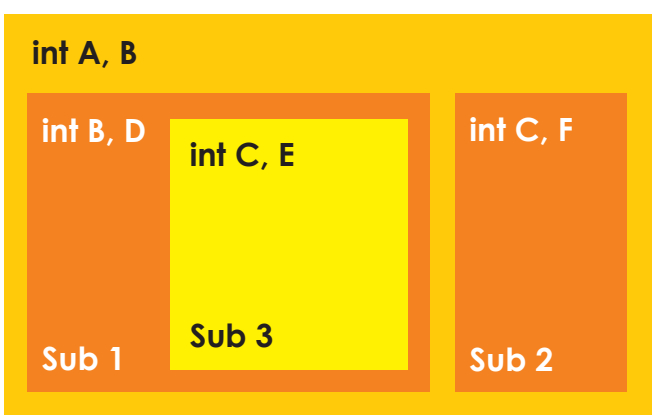
Durante a execução do algoritmo principal, quando encontramos um comando de invocação de um subalgoritmo, a execução daquele/deste é interrompida. A seguir, passamos à execução dos comandos do corpo do subalgoritmo. Quando ele termina, retomamos a execução do algoritmo que o chamou, o principal, no ponto onde foi interrompida (chamada do subalgoritmo) e prosseguimos pela instrução seguinte.

6.1.1. PROCEDIMENTOS E FUNÇÕES

Procedimento é um subalgoritmo que retorna nenhum ou mais valores ao algoritmo chamador. Estes valores são sempre retornados por meio dos parâmetros ou de variáveis globais, mas nunca explicitamente, como no caso de funções. Devido a isso, a chamada de um procedimento não pode surgir no meio de expressões, como no caso de funções. A chamada de procedimentos só é feita em comandos isolados dentro de um algoritmo.

Quanto à **função**, tem seu conceito, trazido da ideia de função matemática, em que um valor é calculado a partir de outro(s) fornecido(s) à função. Dentro de um algoritmo, a chamada a uma função sempre ocorre dentro de uma expressão do mesmo tipo que tem o valor retornado por ela. Sua chamada é feita pelo seu nome, seguido de seus respectivos parâmetros entre parênteses, dentro de uma expressão. A função é executada e, ao seu término, o trecho do comando que a invocou é substituído pelo valor retornado por ela dentro da expressão em que se encontra, e a avaliação da expressão prossegue normalmente.

6.1.2. VARIÁVEIS GLOBAIS E LOCAIS



Variáveis **globais** são aquelas declaradas no início de um algoritmo. Podem ser utilizadas no algoritmo principal e por todos os demais subalgoritmos.Variáveis **locais** são as definidas dentro de um subalgoritmo e, portanto, podem ser utilizadas apenas dentro dele. Outros subalgoritmos, ou até mesmo o principal, não podem utilizá-la.

Observe na figura anterior a existência de três subalgoritmos (*Sub1*, *Sub2* e *Sub3*), todos dentro de um algoritmo maior (o principal). Conceitualmente, os algoritmos

criados "dentro" de um algoritmo principal têm acesso às variáveis criadas neste principal. No entanto, o principal não tem conhecimento das variáveis criadas em seus subalgoritmos.

> Exemplificando, podemos dizer que o subalgoritmo **Sub** tem acesso às variáveis **D** (do **Sub1**) e **A** (do **principal**). Em relação à variável **B** do **Sub1**, sobrepõe, mas não elimina, a variável **B** do **principal.**

É necessária a compreensão conceitual de variáveis globais e locais, porém saber qual a forma de declaração que determinará sua visibilidade dentro de subalgoritmos dependerá exclusivamente da linguagem de implementação. Em relação a pseudocódigos, isto está mais ligado a convenções.

6.1.3. PARÂMETROS

São canais pelos quais se estabelece uma comunicação bidirecional entre um subalgoritmo e o algoritmo chamador. Dados são passados pelo algoritmo chamador ao subalgoritmo. Para o caso de procedimentos, dados também pode ser retornados para o algoritmo chamador através dos parâmetros.

Para o caso de funções, a passagem de dados se dá da mesma forma que para procedimentos, todavia, para o retorno dos dados, é utilizado o comando **Retorne** *<expressão>* para retornar o valor calculado (ou processado) por ela. Ao encontrar este comando, a expressão entre parênteses é avaliada, a execução da função é terminada neste ponto, e o valor da expressão é retornado ao algoritmo chamador. Esta expressão deve ser do mesmo tipo que o valor retornado pela função.

