ELABORATA
INFORMÁTICA

ALGORITMOS

Curitiba / 2018

Srs. Alunos

A Elaborata Informática, com o objetivo de continuar prestando-lhe um excelente nível de atendimento e funcionalidade, informa a seguir algumas regras que devem ser observadas quando do uso do laboratório e seus equipamentos, visando mantê-los sempre em um perfeito estado de funcionamento para um melhor aproveitamento de suas aulas.

É proibido:

- Atender celular. Por favor, retire-se da sala, voltando assim que desligar.
- Fazer cópias ilegais de software (piratear), com quaisquer objetivos.
- Retirar da sala de treinamento quaisquer materiais, mesmo que a título de empréstimo.
- Divulgar ou informar produtos ou serviços de outras empresas sem autorização por escrito da direção Elaborata.
- Trazer para a sala de treinamento, qualquer tipo de equipamento pessoal de informática, como por exemplo:
  - Computadores de uso pessoal
  - Notebooks
  - Placas de vídeo
  - Placas de modem
  - Demais periféricos
  - Peças avulsas como memória RAM, ferramentas, etc.
- O consumo de alimentos ou bebidas
- Fumar

Atenciosamente

Elaborata Informática

Sumário


ELABORATA
INFORMÁTICA
CAPÍTULO 1
INTRODUÇÃO À PROGRAMAÇÃO DE COMPUTADORES

ALGORITMOS

1.1 Lógica de Programação

A lógica de programação é necessária para pessoas que desejam trabalhar com desenvolvimento de sistemas e programas, ela permite definir a sequência lógica para o desenvolvimento. Então o que é lógica de programação? Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo. Estes pensamentos, podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa. Podemos definir então como sequência lógica os passos executados até atingir um objetivo ou solução de um problema.

Na linguagem comum, entende-se por instruções “um conjunto de regras ou normas definidas para a realização ou emprego de algo”. Em informática, porém, instrução é a informação que indica a um computador uma ação elementar a executar. Convém ressaltar que uma ordem isolada não permite realizar o processo completo, para isso é necessário um conjunto de instruções colocadas em ordem sequencial lógica. Por exemplo, se quisermos fazer uma omelete de batatas, precisaremos colocar em prática uma série de instruções: descascar as batatas, bater os ovos, fritar as batatas, etc... É evidente que essas instruções têm que ser executadas em uma ordem adequada – não se pode descascar as batatas depois de fritá-las. Dessa maneira, uma instrução tomada em separado não tem muito sentido; para obtermos o resultado, precisamos colocar em prática o conjunto de todas as instruções, na ordem correta.

1.2 Algoritmo

Um algoritmo é formalmente uma sequência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas. Como exemplos de algoritmos podemos citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos seriam os manuais de aparelhos eletrônicos, como um videocassete, que explicam passo a passo como, por exemplo, gravar um evento. Até mesmo as coisas mais simples, podem ser descritas por sequências lógicas. Por exemplo, usar o orelhão:

- Retirar o fone do gancho;

- Inserir o cartão;
- Ao ouvir o sinal de discar, discar o número desejado;
- Após a conversa, colocar o fone no gancho;
- Retirar o cartão.

1.3 Linguagens de Programação

Um algoritmo escrito em linguagem natural passa a ser chamado de programa depois de convertido para uma linguagem aceita por um computador real, uma linguagem de programação. Todos os trabalhos executados pelo computador são feitos seguindo programas. Um computador sem programas é comparável a uma televisão sem novelas, filmes, jornais, etc. um móvel sem serventia.

Basicamente, um computador só entende a linguagem binária, ou também chamada linguagem de máquina (linguagem que só admite dois símbolos: zero e um). Foi assim que os primeiros programadores fizeram seus programas nos computadores da década de 50. Seus programas eram filas intermináveis de zeros e uns. Para não trabalhar com linguagem binária construíram-se os códigos operacionais, mais fáceis de serem manipulados pelo homem, e deu-se ao próprio computador o trabalho de traduzi-los para a linguagem binária, única que ele entende. Assim um único código, com poucos símbolos, substitui uma fila de zeros e uns.

Para facilitar mais a programação, foi criada a linguagem assembly (linguagem de montagem), que a grosso modo, é a utilização de abreviações de palavras inglesas que indicam a função que se deseja do computador. Exemplos destes códigos: MPY, JMP, ADD, que são abreviações das palavras inglesas multiply, jump, add, significando, respectivamente, multiplicar, saltar, somar. Passar de abreviações à utilização de palavras humanas completas foi só uma questão de tempo. Hoje só programas especiais são escritos em linguagem assembly.

São chamadas linguagens de alto nível as linguagens que utilizam palavras completas e estruturas de expressão semelhantes ao linguajar humano, além de outros recursos importantes. Deve-se entender que “alto nível” não significa maior complexidade e sim o oposto, pois são linguagens mais distantes do linguajar binário, e mais próximas do linguajar humano. Por questões históricas relacionadas ao desenvolvimento dos computadores, estas linguagens geralmente utilizam termos

ingleses, como por exemplo, "if", "else", "while", "for", etc. Este fato se constitui em uma vantagem, pois padroniza os termos. Dentre as principais linguagens de alto nível podemos citar (por ordem de criação):

- Fortran;
- Cobol;
- Basic;
- Pascal;
- C;
- C++;
- Java.

