Aula 1º

Lógica

Prezados alunos, iniciaremos a disciplina "Algoritmos" adquirindo algumas noções básicas de Lógica. A construção desse conhecimento será fundamental para entendermos o raciocínio utilizado na solução dos problemas cotidianos, sejam eles pessoais, acadêmicos, profissionais ou outros.

Para iniciar, é importante sabermos que a Lógica foi uma ciência criada pelo filósofo grego Aristóteles, no século IV a.C., para entender o pensamento humano e distinguir os argumentos certos dos errados.

Ah, esta Aula foi preparada para que você não encontre grandes dificuldades. Contudo, podem surgir dúvidas no decorrer dos estudos! Quando isso acontecer, acesse a plataforma e utilize as ferramentas "quadro de avisos" ou "fórum" para interagir com seus colegas de curso ou com seu professor. Sua participação é muito importante e estamos preparados para ensinar e aprender com seus avanços...

Lembre-se ainda de ler e refletir sobre os objetivos de aprendizagem e as seções de estudo da Aula 1. Afinal, você é o protagonista de sua aprendizagem!

Bons estudos!

Objetivos de aprendizagem

Esperamos que, ao término desta aula, você seja capaz de:

- definir Lógica, reconhecer sua importância e aplicá-la na análise de sistemas;
- conceituar e introduzir o termo Algoritmo;
- desenvolver e/ou ampliar a capacidade de percepção de um problema e utilizá-la na criação de Algoritmos.

Seções de estudo

1 - Lógica: conceituação e aplicação na análise de sistemas
2 - Algoritmo: definição e criação por meio da percepção de um problema

1 - Lógica: conceituação e aplicação na análise de sistemas

Para iniciar nossas reflexões sobre os Algoritmos, nesta Seção vamos compreender brevemente o conceito de Lógica, reconhecer sua importância, além de construirmos conhecimentos sobre suas aplicações no contexto de análise de sistemas.
Ah, durante a leitura desta e das demais Aulas é importante que tenha sempre à mão um dicionário e/ou outros materiais de pesquisa para eliminação de eventuais dúvidas pontuais sobre os termos aqui empregados.

CONCEITO

Lógica é a ciência do raciocínio e da demonstração: é o conjunto de leis, princípios ou métodos que determinam um raciocínio coerente, induzindo a uma solução prática e eficaz de um problema.

A Lógica também pode ser conceituada como uma sequência coerente, regular e necessária de acontecimentos, de coisas ou fatos, ou até mesmo, como a maneira particular do raciocínio de cada pessoa ou de um grupo.

Você já entendeu amplamente o que é Lógica? Em caso de uma resposta afirmativa, parabéns! Contudo, há inúmeros outros conhecimentos para construir sobre o tema... Para tanto, sugerimos que consulte as obras, periódicos e sites indicados ao final desta Aula. Agora, vamos aplicar a Lógica na análise de sistemas!

1.1 - Lógica na Análise de Sistemas

A Lógica de programação é a técnica de encadear pensamentos para atingir um determinado objetivo.

Para quem deseja trabalhar com desenvolvimento de sistemas, ela permite definir a sequência Lógica para alcançar a resolução de um problema. Esses pensamentos são descritos como uma sequência de instruções. Essa sequência é executada passo a passo até atingir a solução.

Os profissionais de análise no seu dia a dia dentro de suas empresas terão que solucionar problemas e atingir os objetivos apresentados com eficiência e eficácia.

No decorrer da disciplina você terá a oportunidade de entender melhor a técnica para desenvolver o raciocínio lógico, lembrando que, para que isso aconteça você precisará ser persistente e praticá-lo constantemente. Passemos, a seguir, para o estudo sobre os Algoritmos.

2 - Algoritmo: definição e criação por meio da percepção de um problema

Algoritmos são sequências de ações ou instruções organizadas logicamente para resolver um problema. São, portanto, os primeiros passos para a construção de programas, uma vez que por meio deles desenvolvemos o raciocínio lógico necessário para a resolução do problema proposto.

Guimarães e Lages definem Algoritmo como:

> [...] descrição de um padrão de comportamento, expressado em termos de um repertório bem definido e finito de ações primitivas, das quais damos por certo que elas podem ser executadas (GUIMARÃES; LAGES, 1994, p. 4).

Ou ainda: "[...] uma norma executável para estabelecer um certo efeito desejado, que na prática será geralmente a obtenção de uma solução a um certo tipo de problema" (GUIMARÃES, 1994, p. 10).

Você sabia que o criador do conceito de Algoritmo foi Leonard Euler e que o primeiro Algoritmo de que se tem notícia refere-se à previsão das fases da lua. No século XVIII, muito antes da existência dos GPSs, Euler possibilitou criar tabelas precisas de navegação, fundamentais para um navio descobrir onde está? (GOVERNO DO ESTADO DO PARANÁ, 2011).

Para todas as tarefas executadas no dia a dia, nosso cérebro constrói Algoritmos, o que acontece de uma forma tão natural que nem percebemos.

Isso significa que mesmo sem notar, para cada problema ou atividade a ser trabalhada, definimos uma sequência Lógica de ações, ou seja, construímos vários Algoritmos todos os dias, mesmo que sem darmos conta disso.

Os Algoritmos são utilizados em praticamente todas as áreas do conhecimento existentes, pois as pessoas envolvidas na resolução dos mais diversos tipos de problemas precisam formular uma sequência de ações para chegar a uma solução ou resultado satisfatório.

Para exemplificar, vamos imaginar uma situação da vida real, na qual podemos perceber a criação de um Algoritmo. Por exemplo, para assistir um filme em DVD, algumas sequências de ações deverão ser executadas. Nesse caso, considere que o aparelho de DVD já está conectado a uma TV e que ambos estão prontos para funcionar. Precisaremos, portanto, executar os seguintes passos:

a) Ligar a TV.
b) Ligar o DVD.
c) Abrir o compartimento do disco.
d) Inserir o disco.
e) Fechar o compartimento do disco.
f) Pressionar a tecla PLAY para iniciar o filme.

Note que a sequência de instruções apresentada precisou ser definida para que fosse possível assistir ao filme no aparelho de DVD. Para nós, pode parecer uma coisa simples, mas tivemos que montar uma sequência de instruções para que fosse possível atender à necessidade (problema) em questão.

Assim como esse exemplo, em nossa vida cotidiana

encontraremos diversas outras situações em que teremos que desenvolver Algoritmos para resolver determinados problemas, tais como: trocar o pneu de um carro, preparar um bolo, arrumar o filho para ir à escola, etc.

Japoneses afirmam ter criado o Algoritmo que adivinha a proximidade da morte. "A insólita descoberta foi feita por programadores da Universidade de Yokohama com base na análise de seis meses de gravações às chamadas para os serviços de emergências da localidade [...]. Segundo os investigadores, o objetivo do Algoritmo é identificar as chamadas de emergência de quem está mais necessitado, no momento da triagem" (CURIOSIDADES 10, 2011).
Como podemos perceber, criar Algoritmos pode nos ajudar a solucionar os mais diversos problemas e encontrar soluções para diferentes áreas do conhecimento.

É importante observar ainda que para cada problema a ser resolvido, existem diversos caminhos que levam à solução desejada, isto é: um problema pode ser resolvido de duas ou mais maneiras diferentes, obtendo o mesmo resultado, ou ainda podemos ter soluções mais eficientes e eficazes que outras para atingir o mesmo objetivo.

No caso do DVD, por exemplo, ao invés de ligar primeiro a TV, poderíamos ter ligado o DVD e depois a TV, o que seria outra sequência para a solução do problema, porém capaz de proporcionar o mesmo resultado. Além dessa, ainda teríamos a opção de criar uma lista de ações com mais detalhes, como: acrescentar a ação de ajustar o canal da TV, ou ainda, pressionar a tecla TV/VÍDEO para ajustar a sintonia com o aparelho de DVD.

Em outras palavras, podemos afirmar que o nível de detalhamento do Algoritmo varia de acordo com o problema a ser resolvido. Entretanto, isso não significa que uma solução mais detalhada seja melhor ou pior que outra menos detalhada, uma vez que a decisão de usar mais ou menos ações para atingir o objetivo deve ser analisada em cada situação separadamente.

No computador vale a mesma regra! Um problema poderá ter duas ou mais soluções diferentes para chegar ao objetivo, que é a sua solução.

Por exemplo, você deseja sacar dinheiro em um caixa automático, então precisa:

a) ir até o banco;
b) abrir a porta do caixa eletrônico;
c) passar o cartão no leitor do caixa;
d) digitar sua senha da conta-corrente;
e) escolher a opção "Saque";
f) digitar o valor desejado e confirmar;
g) retirar o dinheiro.

Vejamos outro exemplo: trocar uma lâmpada! Inúmeras vezes fazemos esse tipo de atividade inconscientemente, sem percebermos os pequenos detalhes. Contudo, agora, vamos ver como ficaria a solução do problema passo a passo:

a) pegar uma escada;
b) posicionar a escada embaixo da lâmpada;
c) buscar uma lâmpada nova;
d) subir na escada;
e) retirar a lâmpada velha;
f) colocar a lâmpada nova.

Mas, e se a lâmpada não estiver queimada?

Neste caso, a pessoa poderá trocar a lâmpada do mesmo modo, mesmo não prevendo essa situação. Para solucionar esse novo problema, podemos efetuar um teste seletivo, verificando se a lâmpada está ou não queimada:

a) pegar uma escada;
b) posicionar embaixo da lâmpada;
c) buscar uma lâmpada nova;
d) ligar o interruptor;
e) se a lâmpada não acender, então:
• subir na escada;
• retirar a lâmpada velha;
• colocar a lâmpada nova.

Viu como a criação de Algoritmos é algo simples que você já está habituado a fazer diariamente? Agora, para sedimentar sua aprendizagem, sugerimos que pare um minuto, reflita sobre o que estudou e tente escrever o Algoritmo que você criou para estudar esta Aula: fui até o escritório, selecionei a apostila da disciplina Algoritmos, abri na página 5, etc. Com o resultado de sua autoavaliação, você poderá escolher entre prosseguir seus estudos ou pedir ajuda para eliminar eventuais dúvidas, antes que elas se tornem uma "bola de neve"! Agora, vamos passar ao estudo sobre a percepção dos problemas, ações indispensáveis para criação e aplicação de Algoritmos.

2.1 - Percepção do Problema

Diferente do que vários profissionais que trabalham com Algoritmos e programação acreditam que a primeira coisa a se pensar não é a solução de um problema, mas a percepção dele.

Essa é a razão pela qual muitas pessoas sentem dificuldades em iniciar o desenvolvimento do Algoritmo: justamente porque focam suas ideias iniciais na solução e não no problema.

Sabendo disso, podemos afirmar que nenhum problema será resolvido se não for minuciosamente entendido. Portanto, ao realizar um Algoritmo o foco inicial deve estar sobre o problema, para depois se pensar em uma solução.

O entendimento detalhado do problema é um fator que diferencia uma solução algorítmica da outra. Assim, quanto mais se conhece sobre o problema em questão, maiores as chances de se criar soluções mais adequadas, eficientes e eficazes.

Fique antenado! No *site* do *You Tube* você pode encontrar inúmeros vídeos sobre a Lógica de programação, a criação e o emprego de Algoritmos. Sugiro que realize buscas utilizando esses termos como palavras-chave, procure assistir alguns deles e se posicionar criticamente em relação ao conteúdo. Com os conhecimentos que está adquirindo, cada vez mais, você se torna capaz de superar o senso comum sobre as informações disponibilizadas nos diferentes meios de comunicação e utilizá-las como fontes de pesquisa, sempre que considerar que se trata de conhecimentos úteis!

Ah, lembre-se de que esse hábito será fundamental no dia a dia de sua profissão!

Daí surge desafios para a criação do Algoritmo que são:

a) usar de criatividade para apresentar uma ou mais alternativas de solução para um problema;

b) transformar ideias abstratas em uma sequência de ações.

Como você pôde notar, a arte de criar Algoritmos não pode ser encarada apenas como a criação de uma solução para o problema, mas como a criação da melhor solução possível.

Já sabemos que podem existir várias soluções para o mesmo problema cabe, então, ao desenvolvedor do Algoritmo pensar em duas ou mais soluções diferentes para poder analisá-las, compará-las e decidir qual delas é a melhor.

Vale lembrar ainda que o tamanho ou complexidade do Algoritmo não é relativamente proporcional à sua qualidade.

Assim, para cada problema a ser levado ao computador, devem-se planejar as operações correspondentes. O automatismo exige que o planejamento dessas operações seja realizado previamente, ou seja, antes de se utilizar o computador.

Em outras palavras, a utilização de um computador, para se resolver qualquer problema exige, antes de qualquer coisa, que se desenvolva um Algoritmo, isto é, que se faça a descrição do conjunto de comandos ordenados que, quando obedecidos, resultarão na realização da tarefa desejada, obtendo os resultados esperados.

Os Algoritmos encontram-se entre a ideia de se resolver um problema por meio do computador e o sistema desenvolvido pelas linguagens de programação.

A elaboração de Algoritmos é, para estudantes de Análise de Sistemas, o passo inicial para a solução de qualquer problema computacional.

Isso significa que identificado o problema, será necessária a escrita de uma série de instruções, normalmente em português, ou bem próximas dele, em uma ordem logicamente definida, para que se obtenha a solução do problema. É desse modo que se torna possível a construção de programas em qualquer linguagem de computador. Para entender melhor a elaboração de um programa, observe atentamente a Figura 1.1:

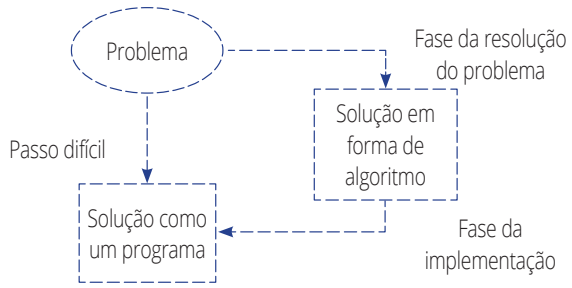


Figura 1.1 Resolução de um problema e programação.
Fonte: acervo pessoal.

Além disso, é importante ressaltar que qualquer tarefa que siga determinado padrão pode ser descrita por um Algoritmo e que para montá-lo, precisamos dividir o problema apresentado em três fases fundamentais. Veja Figura 1.2:

Figura 1.2 Fases de um problema.
Fonte: acervo pessoal.

Em que:

a) Entrada: são os dados de entrada de um Algoritmo;

b) Processamento: são os procedimentos utilizados para chegar ao resultado final;

c) Saída: são os dados já processados.

Compreenderam?
Entrada: são os dados fornecidos inicialmente para se criar um Algoritmo.
Processamento: é a forma como o problema foi resolvido.
Saída: é a solução do problema.
Simples, vocês não acham?

Retomando a aula

Parece que estamos indo bem. Então, para encerrar essa aula, vamos recordar:

1 - Lógica: conceituação e aplicação na análise de sistemas

Na referida Seção você teve a oportunidade de construir conhecimentos sobre Lógica, a qual se trata de uma ciência muito importante, capaz de garantir que nosso pensamento proceda corretamente e que cheguemos às conclusões coerentes e verdadeiras.

Desse modo, vimos que a aplicação da Lógica na análise de sistemas é fundamental para a construção de Algoritmos e, por conseguinte, programas logicamente estruturados para atingir os objetivos almejados pelo(s) seu(s) criador(es).

2 - Algoritmo: definição e criação por meio da percepção de um problema

Iniciamos nosso estudo sobre os Algoritmos, ou seja, uma sequência finita, bem definida e Lógica de instruções que podem ser utilizadas para análise e percepção de um problema em busca de sua solução. Vimos que podemos propor diferentes Algoritmos para encontrar várias soluções e escolher a que melhor atinge os objetivos de um programa que se pretende construir.

É importante observar que os conhecimentos construídos nesta Aula são fundamentais e servem como base de muitos outros conhecimentos importantes para o profissional que atua ou pretende atuar na área de Análise de Sistemas.

Caso você tenha ficado com dúvidas sobre a Aula 1, acesse as ferramentas "fórum","quadro de avisos" ou "chat"e interaja com seus colegas de curso e com seu professor. Participe, afinal você é o personagem principal de sua aprendizagem!

Vale a pena

Vale a pena **ler**

GUIMARÃES, Ângelo de Moura; LAGES, Newton Alberto de Castilho. *Algoritmos e estrutura de dados*. Rio de Janeiro: LTC, 1994.

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de Oliveira. *Algoritmos* - Lógica para desenvolvimento de programação. 2. ed. São Paulo: Érica, 2007.

Vale a pena **acessar**

CURIOSIDADES 10. Japoneses criam Algoritmo que adivinha proximidade da morte. Disponível em: <http://www.curiosidades10.com/tecnologia/japoneses_criam_Algoritmo_que_adivinha_proximidade_da_morte.html#>. Acesso em: 20 maio 2011.

GOVERNO DO ESTADO DO PARANÁ. Bate byte - flagrantes: o criador dos Algoritmos. Disponível em: <http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=1679>. Acesso em: 20 maio 2011.

TEC MUNDO. O que é algoritmo? Disponível em: <http://www.tecmundo.com.br/2082-o-que-e-Algoritmo-.htm>. Acesso em: 20 maio 2011.

Vale a pena **assistir**

YOU TUBE. 1ª de 8 Aulas= Algoritmo - introdução, definição e conceitos. Disponível em: <www.youtube.com/watch?v=3hv5_hWPIeo>. Acesso em: 20 maio 2011.

Minhas anotações

2º Aula

Algoritmos

Estudaremos, neste capítulo, os Algoritmos. Veremos como estruturá-los e identificaremos os métodos que podem ser de ajuda na escrita de programas. É importante que qualquer dúvida seja eliminada na plataforma, utilizando as ferramentas "quadro de avisos" ou "fórum".

Não deixe que suas dúvidas o(a) impeçam de prosseguir em seus estudos. Lembre-se: cada passo compreendido é um passo para uma base melhor, o que vai tornar você um profissional mais qualificado.

Boa aula!

Objetivos de aprendizagem

Ao término desta aula, o aluno será capaz de:

- identificar a relação entre Algoritmos e programas;
- diferenciar as três linguagens de programação e saber quando usá-las;
- reconhecer e entender os símbolos do diagrama de bloco (ou diagrama de fluxo).

Seções de estudo

1 - Algoritmos estruturados
2 - Linguagens de programação

1 - Algoritmos estruturados

1.1 - Introdução

Desenvolver Algoritmos estruturados significa, basicamente, utilizar uma metodologia de projeto de programas, com o objetivo de facilitar:

a) a escrita dos Algoritmos;
b) a leitura e o entendimento dos Algoritmos;
c) a manutenção e modificação dos Algoritmos.

