Algoritmos e Programação: práticas de aula

por

Professora Me. Daniela Duarte da Silva Bagatini

Porto Alegre, 2011.

B144a Bagatini, Daniela Duarte da Silva

Algoritmos e programação: práticas de aula / Daniela Duarte da Silva Bagatini. – Porto Alegre, 2011.

121 p.

ISBN 978-85-65087-04-9

Bibliotecária responsável:

Maritza Silveira Martins

CRB 10/1741

Sumário

Parte I

1. INTRODUÇÃO

Lux e Furtado (2010) convida você para uma reflexão:

Imagine que o gerente de uma pequena fábrica necessita que seja realizada uma alteração nos preços dos produtos que a fábrica vende, em função de novos valores adotados pelo Governo para imposto sobre circulação de mercadorias. As diversas mercadorias estão catalogadas com seus preços de produção e preços de venda em um fichário e caberá a um funcionário novo e inexperiente fazer as alterações. Para auxiliar o trabalho do funcionário e garantir que o resultado seja correto, o gerente resolve passar-lhe as orientações por escrito e elabora o seguinte roteiro:

Instruções para cálculo dos novos preços das mercadorias:

1- *Pegue a ficha da mercadoria no fichário;*
2- *Leia o preço de produção da mercadoria;*
3- *Considere o percentual de lucro igual a 0.80;*
4- *Calcule o preço de venda conforme a fórmula:*

   preço de venda = preço de produção x (1.15 + percentual de lucro);

5- *Escreva na ficha o novo preço de venda.*

Você concorda que qualquer pessoa que siga a sequência de passos acima deverá concluir a tarefa satisfatoriamente? Basta que para isso possua os seguintes conhecimentos prévios:

1- Saber onde está o fichário com as fichas;

2- Saber ler e escrever;

3- Ser capaz de efetuar as operações de soma e multiplicação com números fracionários.

As instruções acima constituem um ***algoritmo*** que, em síntese, nada mais é do que ***a descrição de um conjunto de passos ou ações que visam atingir um objetivo determinado*** **(LUX e FURTADO, 2010).**

1.1 ALGORITMOS

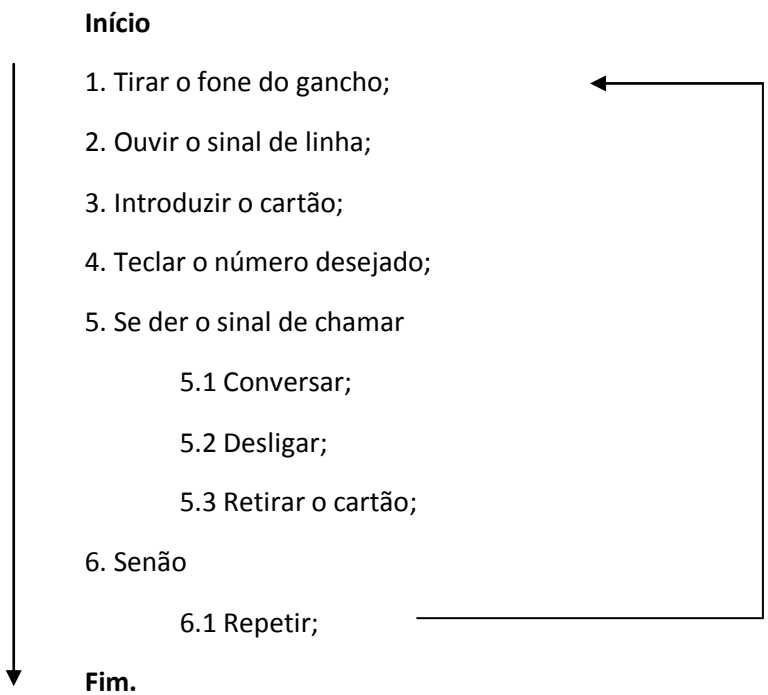
Uma definição completa e que salienta as características principais de um algoritmo é a que segue: "Um algoritmo é uma **sequência finita** de instruções, **bem definidas e não ambíguas** cada uma das quais podendo ser executada **mecanicamente** num **período de tempo finito** e com uma **quantidade de esforço finito**" (LUX e FURTADO, 2010).

- Trata-se de uma sequência finita de instruções: as informações apresentam-se em uma ordem, uma após a outra e são em número limitado;
- As instruções são bem definidas e não ambíguas, ou seja, são claras, não permitindo variadas interpretações;
- As instruções podem ser executadas mecanicamente;
- A execução das instruções encontra um término e depende uma quantidade determinada de esforço.

Garcia (2002) nos chama a atenção para o seguinte fato: "algoritmo não é a solução de um problema, pois, se assim fosse, cada problema teria um único algoritmo. Algoritmo é um conjunto de passos (ações) que levam à solução de um determinado problema,

ou então, é um caminho para a solução de um problema e, em geral, os caminhos que levam a uma solução são muitos."

Veja o exemplo apresentado a seguir por Garcia (2002). O algoritmo tem como objetivo indicar os passos para o uso de um telefone público. Perceba que este conjunto de passos pode ser descrito de forma distinta por diferentes pessoas, que podem chegar ao mesmo resultado, usar o telefone.



Ferting (2005) comenta que:

> *Diferente da linguagem natural, a linguagem de programação é dirigida a orientar a máquina e não pessoas. Máquinas não podem tomar decisões com base em premissas, Máquinas não podem escolher alternativas, mesmo que estas nos pareçam óbvias. Máquinas não podem corrigir comandos mal redigidos. Máquinas não podem descobrir a intenção do programador, mesmo que ela seja (ou pelo menos pareça) clara no contexto. Pessoas fazem tudo isso (pelo menos na maior parte das vezes) sem sequer notar. Por isso, a linguagem*

de programação precisa ter algumas características que a linguagem natural não tem.

Para que o computador execute as tarefas é necessário que estas estejam descritas em uma linguagem apropriada (que possui suas próprias regras) e, também, que nossas instruções possuam uma **lógica** que leve ao resultado esperado.

Podemos conceituar Lógica como **a arte de pensar corretamente**. O filósofo grego Aristóteles criou determinadas regras lógicas que conduzem a conclusões acertadas, quase como se fossem um cálculo. Tais regras constituem o Silogismo. O silogismo caracteriza-se por iniciar com uma sentença que afirme uma propriedade de uma classe, que é uma verdade maior ("Todos os metais são bons condutores de eletricidade"). A sentença seguinte apresenta um indivíduo pertencente à classe ("O mercúrio é um metal") e a terceira sentença conclui que o indivíduo possui a propriedade da classe (Logo, o mercúrio conduz bem a eletricidade).

Exemplo:

Todos os metais são bons condutores de eletricidade,
O mercúrio é um metal,
Logo, o mercúrio conduz bem a eletricidade.

Foi a Matemática, dentre todas as ciências, aquela que mais se aproximou da Lógica, por ser baseada no raciocínio dedutivo. Vejamos como exemplo, as duas proposições abaixo e a dedução que lhes segue:

3 + 1 = 4;
2 + 2 = 4;
Logo, 3 + 1 = 2 + 2.