1.4 Exercícios

1. Escreva um algoritmo que descreva o seguinte processo: trocar uma lâmpada.


ELABORATA
INFORMÁTICA
CAPÍTULO 2
ALGORITMOS COMPUTACIONAIS

ALGORITMOS

2.1 Fases do Algoritmo

Ao montar um algoritmo que será transformado mais tarde em um programa de computador, precisamos primeiro dividir o problema apresentado em três fases fundamentais:

1. **ENTRADA**: São os dados de entrada do algoritmo, necessários para o processamento;
2. **PROCESSAMENTO**: São os procedimentos utilizados para chegar ao resultado final;
3. **SAÍDA**: São os dados resultantes do processamento.

Imagine o seguinte problema: Calcular a média final dos alunos da 3ª Série. Os alunos realizarão quatro provas: P1, P2, P3 e P4. Onde:

Média Final = (P1 + P2 + P3 + P4) / 4

Para montar o algoritmo proposto, faremos três perguntas:

a) Quais são os dados de entrada?

R: Os dados de entrada são P1, P2, P3 e P4.

b) Qual será o processamento a ser realizado?

R: O procedimento será somar todos os dados de entrada e dividi-los por 4 (quatro).

c) Quais serão os dados de saída?

R: O dado de saída será a média final.

ALGORITMO:

Leia a nota da prova 1

Leia a nota da prova 2

Leia a nota da prova 3

Leia a nota da prova 4

Some todas as notas e divida o resultado por 4

Mostre o resultado da divisão

Após desenvolver um algoritmo, ele deverá ser testado. Este teste é chamado de TESTE DE MESA, que consiste em seguir as instruções do algoritmo de maneira precisa para verificar se o procedimento utilizado está correto ou não.

2.2 Pseudo-código

Os algoritmos computacionais são descritos em uma linguagem chamada pseudo-código (às vezes também chamada de pseudo-linguagem). Este nome é uma alusão à posterior implementação do algoritmo em uma linguagem de programação, ou seja, quando formos programar em uma linguagem, por exemplo C, geraremos código em C. Por isso os algoritmos são independentes das linguagens de programação. Ao contrário de uma linguagem de programação, não existe um formalismo rígido de como deve ser escrito o algoritmo. O algoritmo deve ser fácil de se interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação. Neste curso, utilizaremos um tipo específico de pseudo-código em português, semelhante ao "Portugol", para desenvolver os algoritmos.

2.3 Exercícios

Um vendedor ganha 8% para cada produto vendido na loja. Crie um algoritmo que calcule o ganho, num dia, com as seguintes vendas:

a) Um televisor (R$ 990,00)

b) Um refrigerador (R$ 1.400,00)

c) Um microondas (R$ 230,00)

d) Imprima na tela o lucro referente a cada venda e depois, o total do dia.

	***Obs.:*** *Antes de fazer o algoritmo em si, faça o detalhamento do que precisa ser feito em cada uma das 3 etapas principais do algoritmo. Ao final, faça o teste de mesa para confirmar se o algoritmo está correto..*

CAPÍTULO 3

ITENS FUNDAMENTAIS

ALGORITMOS

3.1 Comentários

Quando se torna interessante ou mesmo necessário inserir um comentário explicativo em um algoritmo, este deve ser precedido por "//".

Exemplo:

```
algoritmo "media"
// Função: Calcular a média de 2 números.
// Programador: Rafael David
// Data: 04/03/2004
// Versão: 1

    // Seção de Declarações
    var
        x : real
        y : real
        media : real

    // Seção de Comandos
    inicio
        x <- 10
        y <- 20
        media <- ( x + y ) / 2
        escreva ( "A média é igual a ", media ) // Mostra a média na tela

fimalgoritmo
```

Obs.:Os comandos de comentário não têm nenhuma função de processamento, tendo como finalidades tornar mais fácil a leitura dos algoritmos e auxiliar sua documentação.

3.2 Constantes, Variáveis e Tipos de Dados

Variáveis e constantes são os elementos básicos que um programa manipula. Uma variável é um espaço reservado na memória do computador para armazenar um tipo de dado determinado. Variáveis devem receber nomes para poderem ser referenciadas e modificadas quando necessário. Um programa deve conter declarações que especificam de que tipo são as variáveis que ele utilizará e as vezes um valor inicial. Tipos podem ser por exemplo: inteiro, carácter, etc. As expressões combinam variáveis e constantes para calcular novos valores.

3.2.1 Constantes

Constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa. Uma constante pode ser do tipo de dados inteiro, real, lógico ou caracter (alfanumérico).

Exemplos:

```
    media <- ( x + y ) / 2 // o número 2 é uma constante do tipo inteiro
    pi <- 3.14 // o número 3.14 é uma constante do tipo real
    nome <- "Paulo" // "Paulo" é uma constante do tipo caracter
```

Uma constante do tipo lógico pode assumir os valores VERDADEIRO ou FALSO.