Você já havia pensado sobre as razões e a importância de se desenvolver Algoritmos estruturados?

O grande desafio da escrita de programas é reduzir a complexidade. Muitos programas são escritos de uma forma tão complicada que até o próprio autor tem dificuldade de interpretá-los depois. Para reduzir a complexidade dos Algoritmos deve-se:

a) Escrever o Algoritmo fazendo refinamentos sucessivos. Isso é chamado de desenvolvimento top-down – essa técnica consiste em ir escrevendo as funções principais do programa e depois detalhar cada uma delas em funções menores, até que se tenha um último nível que não é mais possível detalhar.

b) Decompor o Algoritmo todo em módulos funcionais. É mais fácil compreender uma solução se a analisarmos por partes, ao invés de olharmos a solução como um todo. A esse processo chamamos de modularização.

c) Usar soluções simples e não muito extensas em cada um dos módulos, com poucas estruturas de controle para facilitar o entendimento.

Vamos, agora, observar em mais detalhes como utilizar cada uma das técnicas citadas.

1.2 - Principais Técnicas de Estruturação dos Algoritmos

As três principais técnicas para a estruturação dos Algoritmos são:

a) Desenvolvimento top-down: depois que entendemos o problema a ser solucionado, precisamos formular alternativas de solução. Para isso, formulamos uma possível solução, a qual precisa ser refinada, ou seja, detalhada em partes menores até que se tenha uma visão de todos os detalhes.

b) Modularização: a modularização consiste em dividir a solução do problema em partes, ou módulos, cada um com funções bem definidas, dando maior agilidade ao desenvolvimento.

c) Estruturas de controle: o uso de estruturas de controle adequadas afeta diretamente a qualidade do Algoritmo. Muitos programadores têm o costume de usar comandos de desvio incondicional ou de interrupção do Algoritmo. Um desvio incondicional é aquele em que o programa muda seu fluxo de execução arbitrariamente. Esse artifício pode ser usado quando não se tem um completo domínio da lógica do programa, mas deve ser evitado, pois provoca a quebra da estrutura lógica do Algoritmo.

Os Algoritmos estruturados são fantásticos, você não acha? Agora, vamos passar ao estudo sobre as linguagens de programação.

2 - Linguagens de programação

Escrever um programa é o mesmo que traduzir um Algoritmo para uma linguagem de programação qualquer. As linguagens de programação são softwares que permitem transformar um Algoritmo em um programa de computador.

Aprender uma nova linguagem de programação é uma tarefa fácil quando se tem um bom conhecimento de Algoritmos, pois o maior problema na criação de um programa não é a linguagem em si, mas sim as dificuldades na percepção do problema e na formulação de uma boa solução.

Existem linguagens para os mais diversos domínios de aplicação, cada uma com seus propósitos. Algumas são destinadas ao desenvolvimento de aplicações comerciais, outras têm propósito científico, aplicações em inteligência artificial ou na criação de programas para internet.

Toda e qualquer linguagem é constituída por signos. Signos são imagens, gestos, sons melódicos e outros sistemas de significação. Os signos não têm significado, por si mesmos, mas recebem sentidos convencionados e, assim, passam a ser usados na comunicação, seja linguística, seja de programação. A ciência que estuda os signos é a Semiologia (BARTHES, 2006).

As linguagens de programação podem ser dos seguintes tipos: interpretadas, compiladas ou híbridas.

2.1 - Linguagens interpretadas

Nas linguagens interpretadas, o interpretador lê, analisa e executa cada instrução do programa fonte, sem traduzir para uma linguagem de máquina. Cada linha ou instrução é executada na sequência. Quando um erro é encontrado, a execução do programa é interrompida.

O interpretador simula por software uma máquina virtual, na qual o ciclo de execução entende os comandos da linguagem de alto nível.

Note que esse tipo de linguagem oferece algumas desvantagens, como a necessidade da presença do código fonte para a execução do programa. Além disso, o interpretador da linguagem precisa estar instalado no computador onde está o programa.

A execução do programa é bem mais lenta do que nas linguagens compiladas, uma vez que o interpretador precisa analisar as instruções do programa sempre que vai executá-las.

Vamos visualizar como funcionam as linguagens interpretadas na Figura 2.1.

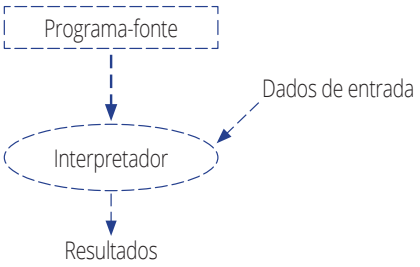


Figura 2.1 Funcionamento das linguagens interpretadas.
Fonte: acervo pessoal.

2.2 - Linguagens Compiladas

Os compiladores têm a tarefa de ler e analisar o programa escrito em uma linguagem de programação – o programa fonte ou código fonte – e traduzi-los para a linguagem de máquina, executando-os diretamente no computador.

Diferentemente da interpretação, a compilação analisa todo o código fonte à procura de erros. Só depois que essa análise termina – e sem que nenhum erro tenha sido encontrado – é que será criado um código intermediário chamado de "código objeto". Esse código objeto será então *linkeditado* (ou ligado) e um código executável será gerado.

Um compilador é um programa (ou um conjunto de programas) que traduz um código fonte para uma linguagem de mais baixo nível (a linguagem alvo, que tem uma forma binária conhecida como código objeto). Normalmente, o código fonte é escrito em uma linguagem de programação de alto nível, com grande capacidade de abstração, e o código objeto é escrito em uma linguagem de baixo nível, como uma sequência de instruções a ser executada pelo processador. O compilador é um dos dois tipos mais gerais de tradutores, juntamente com o interpretador (PESQUOMPILE, 2011).

Nos programas compilados, não há a necessidade da presença do código fonte para a sua execução, assim como o compilador não precisa estar instalado para isso. O programa executável criado é independente da linguagem. Nesse caso, a execução do programa é mais rápida, uma vez que o programa fonte não precisará ser analisado em cada execução.

A figura 2.2 mostra o processo de compilação. Observe-a:

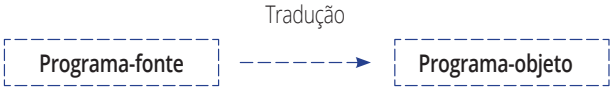


Figura 2.2 Processo de compilação.
Fonte: acervo pessoal.

2.3 - Linguagens Híbridas

As linguagens híbridas usam a interpretação e a compilação. Nesse contexto, o compilador tem o papel de converter o código fonte em um código conhecido por byte code – uma linguagem de baixo nível, que depois é interpretada. Como exemplo, podemos citar a linguagem JAVA, que gera um código chamado de Java Bytecode (EBAH, 2011).

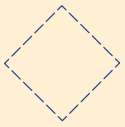
[...] Que uma linguagem híbrida "tem maior portabilidade que uma linguagem compilada"? (UFSCAR, 2011)

Programas escritos em uma linguagem híbrida são mais rápidos que os de uma linguagem interpretada, isso porque as instruções intermediárias são projetadas para serem interpretadas facilmente.

2.4 - Diagrama de Bloco

Também conhecido como diagrama de fluxo, é uma forma padronizada para representar os passos lógicos de um determinado processamento. Com ele, podemos definir uma sequência de símbolos, de significados bem definidos. Sua função é facilitar a visualização dos passos da resolução de um Algoritmo. Dentro do símbolo sempre é escrita a instrução a ser realizada. Veja Quadro 2.2:

Quadro 2.2 Diagrama de bloco (fluxo)

Símbolo	Função
 TERMINAL	Indica o início e/ou fim do fluxo do algoritmo.
 PROCESSAMENTO	Indica cálculos a efetuar, atribuições e valores.
 DECISÃO	Indica a decisão que deve ser tomada, indicando a possibilidade de desvios.

Fonte: acervo pessoal.

Notem que, nos diagramas de blocos, a descrição do raciocínio se dá por meio de símbolos gráficos. É por intermédio deles que resolveremos os Algoritmos.

Vale salientar que esses símbolos foram sendo criados de acordo com a necessidade de apresentar a resolução dos Algoritmos. Nós vimos apenas três deles: **terminal**, **processamento** e **decisão**. No entanto, existem muitos outros, como: **entrada de dados**, **saída de dados**, **saída de dados no vídeo**, **conector**, **preparação**, **linha de comunicação**.

O que dificulta a utilização dos diagramas são os desenhos que se tornam um pouco confusos e complexos, bem como sua difícil manutenção.

Atualmente, eles não são mais utilizados na prática.

Vamos agora refazer os exercícios realizados na Aula 1. Contudo, desta vez utilizando a simbologia dos diagramas de blocos. Faça uma tentativa e depois compare-a com a Figura 2.3.

Figura 2.3 Exercícios utilizando diagramas de blocos.
Fonte: acervo pessoal.

Retomando a aula

Para continuar apresentando um desempenho satisfatório, é importante que retomemos o conteúdo estudado nesta Aula.

1 - Algoritmos estruturados

Na primeira Seção tivemos a oportunidade de entender que desenvolver Algoritmos estruturados significa, basicamente, utilizar uma metodologia de projeto de programas com os objetivos de facilitar a escrita, a leitura e o entendimento dos Algoritmos.

Além disso, estudamos as principais técnicas para a estruturação dos Algoritmos, ou seja, o desenvolvimento top-down, a modularização e as estruturas de controle.

2 - Linguagens de programação

Na Seção 2, construímos conhecimentos sobre as linguagens interpretadas, em que o interpretador simula por software uma máquina virtual que faz com que o ciclo de execução entenda os comandos da linguagem de alto nível; as linguagens compiladas, nas quais os compiladores têm a tarefa de ler e analisar o programa escrito em uma linguagem de programação e traduzi-los para a linguagem de máquina, executando-os diretamente no computador; as linguagens híbridas que usam a interpretação e a compilação; bem como o diagrama de bloco, com o qual podemos definir uma sequência de símbolos, de significados bem definidos que facilitam a visualização dos passos da resolução de um Algoritmo.

Espero que seus estudos referentes à Aula 2 tenham sido proveitosos. Mas, caso ainda tenha dúvidas, sugerimos que acesse o ambiente virtual e utilize as ferramentas apropriadas para se comunicar com seus colegas de curso e com seu professor. Além disso, é importante que reflita sobre os conteúdos e as estratégias didáticas empregadas para a aprendizagem desta Aula: o que foi bom? O que pode melhorar? Lembre-se de que estaremos esperando suas sugestões para melhorar nossos recursos e técnicas didáticas utilizados no curso. Afinal, na EAD a construção de conhecimento é um trabalho de todos. Participe, nós também queremos aprender com você!

Vale a pena

Vale a pena **ler**

BARTHES, Roland. *Elementos de semiologia*. São Paulo: Cultrix, 2006.

GUIMARÃES, Ângelo de Moura; LAGES, Newton Alberto de Castilho. *Algoritmos e estrutura de dados*. Rio de Janeiro: LTC, 1994.

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de Oliveira. *Algoritmos* - lógica para desenvolvimento de programação. 2. ed. São Paulo: Érica, 2007.

VENÂNCIO, Cláudio Ferreira. *Desenvolvimento de algoritmos:* uma nova abordagem. São Paulo: Érica, 2000.

Vale a pena **acessar**

OFICINA DA NET. Algoritmo estruturado. Disponível em: <http://www.oficinadanet.com.br/apostilas/detalhe/576/apostila_de_algoritmo_estruturado>. Acesso em: 28 jun. 2011.

PEQUOMPILE. O que é um compilador. Disponível em: <http://www.oficinadanet.com.br/apostilas/detalhe/576/apostila_de_algoritmo_estruturado>. Acesso em: 28 jun. 2011.

UFSCAR – Universidade Federal de São Carlos. Métodos de implementação de linguagens. Disponível em: <http://www.oficinadanet.com.br/apostilas/detalhe/576/apostila_de_algoritmo_estruturado>. Acesso em: 28 jun. 2011.

Vale a pena **assistir**

FERNANDES, A. R. Entendendo algoritmos - Parte I. Disponível em: <http://www.youtube.com/watch?v=NUtmw47rroI>. Acesso em: 28 jun. 2011.

Minhas anotações

Aula 3º

Linguagem algorítmica

Prezado(a) aluno(a), para iniciar a Aula 3 é necessário que tenha compreendido a estrutura dos Algoritmos, bem como os métodos que podem possibilitar e/ou facilitar a escrita de programas. Assim, se houver ainda alguma dúvida sobre os referidos temas, sugerimos que releia a Aula 2 e procure eliminá-las com o apoio de seu professor, antes de prosseguir.

Já nesta Aula, você será convidado(a) a construir conhecimentos sobre a linguagem algorítmica... Mas, você já pensou sobre a razão pela qual vamos estudar esse assunto?

Pois bem, partindo da linguagem algorítmica é possível escrever Algoritmos para serem seguidos (executados) pelo computador... Viu como ela é importante!

Ah, durante o estudo, lembre-se de que estaremos esperando sua participação no "fórum","quadro de avisos" ou "chat". Agora, é com você!

Boa aula!

Objetivos de aprendizagem

Ao término desta aula, o aluno será capaz de:

- conceituar o termo linguagem Algorítmica e sua função e termos correlacionados, tais como Portugol, dados, operadores e constantes;
- aplicar a linguagem Algorítmica na programação e na análise de sistemas;
- desenvolver e/ou ampliar a capacidade de percepção e de utilização da linguagem algorítmica.

Seções de estudo

1 - Portugol

Vamos começar nossos estudos, conhecendo o "Portugol". Ele é indispensável para quem pretende dominar a lógica de programação, o princípio de construção de Algoritmos, além de desenvolver boas práticas de programação... Bons estudos!

Para entender o que é o Portugol, é preciso responder a uma questão: o que é linguagem Algorítmica?

Uma linguagem algorítmica é uma pseudolinguagem de programação, que utiliza comandos e instruções em Português para representar as ações dos Algoritmos. A essa pseudolinguagem damos o nome de **Portugol**, também conhecida como **Português Estruturado**.

A necessidade de facilitar o trabalho dos profissionais de informática no computador é constante e uma das formas para se conseguir esse objetivo é fazer com que o computador seja, cada vez mais, capaz de compreender a linguagem escrita ou falada, reduzindo ao máximo a quantidade de códigos e símbolos que precisam ser aprendidos e memorizados.

Mas, infelizmente os computadores são um pouco limitados...

Por esse motivo, estamos sempre buscando formas para fazer com que o computador aprenda a nossa língua, ao invés de nós aprendermos a "língua" dele.

Como você pode imaginar, os programadores teriam seu trabalho facilitado se os programas fossem escritos em sentenças padronizadas da linguagem humana. Entretanto, infelizmente, isso não acontece. Desse modo, os programas precisam ser escritos em uma linguagem de programação e há muitas dessas linguagens.

Em tese, o estudante de programação deveria priorizar seus esforços no entendimento e na resolução do problema, bem como no desenvolvimento do raciocínio lógico necessário e na abstração, ou seja, na capacidade de definir e usar estruturas ou operações complicadas, sem visualizar muitos detalhes (SEBESTA, 1999). Contudo, isso só será possível se ele não precisar se preocupar com a tradução de cada um dos comandos que deve utilizar e não perder tempo com mensagens de erro, cujo significado nem sempre é entendido corretamente.

Por essa razão, o aprendizado de Algoritmos torna-se fundamental para a elaboração de programas estruturados e para o aprimoramento da lógica de programação, uma vez que eliminam a preocupação com o idioma em que se apresentam os programas da linguagem estruturada, bem como a necessidade de conhecimento de uma linguagem de programação específica. Isso significa que o processo de aprender uma linguagem de programação para resolver os problemas do Algoritmo pode ser uma tarefa extensa e difícil.

Já a programação em uma linguagem algorítmica é uma simples transcrição de palavras-chave, o que torna o processo bem mais fácil: uma vez predefinidas as sequências lógicas das tarefas ou instruções a serem realizadas passo a passo, necessita-se apenas traduzi-las em uma linguagem própria que o computador reconheça para, então, submetê-las à máquina para análise e obter o seu resultado.

A vantagem dos Algoritmos, portanto, não está no fato de eliminar a adoção de regras comuns na programação, mas no fato de o usuário ter a possibilidade de escrever seu programa em português, o que dará ao programador maior facilidade para compreender e assimilar a lógica do programa, ao mesmo tempo em que exigirá o cumprimento de regras obrigatórias para confecção dos Algoritmos.

Você percebeu como o Portugol pode facilitar o cotidiano profissional do analista em desenvolvimento de sistemas? Assim, é importante que continue buscando conhecimentos sobre esse tema... Para tanto, sugerimos que consulte as obras, periódicos e sites indicados ao final desta Aula.

Na Seção seguinte vamos avançar em nossa aprendizagem estudando os tipos de dados!

2 - Tipos de dados

Você sabia que os "tipos de dados influenciam na forma como o Algoritmo irá trabalhar, o desempenho do Algoritmo e o seu consumo de memória"? (EXPERT.NET, 2011).

Mas, quais são e quais as características dos tipos de dados?

Os dados podem ser classificados em três tipos: numéricos (inteiros e real), caracteres (valores alfabéticos ou alfanuméricos) e lógicos (valores verdadeiros ou falsos).

Vamos entendê-los melhor!

2.1 - Numéricos inteiros

Numéricos e inteiros são toda e qualquer informação numérica que pertença ao conjunto dos números inteiros (negativa, nula, positiva). Vale salientar que, por sua natureza, os números inteiros não possuem parte fracionária.

Exemplo: 38,0,541,-56,-45.

O dado inteiro em Portugol é representado pelo comando "inteiro".

2.2 - Numéricos reais

Numéricos reais são toda e qualquer informação numérica que pertença ao conjunto dos números reais (negativa, nula, positiva e fracionária).

Exemplo: 38,0,541,-56,5.5,3.4.

O dado inteiro em Portugol é representado pelo comando "real".

2.3 - Caracteres

Os caracteres são sequências de valores delimitados por aspas (" ") formadas por:

a) letras de A até Z;
b) números de 0 até 9;
c) por símbolos: &,*,@ e um espaço em branco.

O tipo de dado 'caractere' também é conhecido como alfanumérico e pode ser representado literal ou cadeia.

Exemplo: "Análise", "Rua João Cândido Câmara, 1220", "Fone: 3422-3977", "Cep:79824-900".

O dado caracter em Portugol é representado pelo comando "caractere".

2.4 - Lógicos

Os dados lógicos possuem o valor 'verdadeiro' ou 'falso', sendo que esse tipo de dado poderá representar apenas um dos dois valores. Afirmamos, então, que o tipo de dado é lógico ou booleano.