Ao envio de dados de um algoritmo para outro através de sua chamada, é dado o nome de passagem de parâmetros e esta passagem pode ser dada de duas formas: **por valor** ou **por referência**. Observe a definição de cada uma delas.

Na passagem de parâmetros por valor, o que ocorre é uma cópia do conteúdo enviado para uma variável do mesmo tipo daquela que foi enviada, que existirá apenas no procedimento/função. Com isso, processamentos, avaliações e atribuições podem ser feitos a esta variável/valor recebido, sem que haja qualquer tipo de alteração na variável que se encontra no algoritmo chamador.

Na passagem de parâmetros por referência não existe a cópia do conteúdo enviado para uma variável do procedimento/função, o que ocorre é o envio da referência onde a variável do algoritmo chamador se encontra. Dessa forma, qualquer alteração feita na variável no subalgoritmo refletirá diretamente na variável do algoritmo chamador.

6.1.4. PRÁT ICA

Na sequência, são apresentados algoritmos que trabalharão subalgoritmos em

suas resoluções. Os conceitos de código e regras de implementação serão explicados a cada solução proposta.

Dados o nome e as médias de uma turma com 47 alunos, faça um algoritmo que escreva o nome dos alunos que tiveram a média menor que a média da turma. Para desenvolver este problema você deverá :

- Criar um procedimento para a leitura dos dados e armazenamento em uma matriz declarada como variável global. Neste mesmo procedimento, efetue o cálculo da média aritmética da turma, que também deverá estar armazenada em uma variável global.

- Criar um procedimento para identificação e exibição dos alunos que tenham a média menor que a média da turma.

```
Algoritmo Alunos
Variáveis
        Nomes[47][15] : String
        Medias[47], MediaTurma : Real

Procedimento EntradaDeDados
Variáveis
        I : Inteiro
Início
        Escreva "Informe o nome e média dos 47 alunos da turma"
        MediaTurma = 0
        Para I = 1 Até 47 Faça
                Escreva "Informe o nome do aluno ", I, " de 47"
                Leia Nomes[I]
                Escreva "Informe a média do aluno ", Nomes[I]
                Leia Medias[I]
                MediaTurma = MediaTurma + Medias[I]
        FimPara
        MediaTurma = MediaTurma / 47
Fim

Procedimento SaidaDeDados
Variáveis
        I : Inteiro
Início
        Escreva "Alunos com média abaixo da média da turma"
        Para I = 1 Até 47 Faça
```

```
                Se (Medias[I] < MediaTurma) Então
                        Escreva "Aluno ", Nomes[I], " média ", Notas[I]
                FimSe
        FimPara
Fim

Início
        EntradaDeDados
        SaidaDeDados
Fim
```

No algoritmo acima, é tratada a situação em que é solicitado ao usuário que informe nome e nota de 47 alunos de uma turma e que entre estes alunos, seja informado o nome e média daqueles que tiraram sua média individual inferior à média da turma.

Até onde vínhamos resolvendo nossos problemas, isso seria facilmente resolvido em um algoritmo simples. Tirando os procedimentos identificados nesta resolução e colocando as instruções na sequência apresentada, teríamos um algoritmo completo, que também resolveria o problema.

Imagine a situação em que, como no exemplo, um único algoritmo seja responsável pela entrada e saída de dados, em que, porém, estas operações não ocorressem obrigatoriamente no mesmo momento. É claro que ainda não armazenamos nossos dados em arquivos, ou seja, tão logo o algoritmo termine sua execução, os dados se perdem, mas façamos de conta que os dados estão guardados em arquivos. Sendo assim, nada impede que, em determinado momento, seja feita a entrada de dados e, em outro momento, a saída. A operação a ser executada poderia ser escolhida pelo usuário através de um menu de opções.

Neste caso, a divisão em procedimentos ou funções é mais apropriada, pois se dividem as responsabilidades de um único problema e se trabalha em cada uma como se fosse um único. Talvez esteja pensando "*Mas eu poderia desenvolver este algoritmo usando as estruturas condicionais* ***Se*** *e* ***Escolha****, daria o mesmo resultado...*". Com certeza, para este problema sim, mas este tipo de problema não aparecerá em sua vida profissional, os desafios serão maiores e você deverá estar preparado para quando isso ocorrer.