3.2.2 Variáveis

Variável é a representação simbólica de uma posição de memória, cujo conteúdo pode se alterado ao longo do tempo durante a execução de um programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante. Uma variável possui um nome e um tipo de dados. O nome de uma variável é formado por letras, dígitos e alguns caracteres especiais permitidos, sendo o primeiro símbolo sempre uma letra. O tipo de dados de uma variável, assim como as constantes, pode ser inteiro, real, lógico ou carácter (alfanumérico).

Exemplo:

```
media <- ( x + y ) / 2 // media, x e y são variáveis que
                       // armazenam valores numéricos

idade <- 50 // este comando atribui o valor 50 à variável idade
```

3.2.3 Declaração de Variáveis

Para poder utilizar uma variável, deve-se declará-la utilizando a seguinte sintaxe:

```
<nome> : <tipo>
```

Exemplos:

```
    nome : caracter
    media : real
```

3.3 Operadores

Os operadores são meios pelo qual incrementamos, decrementamos, comparamos e avaliamos dados dentro do computador. Temos três tipos de operadores:

- Operadores Aritméticos
- Operadores Relacionais
- Operadores Lógicos

3.3.1 Operadores Aritméticos

Os operadores aritméticos são utilizados para obter resultados numéricos. Os símbolos para os operadores aritméticos são:

Operação	Símbolo
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Módulo (resto da divisão inteira)	mod

3.3.2 Operadores Relacionais

Os operadores relacionais são utilizados para comparar caracteres e números. Os valores a serem comparados podem ser constantes ou variáveis. Estes operadores sempre retornam valores lógicos (VERDADEIRO ou FALSO). Os operadores relacionais são:

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

Exemplo:

Tendo duas variáveis A e B cujos valores são 5 e 3, respectivamente, os resultados das expressões abaixo seriam:

Expressão	Resultado
A = B	FALSO
A <> B	VERDADEIRO
A > B	VERDADEIRO
A < B	FALSO
A >= B	VERDADEIRO
A <= B	FALSO

3.3.3 Operadores Lógicos

Os operadores lógicos servem para combinar resultados de expressões, retornando se o resultado final é verdadeiro ou falso. Os operadores lógicos são 'e', 'ou' e 'não'. Uma operação 'e' resulta em verdadeiro somente se os dois operandos forem verdadeiros. Uma operação 'ou' resulta em verdadeiro se, pelo menos, um operando for verdadeiro. Uma operação 'não' inverte o valor do operando, ou seja, caso o operando seja verdadeiro o resultado é falso, e vice-versa.

Exemplo:

Tendo três variáveis A, B e C, cujos valores são 5, 8 e 1, respectivamente, os resultados das expressões abaixo seriam:

Expressão	Resultado
(A = B) e (B > C)	FALSO
(A <> B) ou (B < C)	VERDADEIRO
nao (A > B)	VERDADEIRO
(A < B) e (B > C)	VERDADEIRO
(A >= B) ou (B = C)	FALSO
nao (A <= B)	FALSO

3.3.4 Operador de Atribuição

O operador de atribuição '<-' é utilizado para armazenar o resultado de uma expressão em uma variável. Este valor resultante da expressão deve ser do mesmo tipo que a variável que irá armazená-lo. A operação de atribuição tem a seguinte sintaxe:

```
<variável> <- <expressão>
```

Exemplos:

```
nome <- "Paulo"
idade <- 50
condicao <- FALSO
x <- aux
media <- ( x + y ) / 2
contador <- contador + 1
```

Toda variável usada à direita do operador de atribuição não sofre alteração e deve possuir um valor antes da expressão ser avaliada.

3.4 Entrada e Saída de Dados

3.4.1 Comando de Entrada

O comando de entrada 'leia' é utilizado para ler dados de um dispositivo externo, como por exemplo o teclado, e armazená-los em variáveis. A sintaxe deste comando é a seguinte:

```
leia ( <variável> )
```

Exemplo:

```
leia ( nome )
```

3.4.2 Comando de Saída

O comando de saída 'escreva' é utilizado para exibir dados em um dispositivo externo, como por exemplo o monitor de vídeo. A sintaxe deste comando é a seguinte:

```
escreva ( <lista de variáveis e constantes separadas por vírgula> )
```

Exemplos:

```
escreva ( "Oi!" )
escreva ( idade )
escreva ( "A média dos valores é igual a ", media )
```

Se for preciso inserir uma quebra de linha após os dados exibidos, pode-se utilizar o comando 'escreval' em lugar do comando 'escreva'.

3.5 Comandos Delimitadores do Algoritmo

Para indicar o início do algoritmo, utiliza-se o comando 'algoritmo' seguido pelo nome do mesmo colocado entre aspas duplas. O fim do algoritmo é indicado pelo comando 'fimalgoritmo'. Além disso, existe ainda a seção de declaração de variáveis (opcional), que é iniciada pelo comando 'var' e estende-se até o comando 'inicio', que por sua vez marca o começo da seção de comandos do algoritmo.