O dado lógico também é denominado booleano por ter sido desenvolvido na álgebra de George Boole. Este teórico teve uma formação inicial rudimentar. Contudo, foi autodidata. Aos 20 anos, fundou "sua própria escola e dedicou-se ao estudo da Matemática. Em 1840 publicou o seu primeiro trabalho original e em 1844 foi condecorado com a medalha de ouro da Royal Society pelo seu trabalho sobre cálculo de operadores. Em 1847 publicou um volume sob o título The mathematical analysis of logic em que introduziu os conceitos de lógica simbólica demonstrando que a lógica podia ser representada por equações algébricas. Este trabalho foi fundamental para a construção e programação dos computadores eletrônicos iniciada cerca de 100 anos mais tarde" (BRASIL ESCOLA, 2011).

Os dados lógicos também são representados da seguinte forma: Verdadeiro - .v. – Falso - .f.

O dado lógico em Portugol é representado pelo comando "**lógico**".

3 - Operadores

"Operadores são sinais que são alimentados por expressões e que retornam um valor de acordo com a operação realizada" (NOBIOS, 2011).

Vamos entender mais claramente: os operadores possuem uma sequência na qual as expressões são avaliadas e resolvidas. Se dois operadores de uma mesma expressão possuírem o mesmo nível de precedência, a expressão será avaliada da esquerda para a direita. As expressões contidas entre parênteses serão resolvidas em primeiro lugar, a começar pelos parênteses mais internos.

A ordem de precedência é a seguinte:
1º - Expressões dentro de parênteses e funções.
2º - Operador unário menos ou negação.
3º - Operadores aritméticos multiplicativos: *, /.
4º - Operadores aritméticos aditivos: +, -.
5º - Operadores relacionais: =, <>, <, >, <=, >=.
6º - Operadores lógicos: e, ou, não.

Possuímos três tipos de operadores: os aritméticos, relacionais e lógicos, os quais vamos compreender em detalhes, nos tópicos a seguir.

3.1 - Operadores aritméticos

Os operadores aritméticos são utilizados para operações matemáticas a serem realizadas, para obter resultados numéricos nas expressões. São eles:

Quadro 3.1 - Operadores aritméticos

Adição	+
Subtração	-
Multiplicação	*
Divisão Real	/
Divisão Inteira	\
Módulo (Resto de divisão inteita)	MOD

Fonte: acervo pessoal.

Exemplos:

Quadro 3.2 - Exemplos de operadores aritméticos

2 + 3	5
7 * 4	28
5 / 2	2.5 o quociente de uma divisão real
5 \ 2	2 o quociente de uma divisão inteira
5 MOD 2	1 o resto de uma divisão inteira

Fonte: acervo pessoal.

3.2 - Operadores relacionais

Um operador relacional existe para estabelecer uma relação entre dois elementos.

Com isso o resultado da comparação na expressão será sempre falso -.f. ou Verdadeiro –.v. Conheça esses operadores:

Quadro 3.3 - Operadores relacionais

Igual a	=
Menor que	<
Maior que	>
Maior ou igual a	>=
Menor ou igual a	<=
Desigualdade (diferente de)	<>

Fonte: acervo pessoal.

3.3 - Operadores lógicos

Os operadores lógicos permitem estender o uso dos operadores relacionais, permitindo, desse modo, composições lógicas mais sofisticadas nas expressões. Observe-os:

Quadro 3.4 - Operadores lógicos

E - AND ^ para **conjunção**	A expressão (E) é verdadeira se todas as condições forem verdadeiras.
OU - OR v para **disjunção**	A expressão (OU) é verdadeira se pelo menos uma condição for verdadeira.
NÃO - NOT ¬ para **negação**	A expressão (NÃO) inverte o valor da expressão ou condição, se verdadeira inverte para falsa e vice-versa

Fonte: acervo pessoal.

Como você pôde deduzir partindo da leitura do Quadro 3.1, a conjunção de duas proposições é verdadeira se e somente se ambas as proposições são verdadeiras.

Ufa! Isso parece complicado, mas no Quadro 3.5, está tudo exemplificado... Vejam!

Quadro 3.5 - Conjunção de duas proposições

p	q	p ^ q
V	V	V
V	F	F
F	V	F
F	F	F

Fonte: acervo pessoal.

Neste contexto:
a) p ^ q = F.
b) p ^ r = F.
c) q ^ r = F.

Já a disjunção de duas proposições é verdadeira se e somente se, pelo menos, uma delas for verdadeira. Veja:

Quadro 3.6 - Disjunção de duas proposições

p	q	p v q
V	V	V
V	F	V
F	V	V
F	F	F

Fonte: acervo pessoal.

Para o exemplo anterior:
a) p v q = V.
b) p v r = V.
c) q v r = F.

Finalmente, a negação pode ser formada inserindo-se a palavra não antes da proposição. Assim:

Quadro 3.7 - Negação em uma proposição.

p	¬ p
V	F
F	V

Fonte: acervo pessoal.

No decorrer desta e das demais Aulas da disciplina, você terá a oportunidade de entender melhor o uso dos operadores aritméticos. Porém, para que isso aconteça é necessário ser persistente e estudar constantemente sobre esse tema. Pense nisso...

4 - Constantes

Constante é um identificador que armazena um valor fixo, ou seja, um valor não se modifica no decorrer do Algoritmo.

As constantes podem ser dos tipos: numérica, lógica ou caractere (literal). Para entendê-las, vamos conhecer alguns exemplos:

a) **Exemplo 01 (constantes numéricas)**: podem ser representadas por um número inteiro ou real, positivo, negativo ou nulo (32, 3, 1415, -54, 0342).

b) **Exemplo 02 (constantes lógicas)**: podem ser lógicas assumindo um dos seguintes valores: Verdadeiro (V) ou Falso (F).

c) **Exemplo 03 (constantes caractere [literal])**: são valores do tipo caractere, ou seja, qualquer sequência de caracteres (letras, dígitos ou símbolos especiais). A constante literal deve sempre aparecer entre aspas ("Castro Alves", "X1Y2W3", "*A!B?-/", "1234").

Compreenderam?
Uma constante é um valor que não se modifica no decorrer do Algoritmo. Pode ser expressa por "números, valores lógicos, letras, palavras e frases" (ICMC-USP, 2011).

5 - Palavras reservadas

O que lhe vem à mente quando lê o termo: "palavras reservadas"?

As palavras reservadas são nomes utilizados pelo Algoritmo que tem um sentido predeterminado. Portanto, não podem ser redefinidas pelo usuário como identificadores ou utilizados de outra forma senão para a que foram criadas.

Vamos identificar algumas delas:

Quadro 3.8 - Palavras reservadas

se	de	até
então	declare	escreva
senão	fim-algoritmo	procedimento
enquanto	fim-se	função
faça	fim-enquanto	início
repita	fim-para	fim
até que	para	inteiro
real	caractere ou literal	lógico

Fonte: acervo pessoal.

Em sites de busca você pode localizar a lista de palavras reservadas das diferentes linguagens de programação, tais como: Pascal, Java, C, C++, MySQL, Acess, etc. Desse modo, sugerimos que realize pesquisas utilizando o termo "palavras reservadas" como palavra-chave. Nessa ocasião, procure verificar a consistência dos sites pesquisados e das informações neles disponibilizadas, antes de considerá-las como verdadeiras. Lembre-se de que uma simples pesquisa realizada de forma crítica pode ser uma ótima ferramenta de aprendizagem e trabalho!

Retomando a aula

Vamos recordar:

1 - Portugol

Na primeira Seção construímos conhecimentos sobre

o Portugol (Português Estruturado). Vimos que ele é uma pseudolinguagem de programação que utiliza comandos e instruções em Português para representar as ações dos Algoritmos.

2 - Tipos de dados e **3 - Operadores**

Em prosseguimento, reconhecemos e compreendemos os dados numéricos (inteiros e real), caracteres (valores alfabéticos ou alfanuméricos) e lógicos (valores verdadeiros ou falsos), bem como os operadores aritméticos, relacionais e lógicos, os quais podem ser fornecidos para possibilitar a formação de vários tipos de expressões.

4 - Constantes e **5 - Palavras Reservadas**

Para finalizar, estudamos a constante, ou seja, um identificador que armazena um valor fixo e as palavras reservadas, as quais são nomes utilizados pelo Algoritmo que tem um sentido predeterminado e, portanto, só podem ser utilizadas da forma como foram criadas.

Lembre-se que cada Aula estudada representa um importante passo para sua aprendizagem. Dessa forma, é imprescindível que, antes de iniciar a Aula 4, elimine todas as eventuais dúvidas que podem ter em relação ao conteúdo estudado na Aula 3. Para tanto, esperamos sua participação nas ferramentas "fórum", "quadro de avisos" ou "chat". Participe!

Vale a pena

Vale a pena **ler**

EVARISTO, Jaime; CRESPO, Sérgio. *Aprendendo a programar programando numa linguagem Algorítmica executável (ILA)*. 2. ed. Rio de Janeiro: Book Express, 2010.

GUIMARÃES, Ângelo de Moura; LAGES, Newton Alberto de Castilho. *Algoritmos e estrutura de dados*. Rio de Janeiro: LTC, 1994.

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de Oliveira. *Algoritmos* - lógica para desenvolvimento de programação. 2. ed. São Paulo: Érica, 2007.

SEBESTA, Robert W. *Concepts of programming languages*. 4th ed. USA: Addison-Wesley, 1999.

Vale a pena **acessar**

EXPERT.NET. *Homepage*. Disponível em: <http://www.tiexpert.net/programacao/Algoritmo/tipos-de-dados.php>. Acesso em: 28 jun. 2011.

BRASIL ESCOLA. *George Boole*. Disponível em: <http://www.brasilescola.com/biografia/george-boole.htm>. Acesso em: 28 jun. 2011.

ICMC-USP. *Algoritmos*. Disponível em: <http://www.icmc.usp.br/~sce180/sce180-2/Aulas/Algoritmos.pdf>. Acesso em: 28 jun. 2011.

NOBIOS. *Operadores aritméticos, relacionais, lógicos e de atribuição*. Disponível em: <http://everson.com.br/Operadores-Aritmeticos-Relacionais-Logicos-de-Atribuicao>. Acesso em: 28 jun. 2011.

SIEBRA, S. A. *Introdução à programação*. Disponível em: <http://pt.scribd.com/doc/50982387/4/Unidade-2-%E2%80%93-Linguagem-Algoritmica>. Acesso em: 28 jun. 2011.

Vale a pena **assistir**

FERNANDES, G. B. Aula 5 – Programação utilizando o Portugol. Disponível em: <http://www.scribd.com/doc/53213916/Aula-05-Introducao-ao-Portugol-VISUAL>. Acesso em: 28 jun. 2011.

Minhas anotações

4º Aula

Gramática do portugol

Caro(a) aluno(a), que tal conhecermos a "Gramática do Portugol"?

Pois bem, nesta Aula, você será convidado(a) a construir conhecimentos justamente sobre esse tema. Vamos, ainda, reconhecer a importância da referida gramática e aplicá-la na análise de sistemas. Note que não é uma Aula sobre como programar, nem tem como intuito ensinar algoritmos. Portanto, é esperado que você já esteja familiarizado com os conteúdos tratados nas Aulas 1, 2 e 3. Assim, caso considere necessário releia os conteúdos anteriormente estudados e revise-os!

Lembre-se, ainda, de que caso surjam dúvidas no decorrer dos estudos, estaremos esperando sua participação nas ferramentas "chat", "quadro de avisos" ou "fórum". Aceite o desafio! Acesse o ambiente virtual e continue fazendo parte dessa comunidade colaborativa de construção de conhecimentos!

Boa aula!

Objetivos de aprendizagem

Ao término desta aula, o aluno será capaz de:

- definir, introduzir e conceituar o termo: "Gramática de Portugol";
- reconhecer a importância da Gramática de Portugol para a análise de sistemas;
- aplicar os comandos básicos da Gramática de Portugol na programação.

Seções de estudo

1 - Definição e sintaxe geral de um algoritmo
2 - Variáveis (e constantes) da gramática do portugol
3 - Comandos básicos

1 - Definição e sintaxe geral de um algoritmo.

Vamos iniciar nossas reflexões sobre Gramática do Portugol aprendendo seu conceito, importância e aplicações no contexto da programação e da análise de sistemas.
Sugerimos que durante a leitura desta e das demais Seções da Aula 4, anote suas indagações, sínteses e reflexões. Esse hábito, além de contribuir para organização de seus estudos, ajudará a manter um registro de toda a sua trajetória acadêmica, percebendo com mais clareza eventuais dificuldades (para saná-las) e avanços (para comemorá-los).
Bons estudos!

Para entender a Gramática do Portugal, vamos conceituar alguns termos importantes relacionados a ela!

Leland L. Beck afirma que "a **gramática de uma linguagem de programação** é uma descrição formal da sintaxe, ou forma, dos programas e instruções individuais escritas nessa linguagem" (BECK, 1997, *grifo nosso*).

Nesse contexto, Gilvan Vilarim defende que:

> A **Sintaxe** é o nome dado ao conjunto de regras a serem seguidas para a escrita dos Algoritmos. Do mesmo modo que em nossa língua, precisamos seguir algumas regras para escrita, uniformizando os Algoritmos e facilitando sua posterior codificação em programas, caso seja necessário. Já a semântica refere-se ao que é efetuado pelo computador quando ele encontra um comando. "Portanto, se a sintaxe está relacionada à forma de um comando, a **semântica** está relacionada ao seu conteúdo" (VILARIM, 2004, *grifo nosso*).

Reconhecendo a definição da gramática de uma linguagem de programação, da sintaxe e da semântica, é preciso lembrar que, como você já estudou na Aula 3, o Portugol (Português Estruturado) é constituído de letras maiúsculas e minúsculas ("A" - "Z", "a" - "z"), caractere sublinhado ("_"), os dígitos de 0 a 9 e os símbolos especiais + - * / = < > ( ) { } . , ; : '.

A Gramática do Portugol com sua sintaxe e semântica é representada por "um conjunto de palavras (instruções) limitadas e reconhecidas como comandos, que tem regras fixas e não conflitantes de uso, o que significa que uma instrução tem apenas um significado para um compilador [...]" (INFORMÁTICA EDUCATIVA, 2011).

CONCEITO

Para conceituar a sintaxe geral de um Algoritmo, veja o exemplo a seguir. Nele, utilizamos uma sintaxe padrão estabelecida por vários autores em iniciação de Algoritmos.

É válido salientar que essa sintaxe pode variar, todavia, o padrão estabelecido por nós no decorrer deste curso será o que utiliza o programa "VisuAlg" (que é um interpretador de Algoritmos). Com isso, poderemos testar todos os Algoritmos exemplificados, tornando bem prático o entendimento do Algoritmo e seu funcionamento.

Segue a sintaxe:

```
algoritmo "Nome do Programa"
var //Variáveis → Declaração de variáveis do Algoritmo
      <variável01>: <tipo1>
      <variável02>: <tipo2>
// Corpo do Algoritmo à Isto indica um comentário
inicio //inicio sem acentuação, pois o VisuAlg não
//suporta acentuação
      <comando1>
      <comando2>
      <comando3>
fimalgoritmo
```

E aí, você já conseguiu formar uma ideia sobre o que vem a ser a sintaxe geral de um algoritmo? Em caso de resposta negativa, não se preocupe: nosso estudo está apenas começando e com os conteúdos das próximas Seções será mais fácil compreendê-la. Além disso, você pode ampliar seus conhecimentos, consultando as obras, periódicos e sites indicados ao final desta Aula. Seja persistente! Passemos, agora, para o estudo da Seção sobre as Variáveis!

2 - Variáveis (e constantes) da gramática do portugol

Variáveis também são identificadores que armazenam valores, porém, ao contrário das constantes, o valor de uma variável pode mudar dentro do Algoritmo.

As variáveis, assim como as constantes, podem ser dos tipos: numérica (inteiro ou real), lógica ou literal.

Observe que o conceito de variável talvez seja o mais importante na confecção de um Algoritmo, uma vez que um Algoritmo utiliza, quase sempre, uma declaração de variável.

Além disso, por definição, tudo aquilo que é sujeito a variações, que é incerto, instável ou inconstante, em um programa de computador são os dados a serem processados.

2.1 - Nomeando constantes e variáveis

Os nomes de constantes e variáveis (identificadores) são nomes simbólicos para os objetos referenciados nos Algoritmos. Esses nomes são escolhidos pelo usuário para representar endereços de memória onde vão ser alocadas as informações.

O nome da variável, por exemplo, é utilizado para sua identificação e representação dentro de um programa de computador, sendo os identificadores formados por um ou mais caracteres, os quais devem seguir algumas regras:

1) começar sempre por um caractere alfabético ou o símbolo "_" (sublinhado);
2) ser constituídos de caracteres alfabéticos ou numéricos;
3) não podem conter caracteres especiais, como por

exemplo: + - * / = < > ( ) { } . , ; : ';

4) não podem ter o mesmo nome que comandos ou palavras reservadas do Algoritmo ou da linguagem de programação que será usada.

Para entender melhor, vejamos como acontece a definição de um identificador na Figura 4.1!

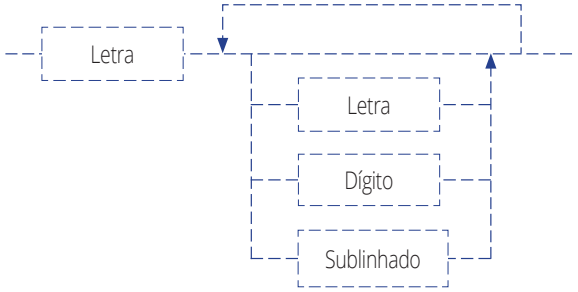


Figura 4.1 Palavras reservadas.
Fonte: acervo pessoal.

Vale salientar que no nome dos identificadores não há distinção entre letras maiúsculas ou minúsculas.

Seguem exemplos de nomes permitidos:

a) cliente b) delta c) X d) BC4R e) MEDIA
f) NOTA_ALUNO g) A h) A123B

Agora, seguem exemplos de nomes não permitidos:

a) 5X b) E(13) c) X-Y d) NOTA/2 e) 123TESTE
f) NOME DO CLIENTE g) valor_em_R$

Geralmente colocamos o nome da variável com aquilo que ela será relacionada na resolução do Algoritmo.

Por exemplo:

var

nome → indica que esta variável receberá um nome - tipo caractere.

endereço → indica que esta variável receberá um endereço - tipo caractere.

soma → indica que esta variável receberá um resultado de soma - tipo real ou inteiro.

idade → indica que esta variável receberá um número – tipo inteiro.

2.2 - Declaração de Variáveis

As variáveis devem ser declaradas logo no início do Algoritmo (ou no início do procedimento ou função como veremos mais adiante). A declaração de variáveis deve iniciar pela palavra reservada "Var" e segue a sintaxe apresentada na figura a seguir:

Ufa!
Isso parece complicado, mas na figura a seguir está tudo explicadinho. Vejam.