Saiba que o objetivo de criar procedimentos e funções é que rotinas comuns a vários problemas possam ser desenvolvidas uma única vez e executadas sempre que necessário. Veremos, no passo seguinte, uma variação desta resolução que dará um poder maior ao usuário e pouco será alterado em seu código para isso. Vejamos agora partes isoladas desta resolução e explicações que dêem suporte para sua compreensão.

```
...

Algoritmo Alunos
Variáveis
        Nomes[47][15] : String
        Medias[47], MediaTurma : Real

Procedimento EntradaDeDados
Início
        ...
Fim

Procedimento SaidaDeDados
Início
        ...
Fim

Início
        ...
Fim
```

As variáveis **Nomes** e **Medias** estão declaradas dentro da implementação do **Algoritmo**, ou seja, do programa principal. Dessa forma, ela é tida como global e pode ser utilizada por qualquer procedimento e função que esteja neste código, além do próprio algoritmo principal.

```
...

Algoritmo Alunos
Variáveis
        ...

Procedimento EntradaDeDados
Variáveis
        I : Inteiro
Início
        Escreva "Informe o nome e média dos 47 alunos da turma"
        MediaTurma = 0
        Para I = 1 Até 47 Faça
                Escreva "Informe o nome do aluno ", I, " de 47"
                Leia Nomes[I]
                Escreva "Informe a média do aluno ", Nomes[I]
                Leia Medias[I]
                MediaTurma = MediaTurma + Medias[I]
        FimPara
        MediaTurma = MediaTurma / 47
Fim

Procedimento SaidaDeDados
```

```
Início
        ...
Fim

Início
        ...
Fim
```

Até onde vínhamos trabalhando na resolução de problemas, tínhamos, logo após a declaração das variáveis, o bloco **Início...Fim**, que trabalha a resolução do problema, porém, como agora trabalharemos estas resoluções mediante o uso de procedimentos e funções, há a necessidade de declaração destes, antes do bloco **Início...Fim**. Note que cada procedimento tem o corpo idêntico ao de um algoritmo completo, com seu nome, declaração de variáveis e resolução. Apenas deve ser sabido que as variáveis declaradas dentro de um procedimento ou função têm seu acesso permitido apenas dentro do procedimento ou função que a declara.

A resolução deste procedimento é simples e semelhante a algumas soluções já implementadas por nós em situações anteriores. Utilizamos uma estrutura de repetição contada para entrada dos dados, tendo uma variável acumuladora inicializada em **zero** antes do início desta e, ao seu término, é feita a operação de divisão para obtenção da média aritmética.

```
...

Algoritmo Alunos
        ...
Procedimento EntradaDeDados
Início
        ...
Fim

Procedimento SaidaDeDados
Variáveis
        I : Inteiro
Início
        Escreva "Alunos com média abaixo da média da turma"
        Para I = 1 Até 47 Faça
                Se (Medias[I] < MediaTurma) Então
                        Escreva "Aluno ", Nomes[I], " média ", Notas[I]
                FimSe
        FimPara
Fim

Início
        ...
Fim
```

A resolução do procedimento acima também não é de outro mundo. Apenas é necessário saber que a entrada de dados deve realizar esta execução e o programador deverá prover ao código este conhecimento, como veremos em passagem posterior.

```
...

Algoritmo Alunos
        ...

Procedimento EntradaDeDados
Início
        ...
Fim

Procedimento SaidaDeDados
Início
        ...
Fim

Início
        EntradaDeDados
        SaidaDeDados
Fim
```

O bloco **Início...Fim** do código anterior representa a execução do algoritmo principal. Veja que são apenas chamados os procedimentos já codificados.

Bem, vamos verificar uma variante deste problema, que, seguramente, justificará o uso de procedimentos e funções. Imagine que você tenha vários clientes que irão fazer o lançamento de notas e nomes de seus alunos. É óbvio que cada um terá turmas com números diferenciados de alunos. Você alteraria seu programa para cada cliente? Isso seria inviável, porém podemos pensar em deixar sua solução apta para atender a todos eles, desde que a única mudança necessária seja a quantidade de alunos existentes em cada turma. Veja a nova resolução como ficaria.

```
...

Algoritmo Alunos
Variáveis
        Nomes[47][15] : String
        Medias[47], MediaTurma : Real

Procedimento EntradaDeDados(QtdeAlunos : Inteiro)
Variáveis
        I : Inteiro
Início
        Escreva "Informe o nome e média dos ", QtdeAlunos, " alunos da
```

```
        turma"
        MediaTurma = 0
        Para I = 1 Até QtdeAlunos Faça
                Escreva "Informe o nome do aluno ", I, " de ", QtdeAlunos
                Leia Nomes[I]
                Escreva "Informe a média do aluno ", Nomes[I]
                Leia Medias[I]
                MediaTurma = MediaTurma + Medias[I]
        FimPara
        MediaTurma = MediaTurma / QtdeAlunos
Fim

Procedimento SaidaDeDados(QtdeAlunos : Inteiro)
Variáveis
        I : Inteiro
Início
        Escreva "Alunos com média abaixo da média da turma"
        Para I = 1 Até QtdeAlunos Faça
                Se (Medias[I] < MediaTurma) Então
                        Escreva "Aluno ", Nomes[I], " média ", Notas[I]
                FimSe
        FimPara
Fim

Início
        EntradaDeDados(10)
        SaidaDeDados(10)
Fim
```