Exemplo:

```
algoritmo "algoritmo_amigo"
    var
        nome : caracter
    inicio
        leia ( nome )
        escreva ( "Olá ", nome, "! Como você está se sentindo hoje?" )
fimalgoritmo
```

3.6 Exercícios

1. Para testar o aprendizado da sintaxe vista até agora, faça um algoritmo que leia um valor para o salário e um valor (em porcentagem) para os impostos. Como saída, o algoritmo deve exibir na tela o salário líquido e o valor dos impostos. Utilize comentários para separar as etapas de entrada, processamento e saída.
2. Escreva um algoritmo que leia dois números inteiros. O algoritmo deve imprimir na tela:

   a) A soma entre os números

   b)A divisão do primeiro pelo segundo

   c)A multiplicação do segundo pelo primeiro


ELABORATA
INFORMÁTICA
CAPÍTULO 4
ESTRUTURAS
DE CONTROLE

4.1 Comandos de Decisão (condicionais)

Os comandos de decisão ou desvio fazem parte das técnicas de programação que conduzem a estruturas de programas que não são totalmente seqüenciais. Com as instruções de salto ou desvio pode-se fazer com que o programa proceda de uma ou outra maneira, de acordo com as decisões lógicas tomadas em função dos dados ou resultados anteriores. As estruturas de decisão fundamentais são:

- se / então
- se / então / senão
- escolha/caso

4.1.1 Se / Então

A estrutura de decisão 'se / então' normalmente vem acompanhada de um comando ou bloco de comandos e funciona da seguinte forma: se determinada condição for satisfeita (ou seja, se for verdadeira) então execute determinado comando ou bloco de comandos.

Exemplo:

O algoritmo abaixo verifica se a média de um aluno é maior ou igual a 7,0. Em caso afirmativo, o algoritmo exibe a mensagem "aprovado" na tela.

```
se media >= 7.0 entao
    escreva ( "aprovado" )
fimse
```

A quebra de linha após o comando 'entao' e o recuo da linha seguinte são propositais. Isto acontece porque pode-se ter mais de um comando que será executado somente se a condição do 'se' for verdadeira. A este agrupamento de comandos que serão executados dá-se o nome de bloco de comandos e ao recuo destes comandos dá-se o nome de indentação.

Exemplo:

```
algoritmo "ordena_numeros"

    var

        x : inteiro
        y : inteiro
```

```
            aux : inteiro // variável auxiliar necessária para realizar a
ordenação

    inicio

        // lê dois números sem necessariamente estarem ordenados
        leia ( x )
        leia ( y )

        // ordena os números
        se x > y entao
            aux <- x
            x <- y
            y <- aux
        fimse

        // exibe os números ordenados
        escreva ( x )
        escreva ( y )

fimalgoritmo
```

4.1.2 Se / Então / Senão

A estrutura de decisão 'se / então / senão', funciona exatamente como a estrutura 'se', com apenas uma diferença: com o comando 'se' somente podemos executar comandos caso a condição seja verdadeira, diferente de 'se / senão', pois sempre um comando (ou bloco de comandos) será executado independente da condição, ou seja, caso a condição seja verdadeira os comandos da condição verdadeira (após o 'entao') serão executados, caso contrário os comandos da condição falsa (após o 'senao') serão executados.

Exemplo:

```
se media >= 7.0 entao
    escreva ( "aprovado" )
senao
    escreva ( "reprovado" )
fimse
```

No exemplo acima está sendo testada uma condição que, se for verdadeira, exibe "aprovado" na tela, caso contrário exibe "reprovado". Podemos também dentro de uma mesma condição testar outras condições, como no exemplo abaixo:

```
se media >= 5.0 entao
    se media >= 7.0 entao
```

```
        escreva ( "aprovado" )
    senao
        escreva ( "em recuperação" )
    fimse
senao
    escreva ( "reprovado" )
fimse
```

4.1.3 Escolha / Caso

A estrutura de decisão 'escolha / caso' é utilizada para testar uma única expressão, que produz um resultado. Compara-se, então, o resultado obtido no teste com os valores fornecidos em cada cláusula 'caso'. Se o valor é encontrado, os comandos referentes à cláusula específica são executados. A cláusula 'outrocaso' é opcional, e os seus comandos são executados quando não é encontrado um valor igual ao resultado obtido. No exemplo abaixo, é testado o conteúdo da variável 'op'. Caso uma das condições seja satisfeita, é atribuído para a variável 'titulo' uma string específica, caso contrário é atribuído a string "Opção errada".

```
escolha op
    caso 1
        titulo <- "Opção 1"
    caso 2
        titulo <- "Opção 2"
    caso 3
        titulo <- "Opção 3"
    caso 4
        titulo <- "Opção 4"
    caso 5
        titulo <- "Opção 5"
    outrocaso
        titulo <- "Opção errada"
fimescolha
```

Segue outro exemplo:

```
escolha nota
    caso 0, 1 ate 3
        escreva ( "reprovado" )
    caso 4 ate 6
        escreva ( "em recuperação" )
    caso 7, 8, 9, 10
        escreva ( "aprovado" )
fimescolha
```