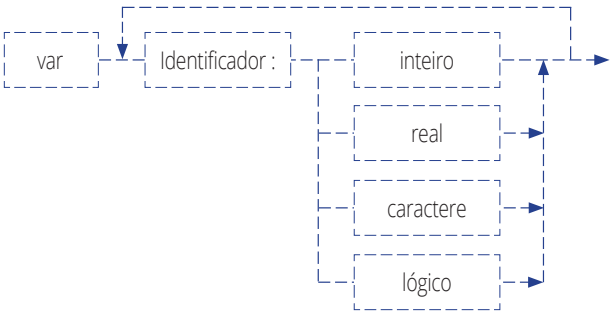


Figura 4.2 Diagrama da definição de uma variável.
Fonte: acervo pessoal.

Exemplo:

```
algoritmo "Exemplo"
var
        VARIÁVEL_1, VARIÁVEL_2: inteiro
        VARIÁVEL_3, VARIÁVEL_4: real
inicio
        leia (VARIÁVEL_1, VARIÁVEL_2)
        leia (VARIÁVEL_3, VARIÁVEL_4)
fimalgoritmo
```

Uma variável só pode receber valores do mesmo tipo declarado, ou seja, variáveis do tipo inteiro só podem receber valores inteiros e assim por diante.

```
algoritmo "Exemplo2"
var
        A, B: inteiro
        SOMA: real
inicio
        leia (A,B)
        SOMA <- A + B
        escreva (SOMA)
fimalgoritmo
```

Observe um detalhe relevante: "a declaração de variáveis deve ser feita dentro de um bloco específico, o qual precisa aparecer logo após a declaração do Algoritmo. O bloco é iniciado pelas palavras-chave 'variáveis' (sim, com acento) e termina com a palavra-chave 'fim-variáveis'. Pelo menos uma variável deve ser declarada dentro do bloco (embora o bloco em si seja opcional) e apenas um bloco em escopo global deve ser declarado" (GPT.BERLIOS, 2011).

Que "variáveis não podem ter caracteres especiais? Portanto, identificadores (nomes de variáveis, funções e do Algoritmo) não podem ter acentos ou caracteres especiais como $, #, etc. A definição de um identificador em G-Portugol é equivalente ao das linguagens populares: uma letra (a-z ou A-Z) seguido de qualquer número de letras ou número" (GPT.BERLIOS, 2011).

3 - Comandos básicos

CONCEITO

Os comandos especificam as ações a serem realizadas pelo computador, como comparações e atribuições. Eles se

constituem por expressões, palavras-chave e operadores.

3.1 - Atribuição

Para atribuirmos um valor a uma variável usaremos o símbolo de atribuição ←. Sua sintaxe é apresentada na figura a seguir:

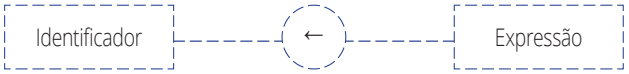


Figura 4.3 Diagrama da atribuição de valores a identificadores.
Fonte: acervo pessoal.

Exemplo: VALOR ← (TOTAL1+TOTAL2) * (A/B)

Nesse exemplo, a variável VALOR receberá o resultado do cálculo da expressão da direita.

3.2 - Expressões

As expressões são constituídas por constantes, variáveis e operadores, que definem o método para calcular um valor. O resultado das expressões pode ser armazenado em uma variável de um tipo compatível com o valor resultado.

São exemplos de expressões:

a > b

Resultado <- 3 + (x * y)

D + num / alfa

DIA <- 30

3.3 - Comentários

Os comentários, cuja importância é indiscutível dentro de um Algoritmo ou programa, são delimitados por "{ }" (chaves) ou "//" (duas barras) e podem aparecer em qualquer lugar do Algoritmo.

Observe que dentro dos delimitadores de comentário não pode haver outros delimitadores, ou seja, não são permitidos comentários aninhados. Os comentários não influenciarão no funcionamento do programa. Portanto, recomendamos a utilização constante dos comentários, pois estes não afetarão em nada no programa, exceto promover maiores esclarecimentos sobre determinadas partes do Algoritmo.

Exemplos de comentários:

```
leia (a,b) //Leitura das variáveis A e B
escreva ("A soma de A e B =",soma)  //Escreve a soma
//dos valores informados pelo usuário
```

3.4 - Comandos de entrada e saída

Todo Algoritmo requer alguma forma de entrada e saída de dados e informações. Um Algoritmo não teria razão se não houvesse possibilidade de entrar com valores para processamento ou mostrar os resultados desse processamento. Em alguns Algoritmos pode ser necessário incluir alguns dados de entrada.

Esses dados serão processados e os resultados serão mostrados ao usuário. Uma das maneiras de exteriorizar os resultados de determinado processamento é por meio da impressão desses resultados. Para que isso seja possível, temos o comando de entrada: **leia**; e o comando de saída: **escreva**.

Para interpretar o Portugol, podemos imaginá-lo como a estrutura de uma árvore (estrutura de dados) que estabelece a sequência em que o Algoritmo deve ser executado.

Comando Leia

Esse comando permite que o usuário informe dados de entrada para o Algoritmo. A sintaxe desse comando é definida na figura a seguir:

Figura 4.4 Diagrama de fluxo do comando LEIA.
Fonte: acervo pessoal.

Exemplos:

```
leia (VALOR)
leia (A, B, CONTADOR)
leia (CÓDIGO, NOME)
```

O comando **leia** lê as variáveis da lista e mantém o cursor preparado de tal forma que o próximo comando **leia**, se houver, vai procurar o valor a ser lido na mesma linha.

```
algoritmo "exemplo 01"
var //Declaração de Variáveis
   NOME: caractere //Variável nome tipo caractere
   NOTA1, NOTA2: real //variável nota1 e nota 2 tipo
   //real

inicio // Início do Corpo do Algoritmo
   leia (NOME) //entre com o Nome da Pessoa, via
   //teclado, ex: Cristiane
   leia (NOTA1) //entre com a nota1, via teclado, ex: 5
   leia (NOTA2) //entre com a nota2, via teclado, ex: 10
   escreva (NOME," ",NOTA1," ",NOTA2) //escreva –
   //Cristiane 5  10, no Monitor
fimalgoritmo //Fim do Algoritmo
```

Comando Escreva

Esse comando permite apresentar ao usuário mensagens e dados contidos em identificadores. A sintaxe do comando de saída é apresentada na figura 4.5:

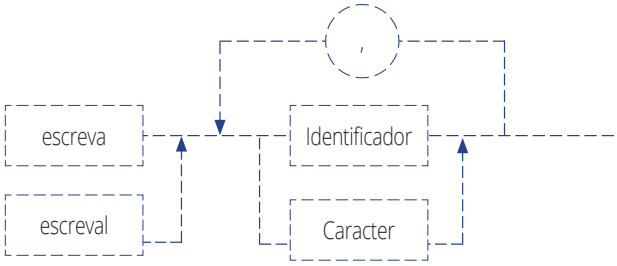


Figura 4.5 Diagrama de fluxo do comando ESCREVA.
Fonte: acervo pessoal.

Exemplos:

```
escreva (SOMA)
escreva ("Media = ",MED)
escreva ("Valor total = ",VALOR)
```

Para saltar uma linha utiliza-se o comando "Escreval".

Um Algoritmo simples ilustrando os comandos leia e escreva é mostrado a seguir:

```
algoritmo "Exemplo 02"
var
   A, B, SOMA: inteiro
inicio
   leia (A) //entre com valor de A ex: à 5
   leia (B) //entre com valor de B ex: à 10
   SOMA <- A + B //SOMA recebe 5+10 = 15
   escreva ("A soma de A e B = ",SOMA) //escreve – A
   //soma de A e B=15
fimalgorimo
```

"O ponto fundamental que guia as diretrizes da linguagem G-Portugol é seu propósito educacional: ela deve expressar processos computacionais de forma que um leigo os compreenda sem enfatizar a si mesmos. Isto é, a linguagem em si deve chamar o mínimo de atenção possível, fazendo com que a compreensão dos processos computacionais seja tão natural quanto ler sua descrição informal, ou não estruturada" (GPT.BERLIOS, 2011).

Iremos avaliar vários Algoritmos, neles executaremos um procedimento chamado teste de mesa que significa seguir as instruções do Algoritmo de maneira precisa para verificar se o procedimento utilizado está correto ou não.

Exemplo 03: entrar com três variáveis, A e B, fazer as seguintes expressões de soma C=(B+A)+2, B=(C*(A+3) e A=(B+C), e escreva o resultado de A, B e C.

```
algoritmo "Exemplo 03"
var
    A, B, C: real
inicio
     A <- 100
     B <-  200
     C <-  (B/A) +2
     B <-  C*(A+3)
     A <-  B+C
     escreva (A, B, C)
fimalgoritmo
```

Visualize no teste de mesa:

Teste de Mesa (variáveis A, B, C)

A	B	C
100		
	200	
		4
	412	
412		
416	**412**	**4**

Figura 4.6 Teste de Mesa (variáveis A, B, C).
Fonte: acervo pessoal.

A vantagem em se testar o Algoritmo em teste de mesa é que se houver erro, este será detectado rapidamente. Em outras palavras, ele facilita a identificação do local do erro e consequentemente sua correção. Por essa razão, também é chamado de teste exaustivo (NIEE – UFRGS, 2011).

Exemplo 04: **entrar** com as variáveis X e Y e **calcular** as expressões e **escrever** o resultado de X, Y e Z.

```
algoritmo "Exemplo 04"
var
    X, Y, Z: real
inicio
    X  <-  2,5
    Y  <-  3,5
    Z  <-  ((X + Y) - 2) * (3 + X)
    Y  <-  Y + 1
    X  <-  Y + 1
    Y  <-  X + Y
    Z  <-  Z + Y + X
   escreva (X,Y,Z:2:2)//nesse caso estamos definindo
    //as casas decimais.
fimalgoritmo
```

Teste de Mesa (variáveis X, Y, Z)

X	Y	Z
2.5		
	3.5	
		22
	4.5	
5.5		
	10	
		37.5
5.5	10	37.5

Figura 4.7 Teste de Mesa (variáveis X, Y, Z).
Fonte: acervo pessoal.

Exemplo 05 - Entre com as variáveis de S e R e calcule as expressões, U=S*R e X=U/5+R e escreva o valores de S,R,U e X.

```
algoritmo "Exemplo 05"
var
      S, R, U: inteiro
      X: real
inicio
      S <- 3
      R <- S
      U <- S * R
      X <- U / 5 + R
      escreva (S, R, U, X)
fimalgoritmo
```

Observe o teste de mesa:

S	R	U	X
3			
	3		
		9	
			4.8
3	3	9	4.8

Figura 4.8 Teste 2 de Mesa (variáveis X, Y, Z).
Fonte: acervo pessoal.

Exemplo 06 - Ler uma temperatura em graus Celsius e escrevê-la convertida em graus Fahrenheit. A expressão de conversão é F=(9*c+160)/5, sendo F a Temperatura em Fahrenheit e C a temperatura em Celsius.

```
algoritmo "Farenheit"
var
        C, F: real
inicio
        leia (C)
        F <- (9*C+160)/5
        escreva ("Resultado:", F)
fimalgoritmo
```

Teste de Mesa (variáveis C, F)

C	F
10	
	50
	50

Figura 4.9 Teste de Mesa (variáveis C, F).
Fonte: acervo pessoal.

Lembre-se de que em sites como Scielo, Google Acadêmico, entre outros, você pode encontrar artigos sobre os comandos básicos da Gramática do Portugol. Assim, é importante que realize buscas, procure ler textos relacionados e amplie seu conhecimento em relação ao conteúdo. O hábito da pesquisa será fundamental para sua formação e para o dia a dia de sua profissão!

Exemplo 07 – Ler quatro valores referentes a quatro notas escolares de um aluno e imprimir a média da nota do aluno.

```
algoritmo "Média"
var
        media, P1, P2, P3, P4: real
inicio
        leia (P1)
        leia (P2)
        leia (P3)
        leia (P4)
        media <- (P1 + P2 + P3 + P4) / 4
        escreva ("Media:", media)
fimalgoritmo
```

Teste de Mesa (Média, P1, P2, P3, P4)

M	P1	P2	P3	P4
	10			
		10		
			5	
				8
8,25				
8,25				

Figura 4.10 Teste de Mesa (Média, P1, P2, P3, P4).
Fonte: acervo pessoal.

Exemplo 08 – Escrever um Algoritmo para determinar o consumo médio de um automóvel sendo fornecidos a distância total percorri da pelo automóvel e o total de combustível gasto.

```
algoritmo "consumo"
var
        dist, comb, consumo: real
inicio
        escreva ("Digite em Kms")
        leia (dist)
        escreva ("Digite o Combustível")
        leia (comb)
        consumo <- dist/comb
        escreva ("Consumo:",consumo)
fimalgoritmo
```

Teste de Mesa (Média, P1, P2, P3, P4)

Consumo	Dist	Comb
	10	
		05
2		
2		

Figura 4.11 Teste 2 de Mesa (Média, P1, P2, P3, P4).
Fonte: acervo pessoal.

Nesse caso, o consumo médio do carro foi de 2 km por litro de gasolina.

Lembre-se de que em caso de dúvidas, você não precisa se "afobar", procure entrar em contato conosco por meio do ambiente virtual e seguir as dicas recomendadas ao longo da Aula.

Exemplo 09 – Elaborar um Algoritmo que calcule e apresente o volume de uma caixa retangular, utilizando a fórmula:

volume= comprimento * largura * altura.

```
algoritmo "volume"
var
        vol, comp, larg, alt: real
inicio
        escreva ("Entre com comprimento")
        leia (comp)
        escreva ("Entre com a largura")
        leia (larg)
        escreva ("Entre com a altura")
        leia (alt)
```

```
        vol <- ((comp * larg)*alt))
        escreva ("O Volume é:", vol)
fimalgoritmo
```

Como você pode observar, nesse caso foi colocado um parêntese a mais no Algoritmo. Isso indica que o primeiro cálculo será do parêntese mais interno (comp*larg) em seguida será multiplicado o resultado vezes a altura.

Retomando a aula

Estamos indo bem, não é mesmo? Então, para encerrar esse tópico, vamos relembrar os conteúdos estudados na Aula 4:

1 - Definição e sintaxe geral de um algoritmo

Iniciamos a construção da aprendizagem dos conteúdos sobre a Gramática do Portugol, estudando a sintaxe geral de um Algoritmo.

2 - Variáveis (e constantes) da gramática do portugol

Na Seção 2, vimos que as variáveis, talvez o conceito mais importante na elaboração de um algoritmo, são identificadores que armazenam valores, os quais não podem mudar dentro do Algoritmo. Identificamos que elas podem ser numéricas (inteiro ou real), lógicas ou literais e tivemos a oportunidade de entender a forma como acontece a nomeação das constantes e das variáveis.

Em seguida, pudemos perceber que as variáveis devem ser declaradas logo no início do Algoritmo (ou no início do procedimento ou da função).

3 - Comandos Básicos

Finalmente, estudamos os comandos que se constituem por operadores, expressões e palavras-chave e especificam as ações a serem realizadas pelo computador, como comparações e atribuições. Estudamos, portanto, a atribuição, as expressões, os comentários e os comandos de entrada e saída.

Vale a pena

Vale a pena ler

BERG, Alexandre; FIGUEIRÓ, Joice Pavek. *Lógica de programação*. 3 ed. Canoas: Ulbra, 2006.

SAID, Ricardo. *Curso de lógica de programação*. São Paulo: Universo dos livros, 2007.

Vale a pena acessar

INFORMÁTICA EDUCATIVA. *Portugol/plus:* uma ferramenta de apoio ao ensino de lógica de programação baseado no portugol. Disponível em: <http://www.c5.cl/ieinvestiga/actas/ribie98/118.html>. Acesso em: 28 jun. 2011.

GPT.BERLIOS. *Programando em Portugol*. Disponível em: <http://gpt.berlios.de/manual_nodes/node4.html>. Acesso em: 28 jun. 2011.

NIEE – Núcleo de Informática na Educação Especial. Homepage. Disponível em: <http://www.niee.ufrgs.br>. Acesso em: 28 jun. 2011.

SILVA, T. G-Portugol. Disponível em: <http://gpt.berlios.de/manual_big/manual.html#SECTION00400000000000000000>. Acesso em: 28 jun. 2011.

NIEE-UFRGS. *Interpretador de Portugol*. Disponível em: <http://www.niee.ufrgs.br/eventos/CBCOMP/2004/pdf/Algoritmos/t170100165_3.pdf>. Acesso em: 28 jun. 2011.

Vale a pena assistir

FRAMOS, R. Português estruturado – portugol. Disponível em: <http://www.youtube.com/watch?v=xncTAg00fsk>. Acesso em: 28 jun. 2011.

Minhas anotações

Aula 5º

Estruturas condicionais

Prezado(a) aluno(a), iniciaremos a Aula 5 adquirindo algumas noções básicas de Estruturas Condicionais. Estruturas como essas são conhecidas como estruturas de decisão ou de seleção e se caracterizam pela execução de determinados códigos de programação dependendo da veracidade de uma condição. É a estrutura que permite a tomada de decisão, em um Algoritmo, mediante a análise lógica de uma condição.

Com o conhecimento deste conceito poderemos estabelecer condições para que comandos sejam executados ou não. Será fundamental para entendermos o raciocínio utilizado na solução dos problemas em informática, acadêmicos, profissionais ou outros. Interessante, não é mesmo?

Então, para ampliar ao máximo sua capacidade de construir conhecimentos sobre as estruturas condicionais, é importante que organize conscientemente o material que pretende estudar e planeje o tempo que lhe dedicará. Sempre será um tempo estimativo, porque haverá temas que consumirão mais tempo do que o esperado e outros, muito menos.

Pense nisso e boa aula!

Boa aula!

Objetivos de aprendizagem

Ao término desta aula, o aluno será capaz de:

- definir Estruturas Condicionais, reconhecer sua importância e aplicá-la em análise de sistemas;
- conceituar e introduzir o termo: Estruturas Condicionais;
- desenvolver e/ou ampliar a capacidade de percepção da condição da estrutura e utilizá-la.

Seções de estudo

1 - Estruturas condicionais

1 - Estruturas condicionais

Até o presente momento, utilizamos as estruturas de processamento de entrada e saída que compõem os Algoritmos puramente sequenciais. Isso quer dizer que aprendemos a utilizar as variáveis, as constantes e os operadores aritméticos.