A resolução acima, permite a execução completa e com sucesso para qualquer quantidade de alunos de uma turma, desde que esta quantidade não ultrapasse o máximo permitido na matriz, ou seja, o seu tamanho físico, que ainda ficou em 47, como pode ser visto no início do algoritmo. Note as alterações feitas nas linhas que estão em itálico.

Observe que ainda não foi tratada a verificação de o lançamento das notas ter realmente ocorrido antes da solicitação dos alunos com média inferior à média da turma. Podemos ter uma resolução mais próxima da necessidade com a exibida a seguir.

```
...

Algoritmo Alunos

Variáveis
        Nomes[47][15] : String
        Medias[47], MediaTurma : Real
        TF : Inteiro
```

```
Procedimento EntradaDeDados(QtdeAlunos : Inteiro)
Variáveis
        I : Inteiro
Início
        Se (QtdeAlunos > TF) Então
                Escreva "A quantidade informada é superior a permitida"
                Retorne
        FimSe

        Escreva "Informe o nome e média dos ", QtdeAlunos, " alunos da
        turma"
        MediaTurma = 0
        Para I = 1 Até QtdeAlunos Faça
                Escreva "Informe o nome do aluno ", I, " de ", QtdeAlunos
                Leia Nomes[I]
                Escreva "Informe a média do aluno ", Nomes[I]
                Leia Medias[I]
                MediaTurma = MediaTurma + Medias[I]
        FimPara
        MediaTurma = MediaTurma / QtdeAlunos
Fim

Procedimento SaidaDeDados(QtdeAlunos : Inteiro)
Variáveis
        I : Inteiro
Início
        Se (QtdeAlunos > TF) Então
                Escreva "A quantidade informada é superior a permitida"
                Retorne
        FimSe

        Escreva "Alunos com média abaixo da média da turma"
        Para I = 1 Até QtdeAlunos Faça
                Se (Medias[I] < MediaTurma) Então
                        Escreva "Aluno ", Nomes[I], " média ", Notas[I]
                FimSe
        FimPara
Fim

Início
        TF = 47
        EntradaDeDados(10)
        SaidaDeDados(10)
Fim
```

Na resolução anterior, foi declarada uma variável (TF), que tem como responsabilidade informar a quantidade máxima (o tamanho físico) das matrizes que receberão os dados informados pelo usuário. O incoveniente é que esta informação deve ser feita sempre no algoritmo principal, e a variável deverá ter o nome TF sempre, pois é assim que é utilizada nos procedimentos. Isso, para o caso de outros algoritmos fazerem uso destes procedimentos.

Note a verificação de limite no início de cada procedimento e, em caso de limite excedido, a instrução Retorne faz com que o procedimento deixe de ser executado e retorne para o procedimento que o chamou, neste caso o algoritmo principal.

PARA REFLETIR

Ficaram claros para você os conceitos relacionados com os subalgaritmos descritos neste início da Unidade VI?

6.2. REGISTROS

São estruturas que podem agregar diferentes informações. Dessa maneira, podem ser feitas diferentes combinações, gerando outros tipos de dados.

Importante!
Um registro é uma coleção de campos, em que cada campo pode ser de um tipo de dado diferente. Por isso, os registros são conhecidos como variáveis compostas heterogêneas.

Um registro consiste em um certo número de itens de dados, chamados membros da estrutura, que não necessitam ser do mesmo tipo, agrupados juntos, como mostra a figura a seguir.