4.1.4 Exercícios

1. Escreva um algoritmo que leia dois números e mostre na tela o maior deles.
2. Escreva um algoritmo que leia três números e mostre na tela o menor deles.
3. Escreva um algoritmo que leia um carácter, identificando se é uma vogal
4. Escreva um algoritmo que leia um ano e calcule se este é bissexto, imprimindo esta informação na tela. Obs.: Um ano é bissexto quando dividido por 4 possuir resto 0.
5. Escreva um algoritmo que leia uma data de nascimento e imprima na tela se o usuário é maior ou menor de idade.
6. Escreva um algoritmo que leia um número e, caso ele seja par, mostre na tela a mensagem "número par".
7. Escreva um algoritmo que leia um número de 1 a 7 e imprima na tela o dia da semana, conforme a sequência: 1=domingo, 2=segunda, 3=terça, etc. Utilize escolha/caso.
8. Escreva um algoritmo que leia as 4 notas de um aluno e calcule a média final. Se o aluno possuir média inferior a 50, imprimir “aluno reprovado”. Se a média for maior 70, imprimir “aluno aprovado”. Outros valores, imprimir “aluno em recuperação”.

4.2 Comandos de Iteração (ou Repetição)

Utilizamos os comandos de iteração quando desejamos que um comando ou um determinado conjunto de comandos sejam executados repetidamente. As estruturas de repetição fundamentais são:

- enquanto / faça
- repita / até
- para / até / faça

4.2.1 Enquanto / Faça

Neste caso, o bloco de operações será executado enquanto a condição for verdadeira. O teste da condição será sempre realizado antes de qualquer operação. Enquanto a condição for verdadeira, o processo se repete. Pode-se utilizar esta estrutura para trabalhar com contadores, como no exemplo abaixo:

```
cont <- 1 // inicializa contador
enquanto cont <= 10 faca
    escreval ( cont )
    cont <- cont + 1 // incrementa contador
fimenquanto
```

Segue outro exemplo:

```
cont <- 0
soma <- 0 // inicializa acumulador
leia ( num )
enquanto cont <= num faca
    soma <- soma + cont // adiciona o valor atual de cont ao acumulador
    cont <- cont + 1
fimenquanto
escreva ( soma )
```

4.2.2 Repita / Até

Neste caso, executa-se primeiro o bloco de instruções e somente depois é realizado o teste da condição. Se a condição for verdadeira, o fluxo do programa continua normalmente. Caso contrário, os comandos (ou o comando) do laço são processados novamente.

Exemplo:

```
repita
    escreva ( "Você deseja continuar? (s/n) " )
    leia ( opcao )
ate opcao = "s" ou opcao = "n"
```

Segue outro exemplo:

```
maior_ate_agora <- 0
repita
    leia ( num )
    se num > maior_ate_agora entao
        maior_ate_agora <- num
    fimse
ate num = 0
escreva ( "O maior número digitado foi ", maior_ate_agora )
```

4.2.3 Para / Até / Faça

Esta estrutura é ideal quando sabemos o número de vezes que bloco de comandos deve ser executado. O laço só é finalizado quando o número de repetições é atingido. Nesta estrutura, precisa-se dizer qual é a variável que vai controlar a execução

do laço e qual o intervalo do laço. Cada vez que o bloco é executado, o valor da variável de controle é incrementado.

Exemplo:

```
algoritmo "tabuada"
    var
        num : inteiro
        x : inteiro
        result : inteiro
    inicio
        leia ( num )
        para x de 1 ate 10 faca
            result <- x * num
            escreval ( result )
        fimpara
fimalgoritmo
```

Pode-se utilizar o comando 'passo', seguido de um valor inteiro, para alterar a variação que a variável de controle sofre a cada iteração. A seguir, um exemplo de utilização deste comando:

```
leia ( num )
para cont de num ate 1 passo -1 faca
    escreval ( "Faltam ", cont, " iterações para acabar" )
fimpara
escreva ( "Acabou!" )
```

4.2.4 Exercícios

1. Escreva um algoritmo que leia vários números e, ao final, mostre a média deles na tela. A entrada de dados deve parar quando for digitado o número 0.
2. Escreva um algoritmo que leia vários números e, ao final, mostre na tela a quantidade de números lidos pares e ímpares. A entrada de dados deve parar quando for digitado o número 0.
3. Escreva um algoritmo que calcule e imprima na tela a somatória de 10 números pares apenas.
4. Escreva um algoritmo que leia um número e mostre na tela seu fatorial. (Ex.: fatorial de 3 é igual 3x2x1)
5. Escreva um algoritmo que leia um número e mostre na tela se ele é primo ou não. (Número primo é divisível apenas por 1 e por ele próprio)
6. Escreva um algoritmo que leia as 4 notas de 10 alunos, calcule e imprima:

a) A média de cada aluno

b) A quantidade de alunos que passaram (média 70)

c) A quantidade de alunos reprovados (abixo de 50)