Se utilizarmos como referência o exemplo do cálculo da média dos alunos, é possível perceber que foram usados apenas as entradas simples das notas e o cálculo da média. Desse modo, será necessária a aplicação de novas estruturas, se desejarmos que o Algoritmo também nos apresente uma mensagem dizendo "se" o aluno foi aprovado ou reprovado, no caso de sua nota ter sido maior ou igual a 7. Em outras palavras, são necessárias as estruturas condicionais.

Qual a finalidade de usarmos as Estruturas Condicionais?

1.1 - Desvio condicional simples (se.....então)

Esta é a estrutura básica de controle em quase todas as linguagens de programação.

Essa instrução tem por finalidade representar a tomada de uma decisão. Esse comando nos fornece a habilidade de fazer uma decisão simples, se uma dada condição for verdadeira.

Sintaxe:

```
se <condição> entao
        <comando1>
        <comando2>
        .....
        <comandon>
fimse
```

Note que o bloco de comandos será executado se a condição for verdadeira. Caso a condição seja falsa, a execução do Algoritmo não executará o bloco de comandos e passará o controle para a linha imediatamente após o "fim-se".

Na Figura 5.1 você poderá entender melhor o que estamos afirmando. Observe-a com atenção!

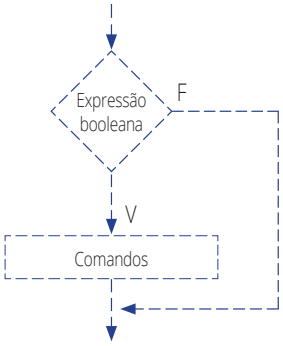


Figura 5.1 Diagrama de fluxo SE... ENTÃO. Fonte: acervo pessoal.

Exemplo 10 – Ler quatro valores referentes a quatro notas escolares de um aluno e imprimir a média da nota do aluno. Se a média for maior que 7 escreva a média e a frase "aluno aprovado".

```
algoritmo "Média"
var
        media, P1, P2, P3, P4: real
inicio
        leia (P1)
        leia (P2)
        leia (P3)
        leia (P4)
        media <- (P1 + P2 + P3 + P4) / 4
        se (media >= 7) entao
            escreva (media,"Aluno Aprovado")
        fimse
fimalgoritmo
```

Veja como fica com o teste de mesa:

Teste de Mesa (Média, P1, P2, P3, P4)

M	P1	P2	P3	P4
	10			
		10		
			5	
				8
8,25				
8,25				

Figura 5.2 Teste de mesa (Média, P1, P2, P3, P4). Fonte: acervo pessoal.

Note que quando chegarmos à comparação (Média >=7), o programa irá analisar: se a média for maior ou igual a 7 (verdadeiro) ele irá imprimir a média, caso contrário, irá sair sem nenhum valor.

Exemplo 11 – Entre com dois valores inteiros, efetue a soma desses valores, e caso a soma seja maior que 10, escreva esse valor.

```
algoritmo "Soma"
var
        A, B : inteiro
        X: real
inicio
        leia (A)
        leia (B)
        X <- (A + B)
        se ( X >= 10) entao
          escreva ("Valor de X:", X)
        fimse
fimalgoritmo
```

E aí, como está sua aprendizagem sobre as Estruturas Condicionais? Está achando simples ou complicado? Independentemente de sua resposta, é importante lembrar que no decorrer do estudo desta e das demais Aulas, você terá a oportunidade de entender cada vez melhor os conteúdos estudados. Para tanto, é fundamental que além de ler esse material, faça pesquisas, consulte referências e interaja com seus

colegas de curso e com seu professor. Afinal, você é o personagem principal de sua aprendizagem! Conte conosco nesta empreitada e participe!

1.2 - Desvio condicional composto (se.....então.....senão)

A estrutura de decisão à qual nos referimos neste tópico é usada quando a ação a ser executada depende de uma inspeção ou teste. Ela nos fornece a habilidade de executar um comando composto, se determinada condição for verdadeira ou falsa.

Para tanto, ela primeiro testa se a condição é verdadeira. Sendo verdadeira, executa um comando. Caso a condição seja falsa, o programa executará outro comando. Veja:

Sintaxe:

```
se <condição> entao
        <comando1>
        <comando2>
        ......
        <comandon>
senao
        <comando1>
        <comando2>
        ......
        <comandom>
fimse
```

Observe que a sequência de comandos do bloco "então" será executada caso a condição seja verdadeira. Se a condição for falsa, executará a sequência de comandos do bloco "senão".

Confira a Figura 5.3:

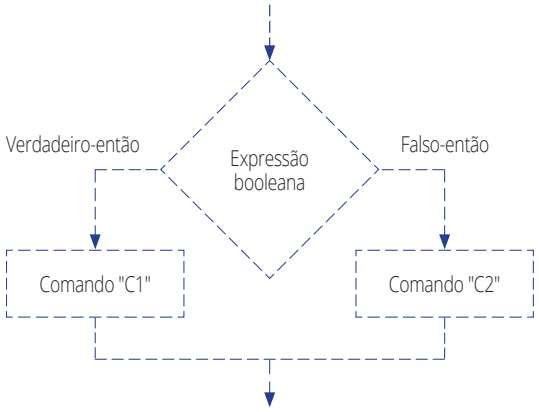


Figura 5.3 Diagrama de fluxo SE... ENTÃO... SENÃO.
Fonte: acervo pessoal.

Viu como a sintaxe de estrutura condicional é algo simples que você pode se habituar a fazer? Agora, para sedimentar sua aprendizagem, sugiro que pare um minuto, reflita sobre o que aprendeu para, depois, continuar o raciocínio no próximo exemplo:

Exemplo 12: dados dois números, determinar o maior entre eles:

```
algoritmo "MaiorNúmero"
var
        A, B: inteiro
inicio
        leia (A)
        leia (B)
        se A > B entao
           escreva ("A é maior que B")
        senao
           escreva (" é maior que A")
        fimse
fimalgoritmo
```

Tomem cuidado com a colocação das (' aspas simples) ou (" aspas duplas), pois para o interpretador VisuAlg existe diferença entre a colocação delas.

Exemplo 13: leia o salário e nome de dois funcionários e verifique qual salário é maior. Escreva o nome do funcionário com salário maior. Se for menor escreva "quero aumento":

```
algoritmo "Salário"
var
   func01, func02: caractere
   valor01, valor02: inteiro
inicio
        escreva ("Funcionário:")
        leia (func01)
        escreva ("Salário:")
        leia (valor01)
        escreva ("Funcionário:"(
        leia (func02)
        escreva ("Salário:")
        leia (valor02)
        se valor01 > valor02 entao
           escreva (func01)
        senao
           escreva ("Quero Aumento")
        fimse
fimalgoritmo
```

Atenção: esquecer uma ou duas chaves que delimitam um comando pode levar a erros de sintaxe ou erros de lógica de um programa! Lembre-se disso...

1.3 - Desvios condicionais encadeados

Desvios condicionais encadeados ocorrem quando necessitamos fazer algumas verificações lógicas de condições sucessivamente.

Nesses casos, uma determinada ação de um Algoritmo somente pode ser executada se um conjunto anterior de condições for verificado. Assim, quando uma ação é executada ela pode levar as outras condições, não havendo limites. Isso nos leva a uma estrutura encadeada.

Quando há um encadeamento de testes, o Algoritmo possui um comando de decisão dentro de outro. Isso fica localizado internamente ao Então ou ao Senão.

Sintaxe:

```
se <condição> entao
        <comando1>
senao
        se <condição> entao
           <comando1>
        senao
```

```
        <comando2>
    fimse
fimse
```

Ufa! Isso parece complicado, mas na figura abaixo está tudo explicadinho. Vejam.

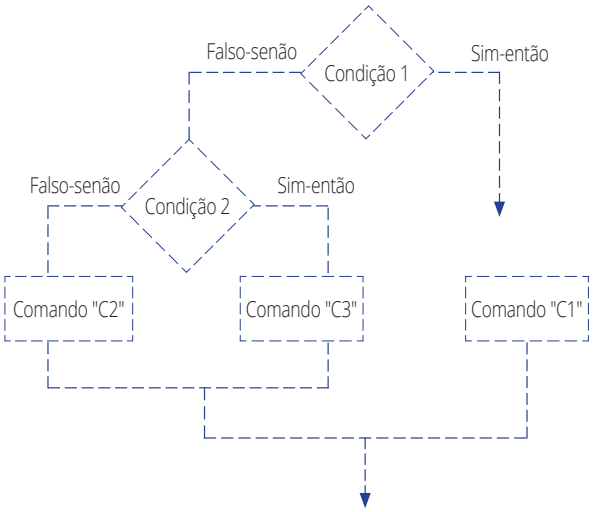


Figura 5.4 Diagrama de fluxo da estrutura.
Fonte: acervo pessoal.

Veja que nesse caso "se" testa primeiro a condição 1 e verifica se ela é verdadeira. Se for, então ele executa o "comando c1". Caso a condição 1 seja falsa, ele executará a "condição 2". Novamente ele verifica se ela é verdadeira. Se for, ele executa o "comando C3". Se for falso, ele executa o "comando C2".

Exemplo 14: dados dois números, determinar o maior entre eles, ou se eles são iguais:

```
algoritmo "Maior Número 2"
var
   A, B: inteiro
inicio
   leia (A)
   leia (B)
   se A > B entao
      escreva ("A é maior que B")
   senao
      se A = B entao
            escreva ("A é igual a B")
      senao
            escreva ("B é maior que A")
      fimse
   fimse
fimalgoritmo
```

O entendimento detalhado de um problema é um fator que diferencia uma solução algorítmica da outra. Assim, quanto mais se conhece sobre o problema em questão, maiores as chances de se criar soluções mais adequadas, eficientes e eficazes. Pense nisso...

Exemplo 15: considere o problema a seguir, no qual se estabelecem três condições para se calcular o reajuste de salário de um funcionário:

a) Para salário < do que 500, o reajuste será de 15%.
b) Para salário >= 500 mas <= 1000, o reajuste será de 10%.
c) Para salário > 1000, o reajuste será de 5%.

```
algoritmo "Salário"
var
      salario, Nsal : real
inicio
      leia (salario)
      se (salario < 500) entao
          Nsal <- (salario * 15)/100 + salario
      senao
        se ((salario>= 500) e (salario <= 1000)) entao
            Nsal <- (salario * 10)/100 + salario
        senao
            Nsal <- (salario * 5)/100 + salario
        fimse
      fimse
      escreva ("Seu Salário:", Nsal)
fimalgoritmo
```

Exemplo 16: considere o problema a seguir, entre com o número de matrícula de um aluno, faça um Algoritmo que imprima o mês de pagamento da anuidade, utilizando a expressão Final = Matrícula / 10. Sendo que:

a) Final= 0 – Janeiro.
b) Final=1 – Fevereiro.
c) Final=2 – Março.
d) Final= 3 – Abril.
e) Outro final – Maio.

```
algoritmo "Mensalidade"
var
   Matricula: inteiro
   Final: real
inicio
  escreva ("Forneça o número de Matrícula:")
  leia (Matrícula)
  Final <-  Matricula / 10
  se Final = 0 entao
      escreva ("Pagamento em Janeiro")
  senao
      se Final = 1 entao
          escreva ("Pagamento em Fevereiro")
      senao
          se Final = 2 entao
              escreva ("Pagamento em Março")
          senao
              se Final =3 entao
                  escreva ("Pagamento em Abril")
              senao
                  escreva ("Pagamento em Maio")
              fimse
          fimse
      fimse
   fimse
fimalgoritmo
```

Exemplo 17: entre com três números inteiros e calcule o maior número entre eles.

```
algoritmo "MaiorNúmero"
var
   A, B, C : inteiro
inicio
   escreva ("Entre com o primeiro número:")
   leia (A)
   escreva ("Entre com o segundo número:")
   leia (B)
   escreva ("Entre com o terceiro número:")
   leia (C)
   se (A > B) e (A > C) entao
        escreva ("O número maior é:", A)
   senao
        se (B > C) entao
              escreva ("O número maior é:", B)
        senao
              escreva ("O número maior é:", C)
        fimse
    fimse
fimalgoritmo
```

Compreenderam?
Condição: comparação que somente possui dois valores possíveis (verdadeiro ou falso);
Utiliza as palavras-chave (então, senão, fim se)
Simples, vocês não acham?

1.4 - Estrutura de seleção múltipla (Escolha.....caso)

O uso desta estrutura proporciona uma solução elegante quanto se tem vários desvios condicionais (SE.....ENTÃO..... SENÃO) encadeados. Ou seja, quando outras verificações são realizadas caso a anterior tenha falhado (ou seja, o fluxo do algoritmo entrou no bloco SENÃO). A proposta da estrutura ESCOLHA.....CASO é permitir ir direto ao bloco de código desejado, dependendo do valor de uma variável de verificação. Porém, esta estrutura só pode ser usada em situações onde a verificação é feita apenas com valores fixos e não em um intervalo.

Sintaxe:

```
escolha <variável de verificação>
caso <valor1>
      <comando1>
      <comando2>
      .....
      <comando n>
caso <valor2>
      <comando1>
      <comando2>
      .....
      <comandon>
caso <valor3>
      <comando1>
      <comando2>
      .....
      <comandon>
outrocaso
      <comando1>
      <comando2>
      .....
      <comandon>
fimescolha
```

Observe o esquema desta estrutura condicional.

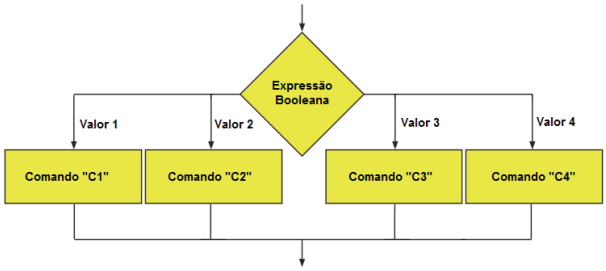


Figura 5.5 Diagrama de fluxo ESCOLHA...CASO.
Fonte: acervo pessoal.

Para exemplificar a vantagem oferecida por esta estrutura, imagine a seguinte situação: você deseja criar um algoritmo para uma calculadora, o usuário digita o primeiro número, a operação que deseja executar e o segundo número. Dependendo do que o usuário informar como operador, o algoritmo executará um cálculo diferente (soma, subtração, multiplicação ou divisão). Vejamos como seria esse algoritmo implementado com o desvio condicional SE.....ENTÃO.....SENÃO.

```
algoritmo "Calculadora1"
var
   numero1: real
   numero2: real
   operacao: caractere
   resultado: real
inicio
     escreva ("Digite o primeiro número: ")
     leia (numero1)
     escreva ("Digite a operação: ")
     leia (operacao)
     escreva ("Digite o segundo número: ")
     leia (numero2)
     se operacao = "+" entao
       resultado <- numero1 + numero2
     senao
       se operacao = "-" entao
         resultado <- numero1 - numero2
       senao
          se (operacao = "*") entao
           resultado <- numero1 * numero2
         senao
           se operacao = "/" entao
             resultado <- numero1 / numero2
           fimse
         fimse
       fimse
     fimse
     escreva ("Resultado: ", resultado)
fimalgoritmo
```

Observe como o desvio condicional encadeado deixa o código mais complexo. É possível a compreensão da lógica,

porém não é muito elegante. Agora vamos ver como ficaria a mesma lógica com a estrutura ESCOLHA.....CASO.

```
algoritmo "Calculadora2"
var
    numero1: real
    numero2 : real
    operacao : caracter
    resultado : real
inicio
    escreva ("Digite o primeiro número: ")
    leia (numero1)
    escreva ("Digite a operação: ")
    leia (operacao)
    escreva ("Digite o segundo número: ")
    leia (numero2)
    escolha operacao
      caso "+"
        resultado <- numero1 + numero2
      caso "-"
        resultado <- numero1 - numero2
      caso "*"
        resultado <- numero1 * numero2
      caso "/"
        resultado <- numero1 / numero2
     outrocaso
        escreva("Operação digitada inválida!")
    fimescolha
    escreval ("Resultado: ", resultado)
fimalgoritmo
```

Retomando a aula

Para encerrar a Aula 5, convido você a recordar o que estudamos aqui:

1 - Estruturas Condicionais

Na primeira e única Seção da Aula 5 você teve a oportunidade de construir conhecimentos sobre Estrutura Condicional, a qual é capaz de garantir que nosso pensamento proceda corretamente e que cheguemos a conclusões coerentes, podendo diferenciá-las por meio do programa entre verdadeiras e falsas.

Vimos ainda que a aplicação de Estruturas Condicionais na análise de sistemas é fundamental para a construção de Algoritmos e, por conseguinte, programas logicamente estruturados para atingir os objetivos almejados pelo(s) seu(s) criador(es).

É importante observar que os conhecimentos construídos nesta Aula são fundamentais e servem como base de muitos outros conhecimentos importantes para o profissional que atua ou pretende atuar na área de Análise de Sistemas.

Caso você tenha ficado com dúvidas sobre a Aula 5, acesse as ferramentas "fórum", "quadro de avisos" ou "chat" e interaja com seus colegas de curso e com seu professor. Lembre-se de que você faz parte de uma comunidade colaborativa de conhecimento... Portanto, estaremos esperando sua participação!

Vale a pena

Vale a pena ler

DEITEL, H. M. C++: *como programar*. Tradução de Carlos Arthur Lang Lisboa e Maria Lúcia Lang Lisboa. 3. ed. Porto Alegre: Bookman, 2001.

SENNE, Edson Luiz França. *Primeiro Curso de Programação em C. 2*. ed. Florianópolis: Visual Books, 2006.

Vale a pena acessar

SOUZA, E. Estruturas condicionais. Disponível em: <http://blog.ericksouza.com/31/>. Acesso em: 28 jun. 2011.

UFERSA - Universidade Federal Rural do Semi-Árido. Informática aplicada – aula 6. Disponível em: <http://www2.ufersa.edu.br/portal/view/uploads/setores/164/arquivos/InformaticaAplicada/Aula06_sequencial_apresentacao_1.pdf>. Acesso em: 28 jun. 2011.

WIKILIVROS. Estrutura condicional. Disponível em: <http://pt.wikibooks.org/wiki/Pascal/Estrutura_Condicional>. Acesso em: 28 jun. 2011.

Vale a pena assistir

GRAN CURSO. Episódio 44 - Estruturas condicionais. Disponível em: <http://www.youtube.com/watch?v=MrwWdS_HNzQ>. Acesso em: 28 jun. 2011.

Minhas anotações

6º Aula

Estruturas de repetição

Prezado(a) aluno(a), vamos começar a Aula de "Estruturas de Repetição", falando resumidamente sobre seu conceito: "em ciência da computação, uma estrutura de repetição é uma estrutura de desvio do fluxo de controle presente em linguagens de programação que realiza e repete diferentes computações ou ações dependendo se uma condição é verdadeira ou falsa, em que a expressão é processada e transformada em um valor. Estão associados a uma estrutura de repetição uma condição (também chamada 'expressão de controle' ou 'condição de parada') e um bloco de código: verifica-se a condição, e caso seja verdadeira, o bloco é executado. Após o final da execução do bloco, a condição é verificada novamente, e caso ela ainda seja verdadeira, o código é executado novamente" (SILVA, 2011).