DADOS DE FUNCIONÁRIOS		
Código : 09182	**Nome :** Hermenegildo Florentil	**Sexo :** Masculino
Endereço : Rua dos Registros Isolados, 736		
Cargo : Chefe de Divisória	**Salário :** $ 455,46	

Há situações em que deparamos com um problema de programação, quando se deseja agrupar um conjunto de tipos de dados não similares sob um único nome. O primeiro impulso seria, talvez, usar uma matriz (conjunto). Como matrizes requerem que todos os seus elementos sejam do mesmo tipo, provavelmente solucionará o problema selecionando uma matriz para cada tipo de dado. O resultado tornaria o programa ineficaz na maneira de manejar os dados.

O exemplo tradicional de uso de registro é uma folha de pagamento: um funcionário é descrito por um conjunto de atributos tais como nome (uma ***string)***, o número do seu departamento (um ***inteiro***), salário (um ***float***), e assim por diante. Provavelmente, haverá outros funcionários, e você vai querer que seu programa os guarde, formando uma matriz de registro.

6.2.1. PRÁTICA

Segue agora apresentação de algoritmos que trabalharão estrutura heterogênea de dados em suas resoluções e, seguindo a estrutura já trabalhada nas unidades anteriores, os conceitos de código e regras de implementação serão explicados em cada solução proposta.

Faça um algoritmo que solicite ao usuário o preço de compra e o preço de venda de um único produto e informe ao usuário:

- O valor do lucro obtido, caso exista;
- Caso o lucro seja negativo, deverá ser informado ao usuário que ele obteve prejuízo na venda.

```
Algoritmo VerLucro
Variáveis
        Produto : Registro(Custo, Venda, Lucro : Real)
Início
        Escreva "Informe o preço de compra do produto"
        Leia Produto.Custo
        Escreva "Informe o preço de venda do produto"
        Leia Produto.Venda
        Produto.Lucro = (Produto.Venda – Produto.Custo)
        Se (Produto.Lucro > 0) então
                Escreva "A venda obteve um lucro de ", Produto.Lucro
        Senão Se (Produto.Lucro < 0) então
                Escreva "A venda obteve um prejuízo de ", Produto.Lucro
        Senão
                Escreva "Não houve lucro nem prejuízo na venda"
        FimSe
Fim
```

Antes de tentarmos resolver o enunciado, é interessante a visualização do registro para sua devida compreensão. Dessa forma, vejamos a representação:

DADOS DOS PRODUTOS	
Preço de Compra (*custo*) :	Preço de Venda :
Lucro Obtido :	

Observe que, na montagem do registro, não existem valores para os ***campos***

existentes, ou seja, estes valores deverão ser preenchidos pelo usuário. Para facilitar a compreensão, podemos imaginar que serão entregues fichas aos usuários, as quais serão preenchidas e então, depois disso, os dados serão informados ao algoritmo.

Ainda em relação à representação do registro, o campo **Lucro Obtido**, poderia não fazer parte dele, pois pode ser obtido este valor por meio de uma operação de subtração entre os dois outros campos (**Venda** e **Compra**).

Bem, por ser o primeiro exemplo a utilizar registros, no algoritmo acima é resolvido um simples problema de subtração e identificação de possibilidades. Note a seguir a declaração de uma variável que seja do tipo registro.

```
...
Algoritmo VerLucro
Variáveis
        Produto : Registro(Custo, Venda, Lucro : Real)
Início .
        ...
Fim
```

O acesso às variáveis (campos ou membros) do registro é feito mediante a separação do nome da variável de registro e nome do campo por um ponto. A forma de exibição e entrada de dados segue o visto em todos os exemplos anteriores. Verifique o código a seguir.

```
...
Algoritmo VerLucro
        ...
Início
        Escreva "Informe o preço de compra do produto"
        Leia Produto.Custo
        Escreva "Informe o preço de venda do produto"
        Leia Produto.Venda
        ...
Fim
```

Dado que o enunciado solicita, como resultado final do processamento, apenas o valor do lucro e já deu a dica de que, para ser obtido, ele necessita do preço de venda e de compra, após a informação do valor é feita uma operação de subtração e armazenado seu valor em um campo do registro.

```
...
Algoritmo VerLucro
        ...
Início
```

```
        Escreva "Informe o preço de compra do produto"
        Leia Produto.Custo
        Escreva "Informe o preço de venda do produto"
        Leia Produto.Venda
        ...
Fim
```

Observe que, no trecho acima, é verificado, além da obtenção de lucro ou prejuízo, a situação em que não houve nem um nem outro. Ou seja, o produto pode ter sido vendido pelo mesmo preço que foi comprado. Verifique que isso não estava claro no enunciado, mas existem situações que exigem que nosso raciocínio veja além do que está escrito, com bastante controle, é claro.