7. Escreva um algoritmo que leia dois números e imprima na tela a multiplicação do primeiro pelo segundo número. Ao final, o algoritmo deve perguntar se desejar calcular outros números.
8. Escreva um algoritmo que calcule o salário líquido de 5 funcionários de uma empresa, imprimindo na tela o maior e o menor salário. Ao final, perguntar se deseja calcular novos salários. Obs.: O valor do imposto a descontar é de 12%.
9. Escreva um algoritmo que leia a altura e o peso de 20 atletas. Imprima na tela a maior e menor altura, o maior e menor peso e o total de atletas com altura superior a 1,85m.
10. Escreva um algoritmo de um menu para um sistema de locadora de DVDs. O usuário terá 5 opções (selecionadas conforme o número digitado): 1-Cadastro cliente, 2-Cadastro filmes, 3-Locação, 4-Devolução, 5-Pesquisa. Quando o usuário digitar uma das opções, será impresso na tela a frase referente ao menu (ex.: Se escolher locação: “Tela locação”). Se o usuário digitar 0, o sistema será encerrado.
11. Escreva um algoritmo, utilizando enquanto, que leia um número e imprima este valor elevado ao quadrado. No final, perguntar se deseja calcular outro número.


ELABORATA
INFORMÁTICA
CAPÍTULO 5
VETORES E MATRIZES

Foi visto, no início deste curso, ser possível dar um nome para uma posição de memória, sendo que a esta será associado um valor qualquer. Ocorre que, muitas vezes, a utilização de variáveis simples não é suficiente para resolver certos problemas computacionais, como por exemplo, construir um algoritmo para ler o nome de várias pessoas e imprimir um relatório destes mesmos nomes, mas ordenados alfabeticamente.

Para resolver problemas como este, e outros, foi criado uma nova maneira de definir variáveis, a qual foi denominada de variável composta ou indexada. Uma variável indexada corresponde a uma sequência de posições de memória, à qual daremos um único nome, sendo que cada uma destas posições pode ser acessada através de um índice. O índice corresponde a um valor numérico inteiro. Cada uma das posições de memória de uma variável indexada pode receber valores no decorrer do algoritmo como se fosse uma variável comum, a única diferença reside na sintaxe de utilização desta variável.

5.1 Vetores

Vetores são variáveis compostas unidimensionais, pois utilizam um único índice, e homogêneas, pois só podem conter dados de um mesmo tipo.

Exemplo:

```
algoritmo "vetores"
    var
        i : inteiro
        vet_nomes : vetor[1..10] de caracter
    inicio
        para i de 1 ate 10 faca
            leia ( vet_nomes[i] )
        fimpara
        para i de 10 ate 1 passo -1 faca
            escreval ( vet_nomes[i] )
        fimpara
fimalgoritmo
```

A seguir, outro exemplo:

```
algoritmo "vetores2"
    var
        i : inteiro
        soma : inteiro
        vet_int : vetor[1..10] de inteiro
    inicio
        soma <- 0
        para i de 1 ate 10 faca
            leia ( vet_int[i] )
            soma <- soma + vet_int[i]
        fimpara
        escreva ( soma )
fimalgoritmo
```

5.2 Matrizes

Matrizes são variáveis compostas homogêneas bidimensionais, ou seja, diferenciam-se dos vetores pelo fato de utilizarem dois índices ao invés de um.

Exemplo:

```
algoritmo "matrizes"
    var
        i : inteiro
        j : inteiro
        mat_num : vetor[1..5, 1..4] de inteiro
    inicio
        para i de 1 ate 5 faca
            para j de 1 ate 4 faca
                mat_num[i, j] <- i + j
            fimpara
        fimpara
fimalgoritmo
```

5.3 Exercícios

1. Escreva um algoritmo com um vetor de 5 posições contendo os nomes das frutas: “Maça”, “Laranja”, “Pera”, “Uva” e “Limão”. O algoritmo deve imprimir “Uva”.
2. Escreva o algoritmo anterior (frutas), porém o usuário pode pesquisar qualquer uma das opções cadastradas. (ex.: pode digitar 1 e ver o resultado).

3. Escreva o mesmo algoritmo das frutas, porém o usuário poderá pesquisar pelo nome. Se não houver fruta cadastrada, ou o nome for digitado errado, o sistema deverá informar: “Fruta não cadastrada ou nome incorreto!”. Ao final, perguntar se deseja pesquisar outro nome.
4. Escreva um algoritmo que cadastre 10 nomes de clientes em um vetor de 10 posições e depois imprima todos na tela. Imprima também o nome do primeiro e do último cliente cadastrado.
5. Escreva um algoritmo que cadastre 10 nomes e seus respectivos telefones. Imprima cada nome com o respectivo telefone.
6. Escreva um algoritmo, para um pequeno sistema, que cadastre 12 nomes e seus respectivos telefones. Após o cadastro, o sistema pergunta se o usuário deseja fazer uma pesquisa pelo nome. Se encontrar, imprima o nome e seu respectivo telefone, caso contrário, imprima “cliente não encontrado”. Ao final perguntar se deseja fazer nova pesquisa. Quando o usuário digitar 0, o sistema é finalizado.
7. Escreva um algoritmo para um sistema de mercado contendo:

   a) Um menu com as opções: 1-Cadastrar produto, 2-Pesquisar Produto, 3-Pesquisar Preço, 4-Estoque Geral, 0-Sair do sistema;

   b) Na tela de cadastro de produto, cadastrar nome, preço, descrição do item e quantidade no estoque;