Vamos entender melhor as Estruturas de Repetição no decorrer desta Aula e, como você poderá observar, a construção desse conhecimento será fundamental para entendermos o raciocínio utilizado na solução dos problemas em programação, o que poderá facilitar o entendimento do conteúdo de outras disciplinas e o cotidiano no ramo profissional.

Gostaríamos de lembrá-lo de que preparamos a Aula para facilitar sua aprendizagem. Contudo, é natural que sujam dúvidas no decorrer dela. Quando isso acontecer, acesse a plataforma e utilize as ferramentas "quadro de avisos" ou "fórum" para interagir com seus colegas de curso ou com seu professor.

Boa aula!

Objetivos de aprendizagem

Ao término desta aula, o aluno será capaz de:

- conceituar e identificar as "Estruturas de Repetição";
- reconhecer a importância das Estruturas de Repetição;
- desenvolver e/ou ampliar a capacidade de percepção de um problema e utilizar os conhecimentos construídos na criação de Algoritmos.

Seções de estudo

1 - Estruturas de repetição

1 - Estruturas de repetição

Vamos iniciar nossas reflexões estruturas de repetição, procurando compreender brevemente o seu conceito e tipologia.
Para facilitar sua aprendizagem lembre-se de estabelecer intervalos periódicos (de no mínimo 10 minutos) em meio aos seus estudos, uma vez que estudar muitas horas seguidas pode diminuir seu nível de atenção e seu ânimo. Aprender é muito bom, você não acha?
Então, organize-se e bons estudos!

Em alguns casos é necessário repetir uma parte do Algoritmo um determinado número de vezes. Para tanto, estão disponíveis as estruturas de repetição, que realizam o processamento de um determinado trecho tantas vezes quantas forem necessárias. Essa repetição também pode ser chamada de laço ou loop.

As estruturas de repetição podem ser divididas em duas:

a) Loops contados: quando se conhece previamente quantas vezes o comando composto no interior da construção será executado.

b) Loops condicionais: quando não se conhece de antemão o número de vezes que o conjunto de comandos no interior do laço será repetido, pelo fato dele estar amarrado a uma condição sujeita à modificação pelas instruções do interior do laço.

Ideia relevante: caso o bloco de código nunca modificar o estado da condição, a estrutura será executada para sempre, uma situação chamada laço infinito. Da mesma maneira, é possível especificar uma estrutura em que o bloco de código modifica o estado da condição, mas esta é sempre verdadeira (CENAPAD SP, 2011).

1.1 - Variáveis contadoras

Quando uma variável é contadora ela tem por característica armazenar dentro de si um número referente a certa quantidade de elementos ou iterações.

1.2 - Variáveis acumuladoras

Quando uma variável é acumuladora ela tem por característica armazenar dentro de si o resultado acumulado de uma série de valores.

Uma das principais características que consolidaram o sucesso na utilização dos computadores para a resolução de problemas foi a sua capacidade de repetir o processamento de um conjunto de operações para grandes quantidades de dados. Exemplos de conjuntos de tarefas que repetimos diversas vezes dentro de uma situação específica podem ser observados largamente no nosso dia a dia (FERRARI; CHEMINEL, 2011).

1.3 - Repetição condicional - Enquanto.....faça

A estrutura ENQUANTO.....FAÇA executa uma sequência de comandos repetidas vezes, enquanto uma determinada condição permanece válida (verdadeira).

Para tanto, a referida estrutura faz o teste da condição antes de iniciar a repetição; se o primeiro teste falhar, o bloco de instruções de comandos no seu interior não é executado nenhuma vez e a execução prossegue normalmente pela instrução seguinte ao Fimenquanto.

Nesse contexto, se a condição for verdadeira o comando composto é executado e ao seu término retorna-se ao teste da condição. Assim, o processo acima será repetido enquanto a condição testada for verdadeira. Quando esta for falsa, o fluxo de execução prosseguirá normalmente pelas instruções posteriores ao Fimenquanto.

Sintaxe:

```
enquanto <condição> faça
  <comando1>
  <comando2>
      .....
  <comandon>
fimenquanto
```

Note que uma vez dentro do corpo do laço, a execução somente o abandonará quando a condição for falsa. O usuário desse tipo de construção deve estar atento à necessidade de que em algum momento a condição deverá ser avaliada como falsa.

Caso contrário, o programa permanecerá indefinidamente no interior do laço, o que é conhecido como laço infinito. Confira a Figura 6.1:

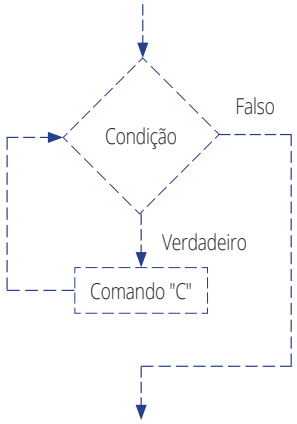


Figura 6.1 Diagrama de fluxo da estrutura ENQUANTO...FAÇA.
Fonte: acervo pessoal.

Exemplo 18: Faça um Algoritmo que escreva os números inteiros positivos menores que 100.

```
algoritmo "Repeticao01"
var
   NUMERO: inteiro
inicio
      NUMERO <-  0 // Inicia a variável contador = 0
      enquanto NUMERO < 100 faca
          escreval (NUMERO) // para pular uma linha
          //"l (letra L)'
          NUMERO <-  NUMERO + 1 // variável
          //acumuladora = numero=0+1
```

```
    fimenquanto
fimalgoritmo
```

Notem que a variável número foi utilizada também como contadora. Ela irá controlar o número de vezes que o trecho do programa deve ser executado. Neste exemplo ela foi iniciada com zero e no teste da condição zero é menor que 100. Logo, a condição é verdadeira, ele entrará no loop escreve "0", em seguida realiza o incremento da variável (NÚMERO=0+1).

O novo valor de número será "1", ele volta para a condição do enquanto e testa novamente "1 é menor que 100". Ele repetirá esses passos até que a condição seja falsa, ou seja, que o número seja 100.

Caso você tenha ficado com dúvidas sobre a repetição condicional ENQUANTO.....FAÇA ou sobre outros temas estudados nesta Aula, acesse as ferramentas "fórum", "quadro de avisos" ou "chat" e interaja com seus colegas de curso e com seu professor. Lembre-se de que é importante eliminar eventuais "pedras do caminho" para que sua aprendizagem siga seu curso da melhor forma possível! Passemos, a seguir, ao estudo de novos exemplos.

Exemplo 19: Mostrar a soma dos números pares inteiros positivos menores que 100.

```
algoritmo "InteiroPositivo"
var
    SOMA, NUMERO: inteiro
inicio
    NUMERO <- 2 // inicia a variável contadora
    SOMA <- 0
    enquanto NUMERO < 100 faca
        SOMA <- SOMA + NUMERO // expressão
        //soma=0+2
        NUMERO <-  NUMERO + 2 // variável
        //acumuladora numero=2+2
    fimenquanto
    escreva ("A soma dos números pares inteiros é:",
    SOMA)
fimalgoritmo
```

Exemplo 20: Segue um Algoritmo que escreve 10 vezes a frase "FLAMENGO". Confira:

```
algoritmo "Repetição"
var
  I : inteiro
inicio
    I <- 1 // Inicia a variável contador = 1
    enquanto I <= 10 faca
        escreval ("FLAMENGO") // Escreval – irá
        //pular uma linha
        I<- I + 1 // variável acumuladora = I=1+1
    fimenquanto;
fimalgoritmo
```

Exemplo 21: Faça um Algoritmo que leia um número inteiro positivo e apresente o resultado de sua tabuada de um a dez (1 a 10).

Supondo que o valor desejado seja 2, então:

```
2 x 1 = 2        2 x 2 = 4
2 x 3 = 6        2 x 4 = 8
2 x 5 = 10       2 x 6 = 12
2 x 7 = 14       2 x 8 = 16
2 x 9 = 18       2 x 10 = 20
```

Vejamos:

```
algoritmo "Tabuada"
var
    valor, contador: inteiro
inicio
    escreva ("Entre com o Valor:")
    leia(valor)
    contador <-  1 // Inicia a variável contador = 1
    enquanto contador <= 10 faca
        escreval (valor, " x ",contador," = ", (valor *
        contador) )
        contador <- contador +1 // variável
        //acumuladora = contador= 1+1
    fimenquanto
```

E aí, você já entendeu amplamente o que é Estrutura de Repetição, suas variáveis e repetições condicionais estudadas até aqui? Em caso de uma resposta negativa: não se preocupe, ainda temos outros conteúdos que nos ajudarão a clarear as ideias... Seja persistente! Agora, se sua resposta for afirmativa: Parabéns! Contudo, para ambas as respostas é válido lembrar que há inúmeros outros conhecimentos para construir sobre o tema... Para tanto, sugerimos que consulte as obras, periódicos e sites indicados ao final desta Aula. Vejamos, agora, outra!

1.4 - Repetição Condicional - repita... até que

A estrutura "Repita" executa um bloco de comandos até que uma condição seja verdadeira. Isso significa dizer que ela executa os comandos enquanto a condição for falsa. Quando essa condição passar a ser verdadeira, a repetição se encerrará.

Vale salientar que os comandos dentro do bloco dessa estrutura serão executados pelo menos uma vez. Quando a condição é encontrada, ela será testada. Se for verdadeira passa o controle para o comando imediatamente abaixo da instrução "Até que". Se a condição for falsa, os comandos do bloco são novamente executados, até que se tenha uma condição verdadeira.

Sintaxe:

```
repita
    <comando1>
    <comando2>
    .....
    <comandon>
ate <condição>
```

Exemplo 22: Faça um Algoritmo que escreva os números inteiros positivos menores que 100.

```
algoritmo "Repita ate"
var
    NUMERO: inteiro
```

```
inicio
  NUMERO <-  0 // inicia o contador em 0
  repita
        escreval (NUMERO) // Escreval – irá pular
        //uma linha na execução do Algoritmo
        NUMERO <-  NUMERO + 1 // variável
        //acumuladora
 ate NUMERO >= 100
fimalgoritmo
```

Notem que a variável número foi utilizada também como contadora, ela irá controlar o número de vezes que o trecho do programa deve ser executado. Nesse exemplo ela foi iniciada com zero. Diferente do enquanto, quando chegarmos à linha de execução do repita automaticamente, ele entrará no loop, não testando a condição, logo escreve "0", em seguida realiza o incremento da variável (NÚMERO=0+1).

O novo valor de número será "1", ele agora irá testar a condição e somente irá parar quando esta for verdadeira, quando o número for maior ou igual a 100.

Exemplo 23: Mostrar a soma dos números pares inteiros positivos menores que 100.

```
algoritmo " InteiroPositivo"
var
  SOMA, NUMERO: inteiro
inicio
      NUMERO <-  2
      SOMA <-  0 // Inicia a variável contador = 0
      repita
            SOMA <-  SOMA + NUMERO
            NUMERO <-  NUMERO + 2 // variável
            //acumuladora
      ate NUMERO > 100
      escreva ("A soma dos números pares inteiros é: ",
      SOMA)
fimalgoritmo
```

Exemplo 24: Segue um Algoritmo que escreve 10 vezes a frase "FLAMENGO". Veja: repetido

```
algoritmo "Repetição"
var
      I : inteiro
inicio
      I <-  1 // inicia a variável contador
      repita
          escreval ("FLAMENGO O MENGÃO")
          I <-  I + 1
      ate I >=10 // variável acumuladora
fimalgoritmo
```

Exemplo 25: Faça um Algoritmo que leia um número inteiro positivo e apresente o resultado de sua tabuada de um a dez (1 a 10). repetido

Supondo que o valor desejado seja 2, então:

2 x 1 = 2 2 x 2 = 4
2 x 3 = 6 2 x 4 = 8
2 x 5 = 10 2 x 6 = 12
2 x 7 = 14 2 x 8 = 16
2 x 9 = 18 2 x 10 = 20

```
algoritmo "Tabuada'
var
  valor, contador: inteiro
inicio
      escreva ("Entre com o Valor:")
      leia (valor)
      contador <-  1 // Inicia a variável contador = 1
  repita
      escreval (valor," x ",contador," = ", (valor * contador))
      contador <-  Contador +1 // variável acumuladora
 ate contador >= 11
fimalgoritmo
```

Você sabia que as estruturas de repetição condicionais podem ser classificadas em pré-teste e pós-teste? "A estrutura de repetição condicional com pré-teste permite que várias instruções sejam executadas repetidamente enquanto uma condição qualquer for verdadeira. A estrutura de repetição condicional com pós-teste permite que várias instruções sejam executadas repetidamente enquanto uma condição qualquer não for verdadeira" (LAWISCH, 2011).

1.5 - Repetição contados - Para... até...faça

Como você já sabe, quando uma sequência de comandos deve ser executada repetidas vezes, tem-se uma estrutura de repetição. Na repetição contados o Algoritmo apresenta previamente a quantidade de repetições. A estrutura de repetição, assim como a de decisão, envolve sempre a avaliação de uma condição.

A repetição ocorre por meio de uma variável de controle definida pelo analista que atua como uma contadora automática. Fornecemos um valor inicial no próprio comando, enquanto esse limite final não for ultrapassado, o conteúdo dessa variável é incrementado em uma unidade automaticamente, e os comandos dentro do loop são executados.

Sintaxe:

```
para <variável> de <valor inicial> ate <valor final>
passo <incremento> faca
        <comando1>
        <comando2>
        .....
        <comandon>
fimpara
```

Essa estrutura de repetição utiliza uma variável a ser incrementada de um valor inicial para um valor final. Os comandos serão executados tantas vezes quantos forem os incrementos da variável.

Esse incremento do valor da variável é definido pelo comando passo, que indica qual o valor a ser acrescentado ou subtraído da variável. Quando o incremento for de 1, não é necessário informar o passo.

O valor da variável de controle não deve ser alterado dentro do bloco de comandos da estrutura "Para".

É recomendável o uso dessa instrução sempre que se souber o valor inicial e o valor final da variável de controle.

Exemplo 26: Faça um Algoritmo que escreva os números

inteiros positivos menores que 100.

Veja a Figura 6.2:

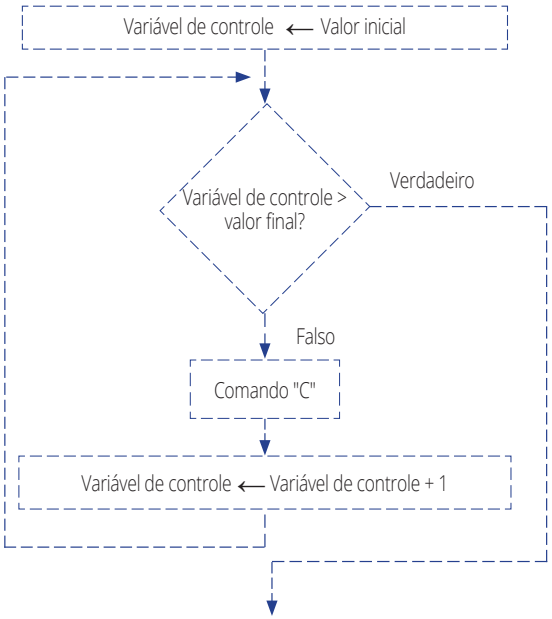


Figura 6.2 Diagrama de fluxo da estrutura PARA...ATÉ...FACA. Fonte: acervo pessoal.

Ou:

```
algoritmo "Inteiro"
var
  NUMERO: inteiro
inicio
 para NUMERO de 0 ate 99 faca // numero será
 //incrementado de zero até 99
      escreval (NUMERO)
  fimpara
fimalgoritmo
```

Como você pode verificar, nesse exemplo não houve a necessidade de iniciar a variável contadora ou até mesmo incrementar o seu valor, o próprio comando de repetição contados faz essa tarefa. A variável número irá se autoincrementar sozinha de um em um até chegar ao valor final 99.

Viu como estudar Estrutura de repetição não é tão complicado como poderia ter imaginado? Agora, para sedimentar sua aprendizagem, sugiro que pare um minuto, reflita sobre o que estudou e tente escrever resumidamente o que aprendeu até aqui. Com o resultado de sua autoavaliação, você poderá escolher entre prosseguir seus estudos ou pedir ajuda para eliminar eventuais dúvidas, antes que elas se tornem uma "bola de neve"!

Exemplo 27: Utilizaremos o mesmo exemplo anterior, escreva os números inteiros positivos menores que 100, mas agora faremos com que o contador pule de 2 em dois números, no incremento da variável.

```
algoritmo "Inteiro"
var
  NUMERO: inteiro
inicio
  para NUMERO de 0 ate 99 passo 2 faca
     escreval (NUMERO)
  fimpara
fimalgoritmo
```

Exemplo 28: Faça um Algoritmo que leia um número inteiro positivo e apresente o resultado de sua tabuada de um até dez (1 a 10).

```
algoritmo "Tabuada'
var
  valor, contador : inteiro
inicio
  escreva ("Informe o valor desejado: ")
  leia (valor)
  para contador de 1 ate 10 passo 1 faca
    escreval(valor, " x ",contador," = ", (valor * contador) )
  fimpara
fimalgoritmo
```

Exemplo 29: Faça um algoritmo para apresentar os resultados da potência de 3, variando o expoente de 0 até 10. Deve ser considerado que qualquer número elevado a zero é 1, e elevado a 1 é ele próprio. Deve ser apresentado observando, a seguinte definição:

$3^0 = 1$
$3^1 = 3$
$3^2 = 9$
(...)
$3^{10} = 59.049$

```
algoritmo "Potencia de 3"
var
      x, y, i : inteiro
inicio
      x <-  3
      y <-  1
      para i de 0 até 10 passo 1 faca
          se i = 0 entao
             y <-  1
          senao
             y <-  y * x
          fimse
          escreval (x, " ^ " , i, " = " , y)
      fimpara
fimalgoritmo
```

Exemplo 30: elaborar um Algoritmo que efetue o cálculo do fatorial de um número. Sendo o fatorial de um número calculado da seguinte forma: 5! = 5*4*3*2*1 = 120.

```
algoritmo "Fatorial"
var
      Cont, Fat, Fatorial : inteiro
inicio
      escreva ("Informe o número desejado")
      leia (Fat)
      Fatorial <-  Fat
      Cont <-  1
      enquanto (Cont < Fatorial) faca
            Fat <-  Fat * Cont
            Cont <-  Cont + 1
```

```
    fimenquanto
    escreva ("O valor do fatorial é: ", Fat)
fimalgoritmo
```

Retomando a aula

Esta Aula foi muito proveitosa e nos possibilitou a construção de importantes conhecimentos. Então, para não nos esquecermos deles, vamos recordá-los:

1 - Estruturas de repetição

Na única Seção desta Aula, você pôde identificar a necessidade de utilizar uma estrutura de repetição para a resolução de um determinado problema que necessite de estruturas de repetição dentro de outras estruturas de repetição.