A Lógica preocupa-se em avaliar quando uma proposição ou juízo é verdadeiro. Tomemos como exemplo a afirmação "Carlos é estudante", que exprime um estado mental (juízo) que será verdadeiro se estiver de acordo com o fato externo de Carlos estudar. Um juízo é falso se exprime ou une mentalmente o que na realidade está

separado. Utilizando a Lógica poderíamos escrever as instruções para o cobrador de entradas em um cinema:

"Se Carlos é estudante Então
cobre meia entrada
Senão
cobre entrada inteira."

A finalidade da disciplina de Algoritmos, fundamental em todos os cursos de Ensino Superior que preparam profissionais para atuar na área das Ciências Exatas, é fornecer os fundamentos teóricos básicos para o desenvolvimento do raciocínio lógico necessário na resolução de problemas computacionais (LUX e FURTADO, 2010).

Mas, o que são problemas computacionais?
São problemas resolvidos pelo computador relacionados com a manipulação de informações, sobre os quais iremos trabalhar este semestre na disciplina de Algoritmos e Programação. Desenvolver algoritmos significa também desenvolver o raciocínio lógico necessário para a resolução de problemas computacionais.

Assim, na disciplina, iremos abordar conceitos básicos sobre processamento de dados, organização de um computador, memória e tipos de dados, identificadores, estrutura de um algoritmo e programação. Também veremos expressões aritméticas, operadores aritméticos, funções matemáticas, expressões lógicas, operadores relacionais, operadores lógicos, tabelas verdade e prioridades entre operadores. Logo, trabalharemos com comandos de atribuição, de entrada e de saída, bem como, comandos de seleção e de repetição. Outro assunto que iremos abordar são os tipos estruturados homogêneos (vetores e matrizes) e heterogêneos e, arquivos. Todos estes assuntos serão explorados na prática e conhecimentos básicos da linguagem C.

Fica a dica: algoritmos só se aprende construíndo e testando. Pratique muitos exercícios!

1.2 PROGRAMAÇÃO ESTRUTURADA E PROGRAMAÇÃO ORIENTADA A OBJETOS

Os problemas computacionais, para que possam ser resolvidos por um computador, deverão estar escritos em uma **linguagem computacional**. Isto significa que, após ser elaborado o algoritmo que resolve o problema, este deverá ser traduzido para uma linguagem conhecida pelo computador. A linguagem que utilizaremos para testar nossos algoritmos no computador chama-se **C**. Esta linguagem foi criada em 1972 por Dennis M. Ritchie e Ken Thompson e tem sido muito usada desde então. A linguagem C foi projetada para a construção de sistemas computacionais, um exemplo de sistema operacional contruído em C é o UNIX. É uma linguagem poderosa, portátil e flexível, que permite planejar programas estruturados.

Existem em programação vários **paradigmas** (formas de programar), como: Programação Estruturada, Programação Orientada a Objetos, Programação Lógica. Trataremos aqui das características dos dois primeiros, por serem mais utilizados na construção de programas comerciais.

A Programação Estruturada é o paradigma utilizado por linguagens como Pascal, C e Fortran. Baseia-se na estruturação dos programas, ou seja, consiste em subdividir o problema a ser resolvido em tarefas, que vão constituir pequenos blocos, independentes entre si e acioná-los a partir de um bloco principal. Tal característica torna mais fácil a tarefa de testar programas, uma vez que os blocos podem ser testados em separado e, também, auxilia na manutenção dos códigos. Quando analisamos um problema sob este paradigma, nossa primeira preocupação é compreender e definir muito bem **a ação** que proporcionará o resultado a atingir e, depois, definirmos os dados que estarão envolvidos na tarefa.

Já a Programação Orientada a Objetos, que vem tendo uso crescente nos últimos anos, utiliza o paradigma Orientado a Objetos, desta forma preocupa-se com **o objeto**, que basicamente possui características (atributos ou dados) e propriedades (métodos ou ação). Exemplos de linguagens orientadas a objeto são: Pearl, C++ e Java.

Para melhor exemplificar as diferenças básicas entre as duas, analisemos o problema de calcular a média aritmética de um aluno, sendo os dados de entrada as duas notas e, como saída, queremos a sua média aritmética.

Ao resolvermos este problema utilizando o raciocínio da programação estruturada, vamos nos preocupar em conhecer bem e depois descrever a *ação de calcular* a média aritmética (m = (nota1 + nota2)/2) e vamos nos utilizar (de forma secundária) de dados para executar a ação, ou seja, necessitaremos de espaço em memória para armazenar as duas notas e o resultado do cálculo e, todas são do tipo numérico real.

Já pelo paradigma da programação orientada a objetos, vamos, primeiramente, nos preocupar com o *objeto média* que possui características próprias e que vamos descrever apontando seus *atributos* e seus *comportamentos*. Os atributos (ou dados) serão as notas e os comportamentos ou métodos serão, neste caso, a fórmula para cálculo da média aritmética. Este modelo de programação, como vincula a um objeto seus dados e métodos, trouxe um novo conceito à programação de computadores, que é a possibilidade de agruparmos objetos com as mesmas características e com eles formarmos o que se chama *classes.* Criamos classes de objetos para não repetirmos a definição de propriedades iguais para objetos diversos que as possuem. Tomemos como exemplo o problema **mesa** e vamos analisá-lo sob o paradigma da orientação a objetos. Existem mesas de diversos tipos, formas e materiais, porém todas têm: atributos (um tampo e pernas de sustentação) e métodos (sustentar outros objetos sobre o tampo).

Com esta definição, criamos a classe mesa e dela podem derivar vários objetos: mesa com tampo redondo e quatro pernas; mesa fixa na parede com duas pernas e tampo retangular de madeira e outras tantas mais.

Uma boa definição para classe é:

"Uma classe é um agrupamento de objetos que revelam semelhanças entre si, no aspecto estrutural e funcional".

Os objetos pertencentes a uma classe, dela herdarão os atributos e métodos e acrescentarão suas particularidades.

Quando, dentro deste paradigma, chegamos no nível de resolução dos métodos dos objetos, voltamos a utilizar o paradigma da programação estruturada, preocupando-nos com a ação. Podemos, portanto, dizer que a Programação Orientada a Objetos inclui os conhecimentos básicos da Programação Estruturada. Em nossa disciplina vamos nos preocupar em aprender esta forma estruturada de programar que envolve o aprendizado de regras básicas, instruções e comandos da lógica de desenvolvimento de algoritmos e, em disciplinas posteriores, utilizá-la novamente, como nas disciplinas que utilizam orientação a objetos.

Vamos praticar?

1. Escreva um algoritmo que apresente os passos necessários para trocar o pneu de um carro. Considere que estão disponíveis no porta-malas um macaco, uma chave de roda e um pneu reserva em boas condições.
2. Escreva um algoritmo para jogar o jogo da forca.
   Para escrever os algoritmos você pode utilizar a estruturação apresentada no exemplo do telefone público.