Dado que, para cada aluno de uma turma de N alunos se tenha: o seu número de registro, seu nome e sua média final, faça um algoritmo que:

- Imprima a média da turma;
- Calcule a percentagem de alunos, cujos nomes comecem pela letra A;
- Determine quantos alunos têm a média superior a 7,0.
- Você deverá fazer uso de registros.

```
Algoritmo Boletim
Variáveis
        Alunos[100] : Registro(Numero : Inteiro, Nome : String, Media : Real)
        QtdeAlunos, QtdeNomesA, QtdeAlunos7 : Inteiro
        MediaTurma, PercentualNomesA : Real
Início
        Repita
                Escreva "Informe a quantidade de alunos"
                Leia QtdeAlunos
        Até (QtdeAlunos > 0 E QtdeAlunos <= 100)
        MediaTurma = 0
        QtdeNomesA = 0
        QtdeAlunos7 = 0
        Para I = 1 Até QtdeAlunos Faça
                Escreva "Informe os dados do aluno ", I, " de ", QtdeAlunos
                Escreva "Número : "
                Leia Alunos[I].Numero
                Escreva "Nome : "
                Leia Alunos[I].Nome
                Escreva "Média Final : "
                Leia Alunos[I].Media
                MediaTurma = (MediaTurma + Alunos[I].Media)
                Se (Alunos[I].Nome[1] = 'A') então
                        QtdeNomesA = (QtdeNomesA + 1)
                FimSe
```

```
                Se (Alunos[I].Media > 7.0) então
                        QtdeAlunos7 = (QtdeAlunos7 + 1)
                FimSe
        FimPara
        MediaTurma = (MediaTurma / QtdeAlunos)
        MediaNomesA = (QtdeNomesA * 100) / QtdeAlunos
        Escreva "A média da turma é : ", MediaTurma
        Escreva  "O percentual de alunos com nomes começando por A é ",
                PercentualNomesA
        Escreva "Existem ", QtdeAlunos, " com média acima de 7.0"
Fim
```

Diferentemente do exemplo anterior a este, em que se solicitavam dados de apenas uma amostra (um produto), este solicita vários dados, mas todos com a mesma estrutura. Dessa forma, podemos imaginar que o usuário responsável pela alimentação de dados no algoritmo receba uma folha semelhante à exibida a seguir:

Relação de Notas da Turma 07/2003		
Número	**Nome**	**Média**
1	Simpoziando Silverado	8,0
2	Astranagildo Herpinóide	5,0
3	Josmefildiano Hiperbáculo	9,0
4	Marinaltina Destraliana	4,0
5	Josefalinilda Matricial	7,0

O enunciado deste problema contempla uma situação um pouco mais complexa, pois envolve, além de registros, estruturas de repetição, estrutura de condição e conjunto, bem como de um pequeno processamento sobre os dados de entrada.

Como visto em outra passagem, temos a necessidade, na declaração de um conjunto, de estabelecer sua capacidade (*tamanho físico*), ou seja, quantas linhas ele suportará. Além disso, o algoritmo deverá garantir que este limite não seja ultrapassado.

```
...

Algoritmo Boletim
Variáveis
        Alunos[100] : Registro(Numero : Inteiro, Nome : String, Media : Real)
        ...
Início
        Repita
                Escreva "Informe a quantidade de alunos"
```

```
            Leia QtdeAlunos
        Até (QtdeAlunos > 0 E QtdeAlunos <= 100)
        ...
Fim
```

Veja acima que o conjunto **Alunos** tem seu tamanho físico estipulado em 100, e que ele é um **Registro**. Mais abaixo, uma estrutura de repetição que terminará apenas quando o usuário informar um valor compreendido entre 1 e 100, ou seja, o algoritmo garante que o tamanho lógico (linhas realmente usadas do conjunto) não ultrapasse o tamanho físico, e que também não seja informado zero ou um valor negativo.