Além disso, tivemos a oportunidade de diferenciar as estruturas de repetição existentes e de reconhecer a aplicabilidade de cada uma delas na resolução de problemas, aprender a utilizar a estrutura de repetição PARA FAÇA cujo controle é realizado por um contador autoincrementável, bem como as estruturas de repetição ENQUANTO FAÇA e REPITA ATÉ cujo controle é realizado pelo usuário (FERRARI, CHEMINEL, 2011).

No decorrer da Aula entendemos que a diferença básica das estruturas de repetição são que ENQUANTO-FAÇA primeiro testa a condição para depois realizar o bloco de comando, ao contrário de FAÇA-ENQUANTO que primeiro executa o bloco para depois realizar o teste e da estrutura PARA-FAÇA que tenha embutida um mecanismo de controle para determinar quando o laço deverá ser terminado (IMASTERS, 2011).

Após o estudo dos conteúdos, concluímos que as estruturas de repetição proveem uma maneira de repetir um conjunto de procedimentos até que determinado objetivo seja atingido, quando a repetição se encerra. Todas as estruturas de repetição têm em comum o fato de haver uma condição de controle, expressa por meio de uma expressão lógica, que é testada em cada ciclo para determinar se a repetição prossegue ou não (FERRARI, CHEMINEL, 2011).

Vale a pena

Vale a pena ler

LEITE, Mario. *Técnicas de programação:* uma abordagem. Rio de Janeiro: Brasport, 2006.

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de. *Algoritmos* - lógica para desenvolvimento de programação. 2. ed. São Paulo: Érica, 2007.

VELOSO, Paulo A. S. *Estrutura e verificação de programas com tipos de dados.* São Paulo: Blucher, 1987.

Vale a pena acessar

CALLE, J. L. D. C. *Introdução à linguagem C.* Atualização de André Leon Gradvohl. Disponível em: <http://www.cenapad.unicamp.br/servicos/treinamentos/apostilas/apostila_C.pdf>. Acessado em: 28 jun. 2011.

FERRARI, F.; CHEMINEL, C. *Introdução a algoritmos e programação.* Disponível em: <http://www.scribd.com/doc/51487241/52/Estruturas-de-Repeticao>. Acessado em: 28 jun. 2011.

FUNDAÇÃO ESCOLA TÉCNICA LIBERATO SALZANO VIERA DA CUNHA. *Estruturas de repetição.* Disponível em: <http://pt.scribd.com/doc/52097901/51/Repeticao-condicional-com-pre-teste>. Acesso em: 28 jun. 2011.

IBM. *IMasters.* Disponível em: <https://www.ibm.com/developerworks/mydeveloperworks/blogs/fd26864d-cb41-49cf-b719-d89c6b072893/?maxresults=100&sortby=0&lang=ru>. Acesso em: 28 jun. 2011.

LAWISCH, A. *Algoritmos.* Disponível em: <http://pt.scribd.com/doc/52097901/53/Repeticao-condicional-com-pos-teste>. Acesso em: 28 jun. 2011.

SILVA, F. M. S. *Notas de aula 4* - estruturas de repetição. Disponível em: <http://www.eagletecnologia.com/ifnmg/tp-1-cs-v/010%20-%20Notas%20de%20Aula%20Pascal%204.pdf>. Acessado em: 28 jun. 2011.

Vale a pena assistir

KEY, A. L. Estruturas de repetição. Disponível em: <http://www.youtube.com/watch?v=KVvBPlQYWeI>. Acessado em: 28 jun. 2011.

Minhas anotações

7º Aula

Estruturas de dados homogêneas

Prezado(a) aluno(a), na penúltima Aula da disciplina "Algoritmos" vamos estudar as estruturas de dados homogêneas. Para iniciar, é importante sabermos que essas estruturas "permitem agrupar diversas informações dentro de uma mesma variável. Esse agrupamento ocorrerá obedecendo sempre ao mesmo tipo de dado, e é por essa razão que elas são chamadas 'homogêneas'. A utilização desse tipo de estrutura de dados recebe diversos nomes, como: variáveis indexadas, variáveis compostas, variáveis subscritas, arranjos, vetores, matrizes, tabelas em memória ou arrays" (UTPR, 2001). Os nomes mais usados e que utilizaremos para estruturas homogêneas são: matrizes (genérico – que estudaremos na Aula 8) e vetores (matriz de uma linha e várias colunas).

Ah, para um aproveitamento mais eficaz, é importante que faça uma primeira leitura rápida do material que irá estudar. Isso lhe permitirá ter uma visão geral do conteúdo da Aula. Depois, releia o texto, sublinhando as ideias principais da maneira que for mais prática para você: linearmente, realizando anotações nas margens, marcando com símbolos ou digitando em um editor de textos. Lembre-se ainda de que mais que ler, é importante que se posicione criticamente sobre os conteúdos aqui estudados. Afinal, você é o personagem principal de sua aprendizagem!

Boa aula!

Objetivos de aprendizagem

Ao término desta aula, o aluno será capaz de:

- conceituar e interpretar o termo "estruturas homogêneas";
- definir, reconhecer a importância das Estruturas Homogêneas: Vetores e aplicá-las na programação;
- desenvolver e/ou ampliar a capacidade de percepção de um problema e utilizá-la na criação de Algoritmos.

Seções de estudo

1 - Estruturas de dados homogêneas
2 - Vetores

1 - Estruturas de dados homogêneas

CONCEITO

As estruturas de dados consistem em organizações lógicas sobre o armazenamento e manipulação dos dados que serão necessários ao algoritmo e posteriormente ao programa resultante de tal representação.

Nas Aulas anteriores, trabalhamos com estruturas básicas, nas quais uma só variável é capaz de armazenar apenas um dado de cada vez. Contudo, nesta Aula iremos nos deparar com situações em que temos a necessidade de armazenar uma grande quantidade de dados ao mesmo tempo, como 100, 300, 500.

Agora, imagine ter que criar todas essas variáveis, declarar uma por uma e depois entrar com todos os seus valores na execução do algoritmo?

Parece complicado, não é mesmo?

Mas a resposta para essa questão é o ponto central de nossa aula.

Isso significa que esse tipo de dado estruturado é também conhecido como estrutura de dados homogênea, uma vez que nelas, o agrupamento de dados obedece sempre ao mesmo tipo de dado.

Observou como podemos aproveitar o conteúdo estudado até aqui? Viu como não é tão difícil quanto podia parecer? Então, vamos continuar estudando, na Seção 2, o conteúdo sobre os Vetores.

2 - Vetores

CONCEITO

Um vetor é uma estrutura composta formada por um conjunto unidimensional (vetor possui somente uma dimensão) de dados do mesmo tipo. Por essa característica afirmamos que os vetores são estruturas de dados homogêneas.

As principais características de um vetor são os inúmeros valores que ele contém. Vale salientar que todos os seus valores são do mesmo tipo de dado, uma vez que ele apresenta somente um único nome de variável e cada conjunto é acessível independentemente, de acordo com o seu índice, que seria a posição na estrutura de dados.

Vejamos o exemplo de um vetor com sete elementos:

Valores	10	05	15	20	30	40	02
Índice	1	2	3	4	5	6	7

Figura 7.1 Vetor com sete elementos.
Fonte: acervo pessoal.

Agora, vamos montar uma estrutura para armazenar a nota de 12 alunos e identificar o nome dessa estrutura como nota. Na execução do Algoritmo os valores serão informados pelo usuário por meio do comando leia. Assim:

Valores	8	5	9	3,5	4	10	5	7,5	8,5	2,5	6	7
Índice	1	2	3	4	5	6	7	8	9	10	11	12

Figura 7.2 Estrutura "nota".
Fonte: acervo pessoal.

Como você pode notar, os índices correspondem às posições que identificam os valores armazenados independentemente dos outros valores, sendo por meio deles manipulados especificamente um ou mais valores que sejam necessários.

Vamos apresentar somente a nota do oitavo aluno, como o índice da estrutura de dados. O oitavo aluno possui a sua nota armazenada na 8ª posição da estrutura, sendo sua nota igual a 7,5.

Sintaxe:

var

<identificador> : vetor [<inicial>.....<final>] de <tipo de dado>

Onde:

<identificador> é o nome atribuído à estrutura de dados (vetor)

<tipo de dado> é o tipo de dado que será armazenado na estrutura (inteiro, real)

<inicial> e <final> correspondem respectivamente aos valores numéricos inteiros que de início e fim dos índices do vetor, sendo o seu intervalo a quantidade exata de elementos do vetor, ou seu tamanho.

Observe que as palavras "vetor" e "de" são palavras reservadas que fazem sentindo para o algoritmo e nunca devem estar ausentes na instrução de declaração de um vetor.

Vamos definir um vetor para armazenar a idade de 50 alunos de uma sala de aula.

```
var
idade : vetor [0..50] de inteiro
```

Podemos imaginar a variável de memória assim:

Idade	15	35	45	25	29	.....	13	40	42	39
Índice	1	2	3	4	5	.....	47	48	49	50

Figura 7.3 Variável "memória".
Fonte: acervo pessoal.

Para saber as posições e os valores guardados nelas, usaríamos os comandos:

```
Idade[3] <-  45
Idade[48] <-  40
Idade[5] <-  29
```

Notem que não poderíamos ter a Idade[52], pois nosso vetor foi declarado até o tamanho de 50 posições. O analista tem que seguir uma lógica que impeça esse tipo de situação, em que o índice do vetor está fora da faixa em que ele foi definido.

Exemplo 31: Vamos fazer um algoritmo para entrar com valores a um vetor definido, atribuir a ele valores e somar os valores do seu conteúdo.

```
algoritmo "Primeiro vetor"
var
    A : inteiro
    idade : vetor[1..10] de real
```

```
        inicio A <- 30
        idade[2] <- A
        idade[2] <- idade[2]+ 4
        idade[8] <- A + 5
        idade[10] <- idade[8] + 5
        idade[1] <- 54
        escreva (idade[2], idade[8], idade[10], idade[1])
    fimalgoritmo
```

Notem que o acesso a um elemento do vetor pode acontecer por meio da especificação do nome do vetor seguido do índice desejado entre colchetes. Logo, idade[2] recebeu o valor de 30, idade[8] o valor de 35.

Como o intervalo declarado na criação do vetor idades ([1..10]) consiste de 10, essa estrutura de dados composta homogênea (ou simplesmente vetor) terá a capacidade de armazenar 10 valores do tipo de dado especificado após a palavra reservada de, ou seja, nesse pequeno exemplo todos os 10 valores serão reais.

Os exemplos apresentados a seguir têm o objetivo de contextualizar e facilitar sua aprendizagem. Dessa forma, sugerimos que se atente para eles e, em caso de dúvidas, acesse o ambiente virtual para saná-las... Aprender é uma construção, para a qual trabalharemos juntos até o final do curso! Participe!

Exemplo 32: Escrever um algoritmo que declare um vetor do tipo real e leia as notas de 30 alunos.

```
algoritmo "VetorNota"
var
      I : inteiro
      notas : vetor [1..30] de real
inicio
  para I de 1 ate 30 passo 1 faca
      escreval ("Informe a nota do aluno: ")
      leia (notas[I])
  fimpara
fimalgoritmo
```

Exemplo 33: Escrever um algoritmo que declare um vetor do tipo real e leia as notas de 30 alunos, e um vetor tipo caractere para ler o nome dos trinta alunos.

```
algoritmo "VetorNota2"
  var
      I : inteiro
      notas : vetor [1..30] de real
      nome : vetor [1..30] de caractere
inicio
      para I de 1 ate 30 passo 1 faca
          escreval ("Nome do aluno: ")
          leia (nome[I])
          escreval ("Nota do aluno: ")
          leia (notas[I])
      fimpara
      escreva (nome[2], nome[3])
fimalgoritmo
```

Notem que no programa VisuAlg o tipo caractere é definido como uma "cadeia de caracteres". Logo, a variável do tipo vetor retornará à quantidade de caracteres que o usuário digitar durante a execução do algoritmo.

Que o "Visualg é um programa que interpreta e executa algoritmos como um "programa" normal de computador. Baseado em uma linguagem parecida com o "Portugol" ensinado em cursos em todo o Brasil, possui recursos como simulação da "tela" do computador, visualização de variáveis, "breakpoints", ajuda on-line, impressão das fontes e outras características que auxiliam o aprendizado das técnicas de programação" (BAIXAQUI, 2011).

Exemplo 34: Faça esse exemplo no VisuAlg e analise o resultado do algoritmo.

```
algoritmo "VetorAluno"
// Declarações
var
  I : inteiro
  nome : vetor [1..10] de caractere
inicio
  I <- 1
  escreva ("Informe o nome desejado terminando com ponto final: ")
  repita
      leia(nome[I])
      I <- I + 1
  ate ((nome[I-1] = ".") ou (I=10))
  escreva(nome[2])
fimalgoritmo
```

Neste exemplo, o programa VisuAlg irá ler um vetor tipo caractere entrando 10 nomes e irá parar a execução do algoritmo quando a variável contadora chegar a 10 ou quando o usuário digitar o " . ".

"O VisuAlg implementa as três estruturas de repetição usuais nas linguagens de programação: o laço contado PARA...ATE...FACA (similar ao FOR...TO...DODO Pascal), e os laços condicionados ENQUANTO... FACA (similar ao WHILE...DO) e REPITA...ATE (similar ao REPEAT...UNTIL). [...] PARA...FACA repete uma sequência de comandos um determinado número de vezes. [...] ENQUANTO...FAÇA, repete uma sequência de comandos enquanto uma determinada condição (especificada por uma expressão lógica) for satisfeita. [...] REPITA ... ATÉ repete uma sequência de comandos até que uma determinada condição (especificada por meio de uma expressão lógica) seja satisfeita" (SCRIBD, 2011).

Exemplo 35: Uma turma do curso de Análise de Sistemas tem 30 alunos. O professor dessa turma deseja calcular e imprimir a nota de cada aluno seguida da média da turma.

Utilizaremos para a realização do exemplo proposto um comando de repetição para ler todas as notas dos alunos, e um segundo comando de repetição para escrever todas as notas que foram digitadas pelo usuário. Veja:

```
algoritmo "VetorNota"
var
  media, soma : real
  nota: vetor[1..30] de real
  I : inteiro // Contador – irá fazer a leitura do vetor
```

```
inicio
  I <-  1 // atribui valor a variável contadora
  soma <-  0
   enquanto I <= 30 faca // laço para fazer a leitura das
   //notas do vetor
       leia(nota[I]) // entre com o valor da nota
       soma     soma + nota[I] // Variável de soma das notas
       I <-  I + 1 // Contador de incremento
  fimenquanto
  media <-  soma/30 // calcula a média das notas, que
  //seria a soma / quantidade (30)
 I <- 1 // Inicia novamente o contador de incremento para
 //escrever as notas do vetor
 enquanto I <=30 faca
       escreval("Nota:",nota[I]) // escreve as notas
       //digitadas
       I<- I +1
  fimenquanto
  escreva("Media da Turma", media) // escreve a média
  //da turma
fimalgoritmo
```

Notem que a variável de controle (contador) que fez a leitura do vetor foi iniciada em I = 1, ele começará a ler a nota[1] até a última nota[30]. Em seguida nós iniciamos novamente a variável controle (contador) I = 1, para que nós possamos escrever o conteúdo do vetor, ou seja, escrever todas as trinta notas armazenadas.

Exemplo 36: Utilizar o mesmo algoritmo anterior, mas agora iremos acrescentar o nome do aluno.

```
algoritmo "VetorAlunoNota"
var
       media, soma : real
       nota: vetor[1..30] de real
       nome: vetor[1..30] de caractere
       I : inteiro // Contador - irá fazer a leitura do vetor
inicio
       I <-  1 // atribui valor a variável contadora
       soma <-  0
       enquanto I <= 30 faca // laço para fazer a leitura das
       //notas do vetor
           leia(nome[I]) // entre com o nome do aluno
           leia(nota[I]) // entre com o valor da nota
           soma <-  soma + nota[I] // Variável de soma das
           //notas
           I <-  I + 1 // Contador de incremento
       fimenquanto
       media <-  soma/30 // calcula a média das notas, que
       //seria a soma / quantidade (30)
       I <-  1 // Inicia novamente o contador de incremento
       //para escrever as notas do
       vetor enquanto I <=30 faca
       //escreve os Alunos e notas digitadas I<- I +1
           escreval ("Aluno:", nome[I], " Nota:",nota[I])
       fimenquanto
       escreva("Media da Turma", media) // escreve a média
       //da turma
fimalgoritmo
```

Definimos aqui, outro vetor para ler o nome dos alunos = nome: vetor[1..30] de caractere e colocamos esse parâmetro na hora de ler os dados junto com as notas. Então, o usuário irá entrar com o nome do Aluno em seguida a sua nota.

A variável soma continuará fazendo a somatória das notas e a variável I será utilizada para ler os trinta nomes e notas.

Depois, para escrever o conteúdo dos vetores Nome e Nota continuaremos utilizando o mesmo recurso, iniciando o valor da variável contadora "I". Em seguida, utilizando-a para escrever o conteúdo dos vetores Nome e Nota.

Se na execução do algoritmo o nome[1]="André da Silva e a nota[1]=10, quando o algoritmo chegasse à parte de escrever o resultado seria: Aluno: André da Silva Nota=10.

Entrem no programa VisuAlg e testem todas essas alterações propostas e vejam o resultado que foi colocado, tanto no teste de mesa quanto na exibição dos valores dos vetores Nome e Nota.

Exemplo 37: Elaborar um algoritmo para receber as notas de um grupo de 30 alunos e, ao final, exibir cada uma das notas acompanhadas da média, da menor e da maior nota lida.

```
algoritmo "VetorNotadois"
var
  media, soma, menor, maior : real
  nota: vetor[1..30] de real
  I : inteiro // Contador - irá fazer a leitura do vetor
inicio
  soma <- 0
  menor <- 10
  maior <- 0
  para I de 1 ate 30 passo 1 faca // laço para fazer a
     //leitura das notas do vetor
     leia(nota[I]) // entre com o valor da nota
     se nota[I] < menor entao
         menor <- nota[I]
     fimse
     se nota[I] > maior entao
         maior <- nota[I]
     fimse
     soma <- soma + nota[I] // variável de soma das notas
  fimpara
  media <- soma/30 // calcula a média das notas, que
  //seria a soma / quantidade (30)
  para I de 1 ate 30 passo 1 faca // fazer a leitura do
     //conteúdo do vetor
     escreval(" Nota:",nota[I]) // escreve as nota digitadas
  fimpara
  escreva("Media da Turma", media," Maior Nota:", maior,
  "Menor Nota:", menor)
fimalgoritmo
```

Nesse exemplo, utilizamos o comando de repetição "para", pois nesse caso não precisamos ficar incrementando o valor do contador. O valor da variável menor foi colocado igual a 10 de propósito, para o primeiro valor digitado ser comparado a um valor predefinido, nesse caso 10.