Referências bibliográficas:

Ziviani, Nivio. Projeto de Algoritmos em C e Java. 2ª ed. São Paulo: Thomson Pioneira, 2004

Feofiloff, Paulo. Algoritmos em Linguagem em C. 1ª ed. Rio de Janeiro: Campus, 2008

Fertig, Cristina; Medina, Marco. Algoritmos e Programação Teoria e Prática. 1ª ed. São Paulo: Novatec, 2005

Oliveira, Álvaro; Boratti, Isaias. Introdução a Programação de Algoritmos. 3ª ed. Florianópolis: Visual Books, 2007

Garcia, Guto; Lopes, Anita. Introdução à Programação: 500 Algoritmos Resolvidos. 1ª ed. Rio de Janeiro: Campus, 2002

Lux, Beatriz; Furtado, João Carlos. Apostila de Algoritmos e Programação. Departamento de Computação, UNISC, 2010

Parte II

2. CONCEITOS BÁSICOS

2.1 PROBLEMAS COMPUTACIONAIS

Problemas computacionais são aqueles que estão relacionados com a manipulação de informações. Um computador é, basicamente, uma máquina que recebe informações, as processa e fornece o resultado deste processamento, que são, novamente, informações (LUX e FURTADO, 2010).

Os dados que entram no computador, informados pelo usuário ou programa, chegam através de dispositivos de entrada (como o teclado ou o *driver* de disco) e são armazenados na memória RAM do computador. Logo, a Unidade Central de Processamento (CPU) é responsável por executar as instruções e operações necessárias para obter o resultado esperado, tendo como fonte os dados de entrada. Os dados produzidos pelo processamento são, também, armazenados na memória RAM e chegam ao usuário através de dispositivos de saída (como o vídeo, o *driver* de disco ou a impressora). Na memória do computador, ficam armazenados o sistema operacional da máquina, o programa e as informações ou dados.

Assim, um algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída.

2.2 TIPOS DE DADOS

Os dados também são diferenciados entre constantes ou variáveis (LUX e FURTADO, 2010):

1. Constante – a informação é dita constante quando não existe a possibilidade de variação no seu valor em tempo de execução do algoritmo.

Exemplo: no cálculo da área de um círculo qualquer, o valor de *Pi* será sempre o mesmo, não importando qual seja o círculo, o que o caracteriza como constante.

2. Variável – a informação é dita variável quando se admite que seu valor possa sofrer alterações durante a execução do algoritmo, ou que ela possa assumir diferentes valores a cada nova execução do algoritmo.

Exemplo: no cálculo da área de um círculo qualquer, o raio do círculo é uma informação variável, dependendo do círculo para o qual se deseja calcular a área.

A atribuição de um valor a uma variável é representada pelo símbolo **=**

2.3 TIPOS PRIMITIVOS DE DADOS

Existem três tipos de dados usados pelo computador: numéricos, alfanuméricos (caracteres) ou lógicos (booleanos). Ao tipo do dado está associado um tamanho necessário em memória para armazená-lo e as operações que o tipo suporta. Isto é necessário para que o computador esteja preparado para operá-los e armazená-los. Assim, os dados ou informações que são processados por um computador deverão ter seu tipo declarado, o que pode ser entendido como classificar a informação quanto ao seu tipo e solicitar um espaço que comporte seu valor, para armazenagem na memória RAM do computador (LUX e FURTADO, 2010).

São quatro os tipos primitivos:

1. Inteiro (int) - qualquer informação que assuma valores numéricos pertencentes ao conjunto dos números inteiros (por exemplo, -34, 90, 0).

Exemplo: **inteiro** quantidade;
inteiro idade;

2. Real (float) - qualquer informação que pode assumir valores numéricos pertencentes ao conjunto dos números reais (por exemplo, 23.56, -34.00, 0.0).

Exemplo: **real** preco;
real notas;

3. Caractere (char) - qualquer informação que pode assumir os caracteres do teclado, tanto alfanuméricos como especiais (por exemplo, 'aula', '5 de fevereiro', '178', '0.55').

Exemplo: **caractere** nomeAluno;
caractere profissao;

4. Lógico (ou booleano) - qualquer informação que possa assumir apenas os valores verdadeiro ou falso.

Exemplo: **lógico** ligado;
lógico aprovado;

2.4 FORMAÇÃO DE IDENTIFICADORES

Identificadores são os nomes que criamos e associamos as variáveis e constantes de um algoritmo e que devem observar as regras abaixo (LUX e FURTADO, 2010):

1. Devem ser formados por um ou mais caracteres, sendo que o primeiro deverá ser uma letra.
2. Caracteres especiais não são válidos (por exemplo: % * + ç ã ; : @ ' ` " ?).
3. Não podem conter espaço.
4. O caractere de sublinha (_) é considerado válido.

Exemplo: a palavra "Média Aritmética" não pode ser usada como identificador, por utilizar o acento agudo que é um caractere especial e por conter espaço. Para tornar-se válido, deveria ser utilizado: **Media_Aritmetica**, **mediaAritmetica** ou **MediaAritmetica**.

Vamos, também, considerar diferenças quanto aos caracteres maiúsculos e minúsculos, de tal forma que os identificadores **Aluno**, **aLuno** e **aluno** correspondem a diferentes variáveis ou constantes. Com isso, estaremos seguindo as regras da linguagem C. Algumas linguagens não fazem esta diferença.

Deve-se observar que não é permitido utilizar o mesmo identificador para nomear diferentes variáveis e/ou constantes ou nomear o algoritmo.

2.6 ESTRUTURA DE UM ALGORITMO

Existem muitas formas de representação de um algoritmo. Na nossa disciplina utilizaremos o pseudocódigo, que tem um grau de rigidez sintática intermediária, entre a linguagem natural e de programação.

Estruturar um algoritmo facilita a legibilidade e compreensão do mesmo. Ao elaborarmos um algoritmo na forma estruturada de programação, devemos observar as seguintes subdivisões (LUX e FURTADO, 2010):

(a) Cabeçalho do algoritmo:

Serve para nomear o algoritmo. Deve iniciar pela palavra **algoritmo** (ou programa) e ser seguido pelo **identificador** do algoritmo (nome) e o sinal **;** (ponto e vírgula). O sinal de ponto e vírgula é utilizado ao longo de todo o algoritmo como separador de instruções.

Exemplos: algoritmo area_doCirculo;
algoritmo raizQuadrada;

(b) Zona de declarações:

Nesta parte do algoritmo são definidas as constantes e declaradas as variáveis necessárias ao seu processamento.

- Definição de Constantes

A declaração de cada constante deve ser precedida pela palavra ***const*** e a seguir é colocado o **identificador** da constante e o **valor** que possui, finalizando com o separador **;** (ponto e vírgula).

Exemplos: algoritmo area_doCirculo;

const PI 3.1416;

const NOME “Jose”;

- Declaração de Variáveis

Declara-se o **tipo** da variável para que seja reservada uma quantidade de memória apropriada para armazenar um dado e indicar que seu conteúdo será referenciado pelo **nome** que foi escolhido.

Exemplo: inteiro idade;

Com a declaração do exemplo acima estaremos reservando um espaço em memória, chamado *idade*, com tamanho suficiente para guardar um número do tipo inteiro e toda vez que usarmos a palavra i*dade* nas instruções do algoritmo, estaremos nos referindo ao valor numérico que está armazenado neste espaço da memória do computador.

Exemplo: real area, raio;

Pelo exemplo, pode-se ver que, havendo mais de uma variável do mesmo tipo, elas podem ser declaradas na mesma linha, separadas por vírgula.