Uma vez obtida a quantidade de vezes que a estrutura de repetição deverá gerenciar (no exemplo a quantidade de alunos), existem algumas variáveis que têm a função de serem ***acumuladoras.*** Sendo assim, devem ser inicializadas com zero antes da estrutura começar.

```
...
Algoritmo Boletim
...
Início
        ...
        MediaTurma = 0
        QtdeNomesA = 0
        QtdeAlunos7 = 0
        Para I := 1 Até QtdeAlunos Faça
                Escreva "Informe os dados do aluno ", I, " de ", QtdeAlunos
                Escreva "Número : "
                Leia Alunos[I].Numero
                Escreva "Nome : "
                Leia Alunos[I].Nome
                Escreva "Média Final : "
                Leia Alunos[I].Media
                ...
        FimPara
        ...
Fim
```

Verifique no código acima a declaração das variáveis acumuladoras e a entrada dos dados necessários ao processamento. Observe que a manipulação de dados por meio de conjuntos de registro não difere do visto até o momento. Há, sim, novo acréscimo, que é o campo do registro onde o dado será armazenado e este campo é separado do nome da variável de registro, por um ponto (**.**).

Uma vez obtidos os dados necessários para o processamento, este deve ser executado.

...

```
Algoritmo Boletim
        ...
Início
        ...
        Para I := 1 Até QtdeAlunos Faça
                ...
                MediaTurma = (MediaTurma + Alunos[I].Media)
                Se (Alunos[I].Nome[1] = 'A') então
                        QtdeNomesA = (QtdeNomesA + 1)
                FimSe
                Se (Alunos[I].Media > 7.0) então
                        QtdeAlunos7 = (QtdeAlunos7 + 1)
                FimSe
        FimPara
        ...
Fim
```

Observe acima que o que lhe pode parecer diferente é a verificação dos alunos que têm seu nome iniciado pela letra A. Pense que uma **string** é um conjunto de caracteres e como tal, podemos acessar seus valores mediante índices, e é isso que é feito.

Terminado o processamento necessário a cada aluno, que deve ser feito dentro da estrutura de repetição, há necessidade de identificar os valores que dependem do término da estrutura.

...

```
Algoritmo VerLucro
        ...
Início
        ...
        Para I := 1 Até QtdeAlunos Faça
                ...
        FimPara
        MediaTurma = (MediaTurma / QtdeAlunos)
        PercentualNomesA = (QtdeNomesA * 100) / QtdeAlunos
        Escreva "A média da turma é : ", MediaTurma
        Escreva  "O percentual de alunos com nomes começando por A é ",
                PercentualNomesA
        Escreva "Existem ", QtdeAlunos7, " com média acima de 7.0"
Fim
```

Após o término da estrutura de repetição, as variáveis acumuladoras terão os valores referentes a todos os alunos, necessários para a identificação da média e do percentual desejado. Sendo assim, as operações são efetuadas, e os valores exibidos ao usuário.

Parabéns! Chegamos ao final desta última unidade!

Nesta unidade, estudamos um pouco sobre os subalgoritmos e os registros.

Foi bastante destacado que o uso de subalgoritmos para resolução de problemas é altamente recomendado, pois, mediante este recurso, podemos particionar um grande problema em problemas menores e tratá-los individualmente, como partes completas. Isso permite, além da modularização de código, uma reutilização deste. O envio e o recebimento de argumentos e retorno de valores processados devem ser sempre utilizados, lembrando que os argumentos de entrada de procedimentos e funções são os dados necessários para que resolvam o problema que se propõem, e o retorno é aquele que se espera obter com o processamento dos argumentos de entrada. As linguagens atuais estão voltando seu uso mais para funções (que em linguagens orientadas a objetos são tratadas como métodos), pois o que diferencia um procedimento de uma função é o retorno de valores.

Como pôde ser visto também na segunda parte desta unidade, o uso de registros permite que dados comuns a um objeto, pessoa ou acontecimento, podem ser agrupados em uma única variável, mesmo sendo de tipos diferentes, o que já não ocorre em conjuntos.

O uso de registros na resolução de algoritmos complexos é essencial, devendo ser verificado, em cada linguagem, como as mesmas implementam este recurso.

Encerramos por aqui, sempre acreditando que o que foi estudado tenha sido de grande utilidade pra você nesta jornada de aprendizado, que deve ser constante.

Parabéns por chegar até aqui!!!

RETOMANDO A CONVERSA INICIAL

Parabéns, você cumpriu mais este objetivo em sua vida. Após esta leitura e o estudo decorrente dela, espero que você já tenha condições de desenvolver seus conhecimentos sobre algoritmos.

Espero que a dinâmica utilizada na organização do conteúdo tenha sido boa e facilitadora para seu aprendizado, e que agora você possa compreender algoritmos e sua importância na formação do técnico em Informática.