Entre no VisuAlg e teste esse algoritmo, faça várias simulações para não restar qualquer tipo de dúvida quanto à utilização dos vetores.

"A manipulação de vetores é grandemente utilizada nos comandos de repetição, especialmente o para (for). Por exemplo, para zerar (colocar o valor 0) em todas as posições de um vetor conjuntos, seria necessário escrever 4 comandos. Assim:
conjuntos[1] ⟵ 0;
conjuntos[2] ⟵ 0;
conjuntos[3] ⟵ 0;
conjuntos[4] ⟵ 0;

Contudo, a mesma operação pode ser feita com apenas um comando de repetição para (for), usando uma variável de controle para representar as posições do vetor:
para i ⟵ 1 to 6 façaconjuntos[i] ⟵ 0;" (GOMES, 2011). É bem mais simples, você concorda?

Exemplo 38: Ler 12 elementos de um vetor, colocá-los em ordem decrescente e apresentar os Elementos Ordenados algoritmo "Classificação"

```
// Tenho que declarar um vetor com tamanho 12
// tenho que declarar as variáveis que irão fazer a troca
//dos elementos do vetor e ajudar na comparação,
//neste caso teremos uma segunda variável para a
//comparação e uma variável x para utilizar na troca.
var
      elementos:vetor[1..12] de inteiro
      I, X, J: inteiro
inicio
// vamos entrar com os valores para os 12 elementos
//do vetor
      para I de 1 ate 12 passo 1 faca
            leia(elementos[I]) // aqui estou entrando
      //com os 12  elementos do vetor.
      fimpara
// agora vou utilizar um comando de repetição para
//encadeado para fazer a comparação dos vetores
//o elemento[1] dever ser comparado com o
//elemento[2], // elemento[3]... até o elemento[12].
//Em seguida o elemento[2] não precisa ser
//comparado com o elemento[1], pois já foram
//anteriormente comparados, passando assim
//a comparar somente com os próximos e assim por
//diante. Seguindo esse raciocino, basta comparar o valor
//do elemento armazenado em elemento[1]
//com o valor armazenado em elemento[2]. Se o
//primeiro for maior que o segundo então trocam-se os
//valores. A variável I fara a leitura do Vetor e a J para o
//valor subsequente.
// Quando I=(1) — J(2,3,4,5..12) — I =(2) —
//J(3,4,5...12).
// Somente quando J atingir 12 é que o looping se
//encerra, retornando o looping a variável I
// e é acrescida de 1, reiniciando o processo. O final será
//quando a variável I = 11 e J=12, e será comparado o
//penúltimo elemento com o seu elemento subsequente,
//no caso o ultimo. Observe também o algoritmo de troca.
// Nesse caso de encadeamento, será executada primeiro
//a rotina mais interna, no caso a
// rotina J, passando o processamento para a rotina mais
//externa, quando a rotina mais
// interna fechar o seu ciclo.
para I de 1 ate 11 passo 1 faca
   para J de I+1 ate 12 passo 1 faca
      se (elementos[I] < elementos[J]) entao
         X <- elementos[I]
         elementos[I] <- elementos[J]
         elementos[J] <- X
      fimse
   fimpara
fimpara
// agora iremos listar o vetor novamente, só que em
//ordem decrescente.
para I de 1 ate 12 passo 1 faca
    escreval("vetor decrescente:", elementos[I])
fimpara
fimalgoritmo
```

Retomando a aula

Até aqui tudo correu muito bem, não é mesmo? Assim, vamos resumir os conteúdos para elaborarmos uma síntese mental sobre eles:

1 - Estruturas de dados homogêneas

Como vimos na primeira Seção da Aula 7, as estruturas de dados são organizações lógicas sobre o armazenamento e manipulação dos dados que serão necessários ao algoritmo e posteriormente ao programa resultante de tal representação.

Em outras palavras, pode ser entendida como o nome dado à organização de dados e algoritmos de forma coerente e racional de modo a otimizar o seu uso. Elas podem solucionar de forma simples problemas extremamente complexos de acordo com o modo como um conjunto de dados é organizado.

Como você pôde notar, esse é um dos temas fundamentais da ciência da computação, utilizado nas diferentes áreas para as mais variadas finalidades.

2 - Vetores

Na referida Seção, vimos que um vetor é uma estrutura composta formada por um conjunto unidimensional de dados do mesmo tipo, característica pela qual afirmamos que os vetores são estruturas de dados homogêneas.

Estudamos ainda as principais características de um vetor, as quais são os inúmeros valores que ele contém. Lembrando que esses valores são do mesmo tipo de dado, uma vez que ele apresenta somente um único nome de variável e cada conjunto é acessível independentemente, de acordo com o seu índice, que seria a posição na estrutura de dados.

Já estamos no ambiente virtual esperando sua participação. Assim, se

ficou com dúvidas, tem comentários ou sugestões a fazer sobre a Aula 7, acesse as ferramentas habituais e interaja conosco. Nosso objetivo é tornar sua aprendizagem um momento de realização pessoal e profissional!

Vale a pena

Vale a pena **ler**

BECK, Leland L. *Desenvolvimento de software básico:* assemblers, linkers, loaders, compiladores, sistemas operacionais, bancos de dados e processadores de textos. Rio de Janeiro: Campus, 1994.

SHIMIZU, Tamio. *Introdução à ciência da computação.* São Paulo: Atlas, 1988.

Vale a pena **acessar**

BAIXAKI. *Programa que interpreta e executa algoritmos.* Disponível em: <http://www.baixaki.com.br/download/visualg.htm>. Acesso em: 28 jun. 2011.

GOMES, R. C. G. *Algoritmos e lógica de programação.* Disponível em: <http://pt.scribd.com/doc/50093335/36/Uso-do-comando-de-repeticao-para-for-com-vetores>. Acesso em: 28 jun. 2011.

SCRIBD. *VisualAlg* - editor e interpretador de pseudocódigos. Disponível em: <http://pt.scribd.com/doc/54436295/Apostila-VisualAlg>. Acesso em: 28 jun. 2011.

UTPR - Universidade Tecnológica Federal do Paraná. *Estruturas de dados homogêneas.* Disponível em: <http://pessoal.utfpr.edu.br/sbkaminski/arquivos/Aula_10%20_Vetores.pdf>. Acesso em: 28 jun. 2011.

Vale a pena **assistir**

BOONG, K. *Vetor* – algoritmo. Disponível em: <http://www.youtube.com/watch?v=MWR37yk2uwo>. Acesso em: 28 jun. 2011.

Minhas anotações

8º Aula

Estruturas homogêneas II

Para continuar os estudos sobre Estruturas Homogêneas, é importante que tenha compreendido amplamente a Aula anterior que introduziu o tema. Nesse contexto, sugerimos que caso ainda reste dúvidas, realize uma revisão do conteúdo e entre em contato conosco pelas ferramentas disponibilizadas no ambiente virtual. Esteja certo(a) de que faremos o possível para facilitar seu aprendizado.

Boa aula!

Objetivos de aprendizagem

Ao término desta aula, o aluno será capaz de:

- reconhecer a importância das Estruturas Homogêneas: Matriz e aplicá-la na programação;
- desenvolver e/ou ampliar a capacidade de percepção de um problema e utilizá-la na criação de Algoritmos.

Seções de estudo

1 - Matriz

1 - Matriz

Nesta Seção, você irá ampliar seus conhecimentos sobre as Estruturas Homogêneas estudando um vetor denominado, Matriz... Você pode estar indignado: mas o que é uma matriz? Como isso pode ser aplicado a um Algoritmo? Eis alguns dos desafios que vamos superar com o estudo proposto a seguir! Prepare e organize seu ambiente de estudos, avise a todos que irá estudar e boa aula!

Até a presente Aula, nós utilizamos uma única variável indexada com apenas uma dimensão, uma coluna para várias linhas. Contudo, existem situações em que a natureza dos dados nos indica que a sua forma de armazenamento possui mais de uma dimensão. Portanto, em Algoritmo uma matriz é um vetor que possui mais de uma dimensão.

Um vetor é uma estrutura de dados homogênea, isto é, agrupa valores de um mesmo tipo; o tipo do vetor é o mesmo tipo dos dados que ele armazena (BRASIL ACADÊMICO, 2011).

É válido observar que o mais comum é a matriz de duas dimensões por se relacionar diretamente com a utilização de tabelas. Dessa forma, sendo bidimensional, a variável atua como uma grade de linhas e colunas, na qual a intersecção entre uma linha e uma coluna armazena um dado.

Cabe observar que um vetor unidimensional também é conhecido como matriz de uma única linha. Uma matriz de duas dimensões está sempre fazendo menção a linhas e colunas e é representado por seu nome e seu tamanho (dimensão) entre colchetes.

No exemplo a seguir temos uma matriz de duas dimensões teste [1..5, 1..4]. Nesse caso, o nome da matriz é "teste", com tamanho de 5 linhas (1 até 5) e de 4 colunas (1 até 4) logo, "teste" é uma matriz de 5 por 4 (5 x 4).

Dentro dessa matriz é possível armazenar 20 elementos:

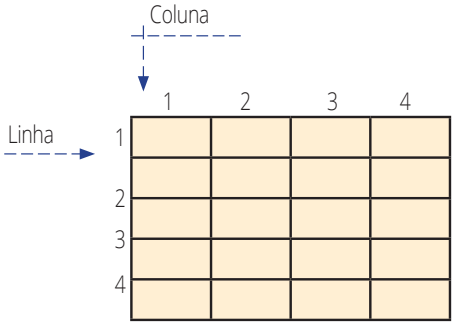


Figura 8.1 Matriz de duas dimensões.
Fonte: acervo pessoal.

Sintaxe:

var

<identificador>:vetor[<inicial>..<final>, <inicial2>..<final2> de <tipo de dado>

Onde:

<identificador> é o nome atribuído à estrutura de dados (Matriz)

<tipo de dado> é o tipo de dado que será armazenado na estrutura (inteiro, real)

<inicial> e <final> correspondem respectivamente à quantidade de linhas da Matriz, sendo uma constante inteira e positiva.

<inicial2> e <final2> correspondem respectivamente à quantidade de colunas da Matriz, sendo uma constante inteira e positiva.

As palavras "vetor" e "de" são palavras reservadas que fazem sentindo para o Algoritmo e nunca devem estar ausentes na instrução de declaração de uma matriz.

A leitura de dados de uma matriz é passo a passo um elemento de cada vez. Note que ela também é referenciada por um índice, nesse caso, indicando a linha e coluna, na qual essa intersecção é o endereço onde o elemento está armazenado.

Vamos imaginar a seguinte matriz - 5 linhas por 5 colunas (5 X 5):

Ufa! Isso parece complicado, mas na figura abaixo está tudo explicadinho. Vejam.

Coluna

Linha	1	2	3	4	5
1	10	5	10	10	10
2	5	4	20	9	9
3	7	3	100	3	3
4	6	3	70	4	4
5	7	8	30	9	9

Figura 8.2 Matriz de 5 linhas por 5 colunas.
Fonte: acervo pessoal.

Vamos ver quais são os valores retornados para as expressões:

a) teste[1,1] – nesse caso o valor armazenado é 10, ou seja, a intercessão da linha 1 com a coluna 1;

b) teste[2,3] – o valor armazenado é 20, ou seja, a intercessão da linha 2 com a coluna 3;

c) soma<- teste[4,4] + teste[5,1] – A variável soma será igual a 4 + 7 = 11;

d) para atribuir um valor, basta referenciar a linha e colunas desejadas. Teste[4,2] <- 50.

Exemplo 39: Elaborar um Algoritmo para entrar com as notas dos alunos em uma matriz de ordem 8 por 4 (oito linhas por quatro colunas). Nesse exemplo serão 32 notas.

Vamos, aqui, utilizar um comando de repetição para controlar as linhas e outro para controlar as colunas.

```
algoritmo "Exemplo Matriz"
var
  //declaração da Matriz 8 x 4
  notas: vetor [1..8,1..4] de real
  I, J : inteiro
inicio
 para I de 1 ate 8 passo 1 faca // declara as linhas da matriz
```

```
    para J de 1 ate 4 passo 1 faca // declara as colunas da
      //matriz
      leia(notas[I,J]) // entre com o valor das notas na Matriz
    fimpara
  fimpara
fimAlgoritmo
```

Observe, a seguir, como ficaria esse mesmo Algoritmo para Ler e Escrever uma Matriz.

```
algoritmo "Matriz"
var
  notas: vetor[1..8,1..4] de real
  I, J : inteiro
inicio
   para I de 1 ate 8 passo 1 faca
     para J de 1 ate 4 passo 1 faca
        leia(notas[I,J]) // entre com o valor das notas na
        //Matriz
     fimpara
   fimpara
   para I de 1 ate 8 passo 1 faca
        //Escreve o valor da Matriz
        escreval(notas[I,1],notas[I,2],notas[I,3],notas[I,4])
   fimpara
fimAlgoritmo
```

Poderíamos utilizar também o mesmo processo de leitura:

```
para I de 1 ate 8 passo 1 faca
  para J de 1 ate 4 passo 1 faca
       // Escreve o valor da Matriz fimpara
       escreval ("Notas dos Alunos em Matriz:",notas[I,J])
  fimpara
fimpara
```

Os conteúdos desta Aula são breves. Mesmo assim podem ocorrer eventuais dúvidas! Nesse caso, você terá a oportunidade de entendê-los melhor até o final desta Aula com os temas e exemplos propostos e com as dicas em destaque, bem como com a consulta das referências disponibilizadas. Continue estudando com dedicação. Estamos indo muito bem!

Exemplo 40: Elaborar um Algoritmo para receber uma matriz 3 por 3, multiplicar os elementos da diagonal principal por uma constante X, que será lida, e exibir a matriz modificada:

4	5	6
7	8	9
4	3	2

x = 2

8	5	6
7	16	9
4	3	4

Figura 8.3 Matriz 3 por 3.
Fonte: acervo pessoal.

```
algoritmo "Matriz Diagonal"
var
  num: vetor[1..3,1..3] de real
  I, J, X : inteiro
inicio
  escreva("Entre com a Constante:")
  leia(X)
  para I de 1 ate 3 passo 1 faca
    para J de 1 ate 3 passo 1 faca
       leia(num[I,J]) // entre com o valor das notas na
       //Matriz
       se I = J entao
          num[I,J] <- num[I,J] * x
       fimse
    fimpara
  fimpara
  para I de 1 ate 3 passo 1 faca
    escreval(num[I,1],num[I,2],num[I,3]) // Escreve o valor
    //da Matriz
  fimpara
fimalgoritmo
```

Retomando a aula

Fomos muito bem até o presente momento, você concorda? Então, para encerrar esse tópico, vamos recordar:

1 - Matriz

Na última Aula desta disciplina tivemos a oportunidade de entender que uma matriz é uma coleção de variáveis de um mesmo tipo que é referenciada por um nome comum.

Vimos ainda que os vetores são matrizes unidimensionais, ao passo que as matrizes bidimensionais são matrizes de matrizes unidimensionais.

Caso você tenha ficado com dúvidas sobre a Aula 8, acesse as ferramentas "fórum", "quadro de avisos" ou "chat" e interaja com seus colegas de curso e com seu professor.
Lembre-se de que os conteúdos estudados nesta disciplina representam apenas o primeiro passo de sua construção de conhecimentos sobre os Algoritmos, uma vez que aprender é algo que vamos fazer nossa vida toda! Assim, é fundamental que continue pesquisando e interagindo com seus colegas de curso sobre o tema, uma vez que sua dedicação pode ser uma das responsáveis pela excelência de sua vida acadêmica e profissional! Pense nisso...
Agora, é com você!

Vale a pena

Vale a pena ler

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de Oliveira. *Algoritmos* - lógica para desenvolvimento de programação. 2. ed. São Paulo: Érica, 2007.

VELOSO, Paulo A. S. *Estrutura e verificação de programas com tipos de dados*. São Paulo: E. Blucher, 1987.

Vale a pena **acessar**

BRASIL ACADÊMICO. *Algoritmos I*. Disponível em: <http://www.brasilacademico.com/apostilas/A8_Vetor.pdf>. Acesso em: 28 jun. 2011.

SIEBRA, S. A. Introdução à programação. Disponível em: <http://pt.scribd.com/doc/50982484/9/Estruturas-Homogeneas-Bidimensionais-Matrizes>. Acesso em: 28 jun. 2011.

UFMA. *Algoritmos 1*. Disponível em: <http://www.deinf.ufma.br/~vidal/Algoritmos1/vetoresmatrizes.pdf>. Acesso em: 28 jun. 2011.

Vale a pena **assistir**

CUISSI, E. *VisuAlg Matriz*. Disponível em: <http://www.deinf.ufma.br/~vidal/Algoritmos1/vetoresmatrizes.pdf>. Acesso em: 28 jun. 2011.

Referências

AVILLANO, Israel de Campos. *Algoritmo de Pascal:* Manual de Apoio. 2 ed. Rio de Janeiro: Editora Ciência Moderna ltda, 2006.

BECK, Leland L. *Desenvolvimento de software básico:* assemblers, linkers, loaders, compiladores, sistemas operacionais, bancos de dados e processadores de textos. Rio de Janeiro: Campus, 1994.

GUIMARÃES, Ângelo de Moura; LAGES, Newton Alberto de Castilho. *Algoritmos e estrutura de dados*. Rio de Janeiro: LTC, 1994.

MANZANO, José Augusto N. G.; OLIVEIRA, Jayr Figueiredo de Oliveira. *Algoritmos* - lógica para desenvolvimento de programação. 2. ed. São Paulo: Érica, 2007.

VELOSO, Paulo A. S. *Estrutura e verificação de programas com tipos de dados*. São Paulo: E. Blucher, 1987.

VENÂNCIO, Cláudio Ferreira. *Desenvolvimento de algoritmos:* uma nova abordagem. São Paulo: Érica, 2000.

SEBESTA, Robert W. *Concepts of programming languages*. 4th ed. USA: Addison-Wesley, 1999.

SHIMIZU, Tamio. *Introdução à ciência da computação*. São Paulo: Atlas, 1988.

Minhas anotações