(c) Corpo do algoritmo

Faz parte do corpo do algoritmo toda a seqüência de instruções que conduz à resolução do problema. O corpo do algoritmo tem seu começo determinado pela palavra **inicio**, seguida por todas as instruções separadas por **;** (ponto e vírgula) e termina com a palavra **fim**.

Exemplo:

algoritmo area_doCirculo;
real area, raio;
const PI 3.1416;
inicio
instrução 1;
instrução 2;
-
-
instrução n;
fim.

Exemplo em C:

```
#include<stdio.h>
#define PI 3.1416
void main(void)
{
float  area, raio;
        instrução 1;
        instrução 2;
        -
        -
        instrução n;
        }
```

Após a construção do algoritmo em pseudocódigo, é necessário que mais um passo seja executado para que o algoritmo possa ser compilado e posteriormente executado pelo computador.

2.7 EXERCÍCIOS PROPOSTOS

1) Dentre os identificadores abaixo, indique quais nomes estão corretos (C) e quais estão incorretos (I). Para os incorretos, justifique sua resposta.

a) X1YZ

b) A9PQ

c) GF,3W

d) 123casa

e) Lápis_cor

f) Aula hoje

g) inteiro

2) Escreva o tipo de dado ideal para se representar cada uma das seguintes informações.

a) O número de telefone

b) Número de estudantes da universidade

c) Se uma pessoa gosta de bala ou não

d) O valor de uma casa

e) A cor de um carro

f) Resultado de uma divisão

3) Identifique os tipos dos seguintes dados.

a) ’13 de agosto’

b) 0.8

c) ‘?’

d) 27

e) -27

Referências biliográficas:

Ziviani, Nivio. Projeto de Algoritmos em C e Java. 2ª ed. São Paulo: Thomson Pioneira, 2004

Feofiloff, Paulo. Algoritmos em Linguagem em C. 1ª ed. Rio de Janeiro: Campus, 2008

Fertig, Cristina; Medina, Marco. Algoritmos e Programação Teoria e Prática. 1ª ed. São Paulo: Novatec, 2005

Oliveira, Álvaro; Boratti, Isaias. Introdução a Programação de Algoritmos. 3ª ed. Florianópolis: Visual Books, 2007

Garcia, Guto; Lopes, Anita. Introdução à Programação: 500 Algoritmos Resolvidos. 1ª ed. Rio de Janeiro: Campus, 2002

Lux, Beatriz; Furtado, João Carlos. Apostila de Algoritmos e Programação. Departamento de Computação, UNISC, 2010

Parte III

3. EXPRESSÕES, OPERADORES E FUNÇÕES

Agora iremos conhecer os diversos operadores e funções que serão utilizados na construção dos algoritmos (LUX e FURTADO, 2010).

3.1 EXPRESSÕES ARITMÉTICAS

São aquelas cujos operadores são aritméticos e cujos operandos são constantes e/ou variáveis do tipo numérico (inteiro e/ou real).

Exemplos: a = b * 5;

y = n * (b +c);

3.1.1 OPERADORES ARITMÉTICOS

+	**Adição**
-	**Subtração**
*	**Multiplicação**
/	**Divisão**
%	**Resto inteiro da divisão entre inteiros**

3.1.2 FUNÇÕES MATEMÁTICAS

- Sempre retornam um valor (resultado), cujo tipo depende do que está definido para cada função.
- Dependem sempre dos parâmetros (argumentos), que são colocados dentro dos parênteses, separados por vírgula, se houver necessidade de mais de um.

Alguns exemplos de funções:

Resultado	Função	Tipo do argumento	Tipo do resultado
Logaritmo neperiano	log(x)	real	real
Logaritmo base 10	log10(x)	real	Real
e elevado à potencia x	exp(x)	real	Real
Valor absoluto de x fabs(-2.5) = 2.5	fabs(x)	real, int	Real
abs(-5) = 5	abs(x)	int, real (trunca o real)	Int
Decompõe um real x em duas partes: y recebe a parte fracionária e a função devolve a parte inteira, ambas como real.	modf(x,y)	real, real	Real
Arredonda o valor de x para cima ceil(5.4) = 6.0	ceil (x)	real	Real
Arredonda o valor de x para baixo floor(5.4) = 5.0	floor(x)	real	Real
Retorna x elevado a potência y pow(2,2) = 4	pow(x,y)	real	Real
Raiz quadrada de x sqrt(36) = 6	sqrt(x)	real	real
Retorna o resto da divisão de x por y fmod (25.55 , 2) = 1.55	fmod(x,y)	real, real	real
Seno de x	sin(x)	real	Real
Cosseno de x	cos(x)	real	Real
Arco-tangente de x	atan(x)	real	Real

e = constante de Euler *= 2.718*

3.1.3 PRIORIDADES NAS EXPRESSÕES ARITMÉTICAS

1) Parênteses mais internos

2) - (menos unário)

3) Funções matemáticas

4) * / %

5) + -

***Observação**: Nos casos de operadores com a mesma prioridade, a expressão será executada observando a sequência de operadores da esquerda para a direita.*

3.2 EXPRESSÕES LÓGICAS

São expressões onde os operandos são relações e/ou variáveis do tipo lógico e os operadores são relacionais ou lógicos.

Exemplos: A > B

(X == Y) OU (X == Z)

3.2.1 OPERADORES RELACIONAIS

Uma relação é uma comparação realizada entre valores do mesmo tipo, que podem ser constantes, variáveis ou expressões. O resultado de uma operação relacional será sempre lógico (verdadeiro ou falso).

São os seguintes os operadores relacionais:

==	**Igualdade**
>	**Maior**
<	**Menor**
<>	**Diferente**
<=	**Menor ou igual**
>=	**Maior ou igual**

- *Observação: na linguagem C, o sinal de diferente é* ***!=***

3.2.2 OPERADORES LÓGICOS

São os seguintes os operadores lógicos: E, OU, NÃO.

Operador E (&&): é utilizado quando necessitamos estabelecer que todos os relacionamentos lógicos de uma sentença devem ser verdadeiros.

Exemplo: Se (Media_aluno >= 7) **E** (Presenca_aluno >= 0.75) Então considere o aluno aprovado

Tabela verdade do operador E: apresenta o resultado obtido de todas as combinações possíveis do operador.

Condição 1	Condição 2	Condição 1 E Condição 2
V	**V**	**V**
V	**F**	**F**
F	**V**	**F**
F	**F**	**F**

Operador OU (||): é utilizado quando necessitamos estabelecer que pelo menos uma das relações lógicas de uma sentença seja verdadeira. É também conhecido como OU INCLUSIVO.

Exemplo: Se (Media_aluno < 7) OU (Presenca_aluno < 0.75) Então considere o aluno reprovado

Tabela verdade do operador OU:

Condição 1	Condição 2	Condição 1 OU Condição 2
V	**V**	**V**
V	**F**	**V**
F	**V**	**V**
F	**F**	**F**

Operador NÃO (!): é utilizado quando é necessário estabelecer o valor contrário de uma determinada condição.