Você viu, na Unidade I, os conceitos de lógica, algoritmo, dados, informação, processamento de dados e lógica de programação, o que certamente deu a você subsídios para compreensão do que o esperava nas próximas unidades.

A Unidade II apresentou os tipos de dados, variáveis e expressões, que certamente você utilizou durante todo o estudo deste livro.

Na Unidade III, você pôs a mão na massa, começou a "programar", fazendo uso de pseudocódigos. Terminou a unidade conhecendo e resolvendo problemas em sua estrutura sequencial e condicional.

A Unidade IV trabalhou especificamente as três variações possíveis para uma estrutura de repetição, e você certamente já sabe os benefícios de uma em relação às outras, o que o auxilia na escolha da correta estrutura para a resolução de um problema.

A Unidade V respondeu a uma pergunta que você seguramente vinha fazendo: "Como trabalhar com um conjunto de dados?". Esta resposta foi dada por meio do uso de conjuntos, trabalhados nesta unidade.

A finalização de seu estudo, com chave de ouro, se deu na Unidade VI. Aí você foi apresentado aos subalgoritmos e aos registros, poderosos recursos que certamente farão parte de seu dia a dia de programador.

Agora, meu amigo, descanse (um pouquinho só, hein!) e se prepare para o próximo módulo de seu curso, quando seus conhecimentos de lógica serão usados em linguagens reais de programação.

Um grande abraço e muito sucesso.

Everton Coimbra de Araújo

REFERÊNCIAS

ARAÚJO, Everton Coimbra de. ***Algoritmos - Fundamento e Prática.*** 3ª ed. Florianópolis, Visual Books, 2007

ASCENSIO, Ana Fernanda Gomes e CAMPOS, Edilene Aparecida Veneruchi. ***Fundamentos da Programação de Computadores – Algoritmos, Pascal e C/C++***. Porto Alegre, Prentice Hall, 2002.

FILHO, Edgard de Alencar. ***Iniciação à Lógica Matemática***. 18ª ed. São Paulo, Nobel, 2000.

FORBELLONE, André Luiz Villar; EBERSPACHER, Henri Frederico. ***Lógica de Programação*: A construção de algoritmos e Estrutura de Dados**. 2 ed. São Paulo, Makron Books, 2000.

LOPES, Anita; GARCIA, Guto. ***Introdução à Programação*: 500 Algoritmos resolvidos**. Rio de Janeiro, Campus, 2002.

SALIBA, Walter Luiz Caram. ***Técnicas de Programação:* Uma abordagem estruturada**. São Paulo, Makron Books, 1993.

SALVETTI, Dirceu Douglas; BARBOSA, Lisbete Madsen. ***Algoritmos***. São Paulo, Makron Books, 1998.

Autor

EVERTON COIMBRA DE ARAÚJO, desde 1987, atua na área de treinamento e desenvolvimento. É desenvolvedor, analista de sistemas e gerente de projetos, pela PPT INFORMÁTICA, desde 1994.

É autor pela editora Visual Books, tendo publicado nesta editora livros sobre Algoritmos, Delphi, C++ Builder, Orientação a Objetos e Java.

Como Mestre em Ciência da Computação, é professor efetivo da Universidade Tecnológica Federal do Paraná (UTFPR), câmpus Medianeira, onde leciona disciplinas no curso tecnológico de Análise e Desenvolvimento de Sistemas e no curso de Pós-Graduação em Desenvolvimento de Sistemas para Web com Java. Também tem atuado pela UTFPR nos cursos de EaD (Educação a Distância), oferecidos pela UAB e ETEC.

Já ministrou aulas em outras faculdades nas disciplinas de Algoritmos, Técnicas de Programação, Estrutura de Dados, Linguagens de Programação (estruturadas e orientadas a objetos, tais como C, Pascal, Delphi, C++ Builder, Java e C#), Análise de Sistemas, UML e Banco de Dados.

Nos últimos anos tem se dedicado às disciplinas relacionadas com o desenvolvimento de aplicações web e na persistência de objetos, focando seus estudos e pesquisas na plataforma Java (JSP, Servlets, JSF e EJB), .NET (ASP.NET) e AJAX.

Tem como objetivo concluir uma coletânea de livros que cubram todas as etapas, teóricas e práticas, referentes ao desenvolvimento de um projeto de sistemas, usando como alicerce todas as disciplinas que compõem diversos cursos de nível superior ligados à informática.

O autor também tem proferido palestras em seminários de informática, voltados tanto para o meio acadêmico como para o empresarial.