   Utilize 4 vetores para referência: Produto, Preço, Descrição e Quantidade
8. Escreva um algoritmo que cadastre um grupo com 5 de números pares e outro grupo com apenas 5 números ímpares. Imprima o resultado da soma do primeiro número do grupo par com o último número do grupo ímpar, o segundo número do grupo par com o penúltimo número do grupo ímpar e assim sucessivamente.
9. Dado uma matriz de ordem 3x3, faça um algoritmo que:

   a) Calcule a soma dos elementos da primeira coluna;

   b) Calcule o produto dos elementos da primeira linha;

   c) Calcule a soma da diagonal principal.

CAPÍTULO 6
MODULARIZAÇÃO

ALGORITMOS

Existe uma estratégia de resolução de problemas muito conhecida que diz que se resolve um grande problema dividindo-o em pequenas partes e resolvendo tais partes em separado. Pode-se exemplificar esta estratégia através de algoritmos, como no caso abaixo:

- Algoritmo para troca de um pneu furado
- Primeira solução: genérica.
- Tirar o pneu furado;
- Colocar o pneu novo.
- Segunda solução: refina-se cada passo anterior.
- Pegar as ferramentas;
- Erguer o carro;
- Tirar o pneu furado;
- Colocar o pneu novo;
- Baixar o carro;
- Guardar as ferramentas.

	*Obs.: Pode-se continuar refinando sucessivamente os passos da segunda solução.*

Essa orientação pode ser aplicada, e com excelente resultado, na construção de programas para computadores. Os profissionais de informática quando necessitam construir um grande sistema, o fazem dividindo tal sistema em partes, sendo então desenvolvida cada parte em separado. Mais tarde, tais partes serão acopladas para formar o sistema. Essa técnica, conhecida genericamente como técnica de refinamentos sucessivos (também conhecida como "top-down" ou ainda “dividir para conquistar”), ganhou na computação o título de modularização.

As partes em que um programa é dividido são conhecidas por vários nomes. Será adotado aqui uma destas nomenclaturas: sub-rotinas. Pode-se dizer que uma sub-rotina é um pedaço de código computacional que executa uma função bem definida, sendo que esta sub-rotina pode ser utilizadas várias vezes no programa. Neste curso, serão tratados dois tipos de sub-rotinas: procedimento e função.

ALGORITMOS

6.1 Procedimentos

Um procedimento é um tipo de sub-rotina que é ativada através da colocação de seu nome em alguma parte do algoritmo. Desta forma, assim que o nome de um procedimento é encontrado, ocorre um desvio no algoritmo, para que os comandos da sub-rotina sejam executados. Ao término da sub-rotina, a execução retornará ao ponto subsequente à chamada do procedimento.

Exemplo:

```
algoritmo "procedimento"
    var
        a : inteiro
        b : inteiro
        aux : inteiro

    procedimento ordena
    inicio
        se a > b entao
            aux <- a
            a <- b
            b <- aux
        fimse
    fimprocedimento

    inicio
        leia ( a )
        leia ( b )
        ordena
        escreva ( a )
        escreva ( b )
finalgoritmo
```

Segue outro exemplo:

```
algoritmo "processa_texto"
   var
        fim : caracter
        car : caracter
        cont_car : inteiro

    procedimento le_brancos
    inicio
        enquanto car = " " faca
            leia ( car )
        fimenquanto
```

```
    fimprocedimento

    procedimento processa_palavra
    inicio
        cont_car <- 0
        enquanto car <> " " e car <> fim faca
            cont_car <- cont_car + 1
            leia ( car )
        fimenquanto
        se cont_car > 0 entao
            escreval ( "Palavra de ", cont_car, " letras." )
        fimse
    fimprocedimento

    inicio
        fim <- "."
        leia ( car )
        le_brancos
        enquanto car <> fim faca
            processa_palavra
            le_brancos
        fimenquanto
fimalgoritmo
```

6.2 Escopo de Variáveis

Dá-se o nome de variáveis globais para aquelas variáveis que são definidas logo após o comando 'var' do programa principal, sendo desta forma visíveis em qualquer parte do programa.

Exemplo:

```
algoritmo "var_global"
    var
        nome : caracter

    procedimento alterar_nome
    inicio
        leia ( nome )
    fimprocedimento

    inicio
        alterar_nome
        escreva ( nome )
fimalgoritmo
```

No Exemplo acima, a variável 'nome' , por ser definida como global, pode ser manipulada dentro de qualquer ponto do programa, sendo que qualquer mudança no seu conteúdo será visível nas demais partes do programa.

Dá-se o nome de variáveis locais às variáveis que são declaradas dentro de uma sub-rotina, sendo que as mesmas só podem ser manipuladas dentro da sub-rotina que as declarou, não sendo visíveis em nenhuma outra parte do programa.