Exemplo: Se NÃO(Media_aluno) >= 7 Então

considere aluno reprovado

Tabela verdade do operador NÃO:

Condição	NÃO Condição
V	**F**
F	**V**

3.2.3 PRIORIDADE ENTRE OS OPERADORES LÓGICOS:

1) NÃO
2) E
3) OU

3.3 PRIORIDADE ENTRE TODOS OS OPERADORES:

1) Parênteses mais internos

2) – (menos unário)

3) Funções matemáticas

4) Operadores aritméticos: 4.2 * / %

4.3 + -

5) Operadores relacionais

6) Operadores lógicos 6.1 NÃO

6.2 E

6.3 OU

Nos casos de operadores com a mesma prioridade, será observada a ordem de execução da esquerda para a direita.

Exemplos:

a) NÃO verdadeiro **E** pow(3, 2) / 3 < 15 – 5 % 7
F **E** 9.0 / 3 < 15 – 5

F **E** 3.0 < 10

F **E** V

F

b) Considerando as variáveis a seguir, forneça o resultado das relações:
CIDADE = “SÃO PAULO”

PESO = 54

BOOLEAN = V

1. CIDADE = “GRAMADO” **E** NÃO BOOLEAN
F **E** NÃO V

F **E** F

F

2. (sqrt(196)) = 14 **OU** PESO = 55
   14.0 = 14 **OU** F

   V **OU** F

   V

3.4 EXERCÍCIOS PROPOSTOS

1) Considerando X = 1, Y = 2 e Z = 5, desenvolva:

a) (Z % Y / Y)

b) X + Y + Z / 3

c) (sqrt(Z / Y + X * Y))

d) Z – abs(X – sqrt(Y * Y))

2) Sendo a = VERDADEIRO e b = FALSO, qual o resultado das expressões abaixo?

a) NÃO a E b OU a E NÃO b

b) NÃO(NÃO(a OU b) E (a OU b))

c) a OU b E NÃO a OU NÃO b

d) (a OU b) E (NÃO a OU NÃO b)

3) Resolva as relações abaixo:

a) Falso OU 20 % 18 / 3 <> 18 / 3 % 20

b) 2 + 8 % 7 >= 3 * 6 - 15 E 3 * 5 / 4 <= pow(2, 3) / 0.5

c) ceil(20 / 3) > sqrt(36) OU floor(18 / 4) >= pow(2, 2)

d) NÃO(5 <> 10 / 2 OU Verdadeiro E 2 - 5 > 5 - 2 OU Verdadeiro)

e) Verdadeiro E 40 / 4 + 3 < 150 / 3

f) Z = 12.55

ceil(Z) / 2 + 50 % 5 <> abs(-20 / 3)

g) fabs(-50 / 2) – 20 > pow(2, 4) OU (13 / 2) < sqrt(36)

h) X = 7

X == sqrt(45 + 4) OU NÃO (6 + 2 <> floor(8.25)) E 9 < 9 % 1

i) A = 8

abs(pow(A, 2) + 0.1 * (-1)) > fmod(25.5 * 2, 2) OU FALSO

4) Escreva o valor que será atribuído a cada uma das variáveis:

a= 3 + 4 * 5;

b= 8 / 4 + 2 * 3;

c= 2 * (10 – 3 * 3) – 1;

d= 5 * (3 + (2 + 3)) / 2 + 1;

e= 1 + 12 / ((7 + 2) / 3) + (6 - 2);

f= 3 + 15 / 2 + 5;

g= 21 / 4 – 2;

h= modf(11, 4) + 8 / 3;

i= sqrt(9) + pow(9, 3);

j= 21 / 4 / 2;

5) Os pares de instruções abaixo produzem o mesmo resultado?

a= (4 / 2) + (2 / 4); **a**= 4 / 2 + 2 / 4;

b= 4 / (2 + 2) / 4; **b**= 4 / 2 + 2 / 4;

c= (4 + 2) * 2 – 4; **c**= 4 + 2 * 2 – 4;

6) Reescreva as instruções a seguir com o mínimo de parênteses possível sem alterara o resultado.

a= 6 * (3 + 2);

b= 2 + (6 * (3 + 2));

c= 2 + (3 * 6) / (2 + 4);

d= 2 * (8 / (3 + 1));

e= 3 + ((13 - 2) / (2 * (9 - 2)));

f= (6 / 3) + (8 / 2);

g= ((3 + (8 / 2)) * 4) + (3 * 2);

h= (6 * (3 * 3) + 6) – 10;

i= (((10 * 8) + 3) * 9);

j= ((-12) * (-4)) + (3 * (-4));

7) a= 5, b= 3 e c= 2, responda se as seguintes expressões lógicas são verdadeiras ou falsas.

a > 3 E b > 5

a > 3 OU b > 5

NÃO (a = 5)

c < 1 OU c <> b E a < b

c = b – 2 OU NÃO (c <> b E a < b)

Referências biliográficas:

Ziviani, Nivio. Projeto de Algoritmos em C e Java. 2ª ed. São Paulo: Thomson Pioneira, 2004

Feofiloff, Paulo. Algoritmos em Linguagem em C. 1ª ed. Rio de Janeiro: Campus, 2008

Fertig, Cristina; Medina, Marco. Algoritmos e Programação Teoria e Prática. 1ª ed. São Paulo: Novatec, 2005

Oliveira, Álvaro; Boratti, Isaias. Introdução a Programação de Algoritmos. 3ª ed. Florianópolis: Visual Books, 2007

Garcia, Guto; Lopes, Anita. Introdução à Programação: 500 Algoritmos Resolvidos. 1ª ed. Rio de Janeiro: Campus, 2002

Lux, Beatriz; Furtado, João Carlos. Apostila de Algoritmos e Programação. Departamento de Computação, UNISC, 2010

4. COMANDO DE ATRIBUIÇÃO, BLOCOS E COMANDOS DE ENTRADA E SAÍDA DE DADOS

4.1 COMANDO DE ATRIBUIÇÃO

O comando de atribuição é utilizado para atribuir um valor a uma variável. Este valor deverá ser do mesmo tipo da variável, por exemplo: se a variável é do tipo inteiro, somente um valor inteiro poderá ser atribuído a ela. Utiliza-se para simbolizar a atribuição o sinal = e lê-se "recebe".

Sintaxe do comando atribuição: **Identificador = valor;**

Exemplo:

```
algoritmo area_doCirculo; //Algoritmo que, dado o raio de um círculo, calcula a sua área

const PI 3.1416;
real area, raio;
inicio
        raio=  5;
        --------------------;
fim.
```

Exemplo em C:

```
//Algoritmo que, dado o raio de um círculo, calcula a sua área
#include<stdio.h> // bibliotecas das funções do C

#define PI 3.1416 // definição de constantes
void main(void)
{
float   area, raio; // definição de variáveis

        raio= 5; // atribuição de conteúdo à variável
        }
```

No algoritmo acima, a instrução **raio= 5.0;** estabelece que a variável raio recebe o valor **5.0**, o que em termos de programação, significa que a variável raio armazenará na memória do computador o valor 5.

Exemplo:

```
algoritmo area_doCirculo; //Algoritmo que, dado o raio de um círculo, calcula a sua área
const PI 3.1416;
real area, raio;
inicio
        raio= 5;
        raio= 20.78; // nova atribuição de conteúdo à variável
fim.
```

Exemplo em C:

```
//Algoritmo que, dado o raio de um círculo, calcula a sua área
#include<stdio.h> // bibliotecas das funções do C

#define PI 3.1416 // definição de constantes
void main(void)
{
float   area, raio; // definição de variáveis

        raio= 5; // atribuição de conteúdo à variável

        raio= 20.78; // nova atribuição de conteúdo à variável
        }
```

No algoritmo acima, a primeira instrução de atribuição à variável **raio** armazena em memória o valor real 5, mas, seguindo as instruções sequencialmente, o computador encontra outra atribuição para **raio**, onde ele armazena 20.78. Com isto, o valor anterior (5) é substituido pelo novo valor, 20.78. O valor 5 que havia sido anteriormente armazenado é “perdido”.

4.2 COMANDO DE ENTRADA DE DADOS

O comando de entrada (leitura) tem por função transferir dados do meio externo para a memória de trabalho do computador. O valor digitado será armazenado na variável.

Sintaxe do comando de entrada de dados: **leia(identificador_da_variavel);**

Exemplo em C: **scanf("%d", &identificador);**

Exemplo:

```
//Algoritmo que, dado o raio de um círculo, calcula a sua área
algoritmo area_doCirculo;

const PI 3.1416 // definição de constantes
real    area, raio; // definição de variáveis
inicio
        leia(raio); // lê conteúdo, entrada de dados
        area= PI * pow(raio, 2);
        fim.
```

Exemplo em C:

```
//Algoritmo que, dado o raio de um círculo, calcula a sua área
#include<stdio.h> // bibliotecas das funções do C

#include<math.h> // bibliotecas das funções do C, como pow

#define PI 3.1416 // definição de constantes
void main(void)
{
float   area, raio; // definição de variáveis

        printf("\n Digite o valor do raio: ");
        scanf("%f", &raio); // lê conteúdo, entrada de dados
        area= PI * pow(raio, 2);
        }
```

No exemplo acima, a instrução **leia**(raio); permite que seja fornecido um valor válido para a variável raio. Desta forma, após executada a instrução, a variável armazenará na memória do computador o valor que foi lido *(lembre-se: se este for do mesmo tipo da variável!).* Logo, será executada a próxima instrução que atribui à variável "area" o resultado da expressão Pi*pow(raio,2).

4.3 COMANDO DE SAÍDA DE DADOS

O comando de saída (escreva) é utilizado para apresentar valores ao meio externo.

Sintaxe do comando de saída de dados: **escreva("texto", identificador_da_variavel);**

Exemplo em C: **printf("texto");**

Exemplo:

```
//Algoritmo que, dado o raio de um círculo, calcula a sua área
algoritmo area_doCirculo;

const PI 3.1416 // definição de constantes
real    area, raio; // definição de variáveis
inicio
        leia(raio); // lê conteúdo, entrada de dados
        area= PI * pow(raio, 2);
        escreva("A área do círculo de raio", raio, "é", area,
"onde considerou-se o valor de PI=", PI); // saída de dados
        fim.
```

Exemplo em C:

```
//Algoritmo que, dado o raio de um círculo, calcula a sua área
#include<stdio.h> // bibliotecas das funções do C

#include<math.h> // bibliotecas das funções do C, como pow

#define PI= 3.1416 // definição de constantes
void main(void)
```

```
{
float   area, raio; // definição de variáveis

        printf("\n Digite o valor do raio: ");
        scanf(“%f”, &raio); // lê conteúdo, entrada de dados
        area= PI * pow(raio, 2);
        printf("\n A área do círculo de raio %.2f é %.2f onde considerou-se o valor de PI= %f", raio, area, PI); // saída de dados

        }
```

O comando escreva poderá fornecer apenas o conteúdo de uma variável ou, se quisermos, poderemos adicionar um texto, como no exemplo acima. Todos os caracteres que configuram um texto deverão estar entre aspas duplas (inclusive o espaço) e devemos separar com vírgula o texto das variáveis (ou constantes). Se, ao executarmos o algoritmo acima, fornecêssemos o valor 2 para raio, a instrução escreva iria indicar como saída o seguinte:

A área do círculo de raio 2 é 12.5664, onde considerou-se o valor de PI = 3.1416

Exemplo completo:

```
//Algoritmo que, dado o raio de um círculo, calcula a sua área
algoritmo area_doCirculo;

const PI 3.1416 // definição de constantes
real    area, raio; // definição de variáveis
inicio
        leia(raio); // lê conteúdo, entrada de dados
        area= PI * pow(raio, 2);
        escreva("A área do círculo de raio”, raio, ”é”, area, ”onde considerou-se o valor de PI=”, PI); // saída de dados
fim.
```

Exemplo completo em C:

```
#include<stdio.h> // bibliotecas das funções do C, como: scanf e printf
```

```
#include<conio.h> // bibliotecas das funções do C, como: getchar
#include<math.h> // bibliotecas das funções do C, como: pow
#define PI 3.1416 // definição de constantes
int main(void)
{
 float   area, raio; // definição de variáveis

 printf("\n Digite o valor do raio: ");
 fflush(stdin);
 scanf("%f", &raio); // leitura de dados
 area= PI * pow(raio, 2);
 printf("\n A área do círculo de raio %.2f é %.2f onde considerou-se o
        valor de PI= %f", raio, area, PI); // saída de dados

 fflush(stdin); // limpa buffer do teclado, funciona junto com entrada
de dados
 getchar(); // parada da tela
}
```

4.4 EXERCÍCIOS PROPOSTOS

Exercícios identificados com (*) estão resolvidos ao final do capítulo.
Implemente todos algoritmos construídos.
Este material segue as referências bibliográficas indicadas ao final, principalmente de Lux e Furtado (2010).

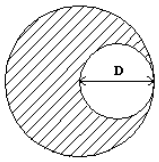
1. Se x possui o valor 15 e forem executadas as seguintes instruções:
   X = X + 3;
   X = X - 6;
   X = 3 * X;
   qual será o valor armazenado em X?

2. (*) Escreva um algoritmo que tenha como entrada o nome de um funcionário, o número de horas que trabalhou e o número de filhos com menos de 18 anos que possui. O algoritmo deverá fornecer como saída o salário que este funcionário irá receber, considerando:
   - Um valor igual a R$ 10.00 por hora trabalhada;
   - Um acréscimo de R$50.00 por filho;
   - Um desconto de 9% como contribuição ao INSS, sobre o total de ganhos.

3. Escreva um algoritmo que recebe como entrada três valores numéricos reais (X, Y e Z), calcula e informa:
   - A área do triângulo que tem X por base e Y por altura
     área do triângulo = *(base * altura) / 2*
   - O comprimento da circunferência de raio Z
     comprimento da circunferência = *2 * PI * raio*
   - A área do trapézio que tem X e Z por bases e Y por altura
     área do trapézio = (*(base maior + base menor) * altura) / 2*
   - A área do retângulo de lados Y e Z
     área do retângulo = *base * altura*
   - A área da superfície de um cubo que tem X por aresta
     área da superfície = *6 * aresta*^{2}

4. (*) Escreva um algoritmo que leia dois valores numéricos inteiros, armazenando-os em duas variáveis (A e B). O algoritmo deverá trocar o valor das duas variáveis, de tal forma que em A fique armazenado o valor de B e vice-versa. Ao final, o algoritmo deverá escrever os valores de A e B.