Exemplo:

```
algoritmo "var_local"

    procedimento escrever
    inicio
        var
            numero : inteiro
            max : inteiro
        inicio
            leia ( max )
            para numero de 1 ate max faca
                escreval ( numero )
            fimpara
    fimprocedimento

    inicio
        escrever
fimalgoritmo
```

É possível definir variáveis globais e locais com o mesmo nome, sendo que qualquer mudança no conteúdo da variável local não afetará o conteúdo da variável global e vice-versa, pois as duas variáveis ocupam posições de memória diferentes. Nesse caso, uma parte do programa que pertence ao escopo da variável local fica impedida de acessar a variável global.

Exemplo:

```
algoritmo "escopo"
    var
        x : inteiro

    procedimento alteracao
    inicio
        var
            x : inteiro
        inicio
```

```
            x <- 1
    fimprocedimento

    inicio
        x <- 0
        escreval ( "x antes: ", x )
        alteracao
        escreval ( "x depois: ", x )
fimalgoritmo
```

6.3 Passagem de Parâmetros

Como estratégia de programação, é desejável que cada sub-rotina não faça uso de dados externos, pelo menos de forma direta, e toda comunicação com o resto do programa seja feito através de parâmetros ou se for o caso, como retorno de função (retorno de funções será abordado posteriormente neste curso). A seguir, serão descritos os dois tipos de passagem de parâmetros:

- Passagem por valor: qualquer alteração no conteúdo de um parâmetro, dentro de uma sub-rotina, não será refletida na chamada da sub-rotina.
- Passagem por referência: qualquer alteração no conteúdo de um parâmetro, dentro de uma sub-rotina, se reflete na chamada da sub-rotina. A inserção da palavra 'var' antes da declaração de parâmetros, na definição da sub-rotina, indica que estes parâmetros serão passados por referência.

Exemplo:

```
algoritmo "parametros"
    var
        x : inteiro
        y : inteiro

    procedimento operacao( a : inteiro; var b : inteiro )
    inicio
        a <- a * 10
        b <- b * 100
        escreval ( "O valor de a é ", a, " e o valor de b é ", b )
    fimprocedimento

    inicio
        x <- 1
        y <- 2
        operacao( x, y )
```

```
        escreval ( "O valor de x é ", x, " e o valor de y é ", y )

fimalgoritmo
```

6.4 Funções

Uma sub-rotina do tipo função possui as mesmas características de um procedimento no que se refere a passagem de parâmetros e escopo de variáveis, mas possui uma importante diferença, que é o retorno de um valor ao término de sua execução, ou seja, uma função sempre deve retornar um valor ao comando que a chamou. Na definição da função, deve ser informado qual o tipo do valor de retorno.

Exemplo:

```
algoritmo "funcao"

    var
        valor1 : inteiro
        valor2 : inteiro
        maior_valor : inteiro
        resultado : real

    funcao media( x, y : inteiro ) : real
        retorne ( x + y ) / 2
    fimfuncao

    funcao maior( a, b : inteiro ) : inteiro
        se a > b entao
            retorne a
        senao
            retorne b
        fimse
    fimfuncao

    inicio
        valor1 <- 3
        valor2 <- 5
        maior_valor <- maior( valor1, valor2 )
        resultado <- media( valor1, 5 ) + media( valor2, 3 )

fimalgoritmo
```

6.5 Exercícios função

1. Escreva um algoritmo que leia dois números inteiros e calcule a média através de uma função. Imprima a média.
2. Escreva um algoritmo que leia dois números inteiros e calcule a multiplicação do primeiro pelo segundo número, através de uma função. Imprima o resultado.
3. Faça um algoritmo que leia uma temperatura em graus Fahrenheit e a converta para graus Celsius usando uma função. A fórmula de conversão é a seguinte: cel = ((far - 32) * 5) / 9.

6.6 Exercício geral

1. Escreva um sistema para urna eletrônica. O usuário poderá escolher entre 3 candidatos, conforme informação na tela (ex.: 1-Lula, 2-Collor, 3-Brizzola, 4-Nulo, 5-Branco). Após cada votação, o sistema pergunta se deseja encerrar as votações. Ao concluir esta etapa, a urna eletrônica deverá imprimir na tela:

   a) O número total de votos

   b) O nome do candidato vencedor

   c) O número total de cada candidato

   d) O percentual de cada candidato

   e) O total de votos nulos e brancos

   f) Percentual de nulos e brancos

   g) O número total de votos

2. Escreva um algoritmo que apresente o seguinte menu:

   01-Multiplicar

   02-Dividir

   03-Cadastrar nome e telefone

   04-Pesquisar nome

   05-Total de operações executadas

   06-Sair do sistema

   As opções 01 e 02 devem ser executadas através de uma função;

Para a opção 03, utilizar vetor;

A opção 05 registra um valor sempre que o usuário escolher uma opção, para imprimi-la no final;

Rua Monsenhor Celso, 256 - 1º andar - Curitiba – Paraná

41.3324.0015

41.99828.2468

Agradecimentos:

Equipe Elaborata Informática