5. Faça um algoritmo para calcular a área da figura hachurada a seguir, sabendo que o diâmetro do círculo menor é igual ao raio do círculo maior. Calcule, também, a percentagem da área do círculo menor sobre a área do círculo maior. *Área do círculo= PI * raio*^{2}



6. (*) Escreva um algoritmo para calcular e informar o valor de conversão para graus centígrados de uma temperatura em graus Fahrenheit, sabendo que *C = (5 * (F - 32))/*9.

7. Escreva um algoritmo para calcular o consumo médio de combustível de um automóvel em Km/l. Considere como dados de entrada: quilometragem inicial, quilometragem final, volume de combustível consumido.

8. Escreva um algoritmo que leia dois valores numéricos do tipo inteiro e escreva o resultado da soma dos quadrados destes números e o resultado da divisão dos dois valores.

9. Faça um algoritmo para calcular o valor de *Y* como função de *X* segundo a função

   $Y= (5X + \sqrt{2})^2/X^2$, considerando um domínio real.

10. Faça um algoritmo para calcular e escrever a tabuada até 10 de um número lido.

11. Construa um algoritmo que, tendo como dados de entrada dois pontos quaisquer no plano, P(x1,y1) e P(x2,y2), escreva a distância entre eles. A fórmula que efetua tal cálculo é:

$$d = \sqrt{\left(\left(x2 - x1\right)^2 + \left(y2 - y1\right)^2\right)}$$

12. (*) Escreva um algoritmo que leia três números inteiros e positivos (A, B, C) e calcule a seguinte expressão:

$$D = \frac{R + S}{2} \qquad \text{sendo} \qquad R = \left(A + B\right)^2 \qquad S = \left(B + C\right)^2$$

13. Faça um algoritmo que leia a idade de uma pessoa expressa em anos, meses e dias e mostre-a expressa apenas em dias.

14. Ler um valor inteiro e escrever seu antecessor e seu sucessor.

15. Faça um algoritmo para calcular o valor total em metros quadrados necessários para revestir as paredes de um banheiro. *Área do quadrado= (largura * altura)^2 + (comprimento * altura)^2*

16. Faça um algoritmo que leia o número de votos brancos, nulos e válidos de uma eleição e escreva o percentual que cada um representa em relação ao total de votantes.

17. Uma loja de brinquedos oferece descontos de 15% sobre o preço de venda de todo o seu estoque. Escreva um algoritmo que leia o preço de um brinquedo antes da promoção e calcule quanto deve ser o novo preço.

18. Escreva um algoritmo que calcule e escreva o fatorial de 5.

19. O custo ao consumidor de um carro novo é a soma do custo de fábrica mais a percentagem do distribuir e o percentual de impostos. Faça um algoritmo que leia estas informações e forneça o valor a ser pago por um automóvel.

20. Escrever um algoritmo que lê o público possível em um estádio de futebol e o total de pessoas conforme o lugar ocupado no estádio (arquibancada ou cadeiras) em um determinado jogo de futebol. O algoritmo deverá informar a renda do jogo,

considerando valor de cada lugar nas arquibancadas igual a R$8.00 e valor de cada lugar nas cadeiras igual a R$12.00. Informe, também, o percentual de público presente ao jogo.

21. Uma vendedora de automóveis usados paga a seus funcionários vendedores um salário fixo por mês, mais uma comissão também fixa para cada carro vendido e mais 5% do valor das vendas por ele efetuados. Escrever um algoritmo que leia o número de carros por ele vendidos, o valor total de suas vendas, o salário fixo e o valor que recebe por carro vendido. Calcule e escreva o salário mensal do vendedor.

22. A empresa Vestebem resolve fazer uma promoção especial e conceder um desconto de 30% sobre o preço de venda de todo o seu estoque. Escreva um algoritmo que leia o preço de venda antes da promoção e calcule quanto deve ser o preço promocional.

23. Um empregado deseja saber se o cálculo do seu salário está correto verificando o seu contra-cheque. Escreva um algoritmo que leia o valor do salário bruto, o valor descontado para o INSS, o valor descontado para o imposto de renda. Calcule e escreva o percentual que foi utilizado para o cálculo do INSS e IR.

24. Uma loja vende bicicletas com um acréscimo de 50% sobre o seu preço de custo. Ela paga a cada vendedor 2 salários mínimos mensais, mais uma comissão de 15% sobre o preço de custo de cada bicicleta vendida, dividida igualmente entre eles. Escreva um algoritmo que leia o número de empregados da loja, o valor do salário mínimo, o preço de custo de cada bicicleta, o número de bicicletas vendidas. Calcule e escreva:
    - salário final de cada empregado;
    - lucro (líquido) da loja.

25. (*) Um mercado vende a dúzia de laranjas pelo dobro do preço de custo. Com a baixa em suas vendas o proprietário resolveu conceder um desconto a seus clientes. Escreva um algoritmo que leia o preço de custo da dúzia de laranjas (em R$), o percentual do desconto (por dúzia) fornecido e o número de laranjas (unidade) adquiridas pelo cliente. Calcule e imprima o valor a ser pago pelo cliente em R$. Após a escrita do resultado acima, deverá ser lido o valor que o cliente pagou ao mercado (em R$) e escrever o troco que deverá ser fornecido em R$.

4.5 EXERCÍCIOS RESOLVIDOS

Exercício 2:

Algoritmo	C
algoritmo salario_funcionario;	#include <stdio.h>

caracter nome;
inteiro horas, numeroFilhos;
real salario;
 	
inicio
leia(nome, numeroFilhos, horas);
salario= (horas*10 + 50*numeroFilhos);
salario= salario * 0.91; *//desconto INSS*
escreva(salario);
fim.
	void main(void)
	{
	char nome[15];
	int horas, numeroFilhos;
	float salario;
	printf("\n Digite o nome do funcionário \n");
	scanf("%s", &nome);
	printf("\n Digite o no. filhos com menos de 18 anos \n");
	scanf("%d", &numeroFilhos);
	printf("\n Digite o total de horas trabalhadas \n");
	scanf("%d", &horas);
	salario= (horas*10 + 50*numeroFilhos);
	salario= salario*0.91;
	printf("\n Salario a receber: %5.2f \n", salario);
	}

Exercício 4:

Algoritmo	C

algoritmo troca;
 	
inteiro a, b, aux; *//aux= variável auxiliar para guardar um valor*
 	
inicio
leia(a, b);
aux= a;
a= b;
b= aux;
escreva(a, b);
fim.
	#include <stdio.h>
	
	void main(void)
	{
	int a, b, aux;
	scanf("%i%i", &a, &b);
	aux= a;
	a= b;
	b= aux;
	printf("\n %i \n%i", a, b);
	}

Exercício 6:

Algoritmo	C

algoritmo temperatura;
 	
real grausCelsius, grausFahrenheit;
 	
inicio
leia(grausFahrenheit);
grausCelsius = (5 * (grausFahrenheit - 32))/9;
escreva(grausCelsius);
	#include <stdio.h>
	
	void main(void)
	{
	float grausCelsius, grausFahrenheit;
	printf("Digite a temperatura Farenheidt \n");
	scanf("%f", &grausFahrenheit);
	grausCelsius = (5 * (grausFahrenheit - 32))/9;

fim.	printf("\n Tempertura Celsius: %5.2f", grausCelsius); }

Exercício 12:

Algoritmo	C

algoritmo expressao;
 	
int a, b, c;
real d, r, s;
 	
inicio
leia(a, b, c);
r= pow((a + b), 2); *//pow=potência*
s= pow((b + c), 2);
d= (r + s)/2;
escreva(d);
fim.
	#include <stdio.h>
	#include <math.h>
	
	void main(void)
	{
	int a, b, c;
	float d, r, s;
	printf("Digite os valores de a, b e c \n");
	scanf("%d%d%d", &a, &b, &c);
	r= pow((a + b), 2);
	s= pow((b + c), 2);
	d= (r + s)/2;
	printf ("\n Divisão dos valores: %5.2f \n", r, d);
	}

Exercício 25:

Algoritmo	C

algoritmo laranjas;
 	
real preco_custoDuzia, desconto, valor, valorPago,
troco, precoVenda, preco_vendaUnidade;
inteiro numero;
 	
inicio
leia(preco_custoDuzia, desconto, numero);
 	
precoVenda= preco_custoDuzia * 2;
desconto= (precoVenda * desconto)/100;
preco_vendaUnidade=(precoVenda–
desconto)/12;
valor= preco_vendaUnidade * numero;
	#include <stdio.h>
	#include <math.h>
	
	void main(void)
	{
	float preco_custoDuzia, desconto, valor,
	valorPago, troco, precoVenda,
	preco_vendaUnidade;
	int numero;
	printf("Digite preço de custo da dúzia \n");
	scanf("%f”, &preco_custoDuzia);
	printf("Digite o percentual de desconto \n");
	scanf("%f”, &desconto);
	printf("Digite o número de laranjas \n");
	scanf("%d”, &numero);
	precoVenda= preco_custoDuzia * 2;
	desconto= (precoVenda * desconto)/100;
	preco_vendaUnidade= (precoVenda–
	desconto)/12;
	valor= preco_vendaUnidade * numero;

troco= valorPago – valor; escreva(valor, troco); fim.	troco= valorPago – valor; printf ("\n Valor: %.2f Troco:%.2f \n", valor,troco); }

Referências biliográficas:

Ziviani, Nivio. Projeto de Algoritmos em C e Java. 2ª ed. São Paulo: Thomson Pioneira, 2004

Feofiloff, Paulo. Algoritmos em Linguagem em C. 1ª ed. Rio de Janeiro: Campus, 2008

Fertig, Cristina; Medina, Marco. Algoritmos e Programação Teoria e Prática. 1ª ed. São Paulo: Novatec, 2005

Oliveira, Álvaro; Boratti, Isaias. Introdução a Programação de Algoritmos. 3ª ed. Florianópolis: Visual Books, 2007

Garcia, Guto; Lopes, Anita. Introdução à Programação: 500 Algoritmos Resolvidos. 1ª ed. Rio de Janeiro: Campus, 2002

Lux, Beatriz; Furtado, João Carlos. Apostila de Algoritmos e Programação. Departamento de Computação, UNISC, 2010

Parte V

5. COMANDOS CONDICIONAIS

Segundo Oliveira (2007 p. 45), “frequentemente, na construção de programas, vamos nos defrontar com problemas onde é necessário selecionar uma, entre duas ou mais situações possíveis”. Os comandos condicionais permitem que sejam determinadas as instruções a serem executadas em função do resultado de expressões lógicas (condições).

Por exemplo: **Se** media > 7 **então** “Aprovado” **senão** “Reprovado”

5.1 COMANDO CONDICIONAL SE

O comando SE determina que, caso o resultado lógico de *Condição* seja verdadeiro deverá ser executado Comando1, em caso contrário deverá ser executado Comando2.

Sintaxe:

SE (Condição)
    Comando1;
SENÃO
    Comando2;

Sintaxe em C:

IF (Condição)
    Comando1;
ELSE

Comando2;

Onde:

- SE e SENÃO são palavras reservadas do comando;
- Condição são expressões lógicas;
- Comando1 e Comando2 simbolizam um comando ou uma sequência de comandos.

Quando tivermos uma sequência de comandos para serem executados, tanto na parte SE quanto na parte SENÃO, devemos demarcá-los com um BLOCO, que fica delimitado pelos comandos Início ("**{**") e Fim ("**}**"). Exemplo:

```
SE (condição)
    {
            Comando1;
            Comando2;
            Comando3;
    }
SENÃO
    {
            Comando4;
            Comando5;
    }
```

Observe bem!!! Demarcando um BLOCO de comandos estaremos garantindo que aqueles comandos deverão ser executados conjuntamente, na sequência em que estão escritos. No caso de não demarcarmos os comandos por um bloco, o comando SE irá considerar apenas o primeiro de todos os comandos que lhe seguem como sendo dependente do seu resultado lógico.

O comando SE pode ser também utilizado na sua forma simples, onde não aparece o SENÃO e, ainda, pode ser composto por várias condições (como: &&, ||, >, <, >=, <=, ==, !=).

Exemplo:

SE (condição1 **&&** condição2)
Comando ou bloco;

5.2 SELEÇÃO ENCADEADA

Dentro de um comando SE podemos ter outros comandos de seleção, ou seja, condições encadeadas.

Sintaxe:

```
SE (condição1)
    SE (condição2)
        Comando ou Bloco1;
    SENÃO
        Comando ou Bloco2;
SENÃO
    SE (condição3)
        Comando ou Bloco3;
    SENÃO
        Comando ou Bloco4;
```

No caso de condição1 ser verdadeira, será testada condição2 que se for verdadeira levará à execução do Bloco1, se condição2, porém, for falsa, será executado o comando SENÃO que conduz a execução do bloco2. Se condição1 for falsa, levará direto para o SENÃO do primeiro SE onde será avaliada a condição3.

Podemos encadear quantos comandos SE forem necessários, a necessidade ou não de delimitar blocos dependerá do número de comandos dependentes.

Exemplo: Faça um programa cuja finalidade é calcular o salário final de um funcionário em função de abono concedido pela empresa. O programa deverá ler o salário base e considerar os seguintes limites para o abono:

- salário menor que 300.00 deverá receber um abono de 30%;
- salário entre os limites 300.00 (inclusive) e 1000 (exclusive) deverá receber abono de 20%;
- salário com valor igual ou superior a 1000.00 deverá receber abono de 10%.

```
Programa calculoSalario;

real sal_base, sal_final; // variáveis

início
        leia(sal_base);
        se(sal_base < 300)
                sal_final= sal_base * 1.30;
        senão
                se(sal_base >= 300   &&   sal_base < 1000)
                        sal_final= sal_base * 1.20;
                senão
                        sal_final= sal_base* 1.10;

        escreva("O salario com abono e: ", sal_final);
fim.
```

Exemplo em C:

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
 float sal_base, sal_final; // variáveis

 printf("\n Digite salario base: ");
 scanf("%f", &sal_base);

 if(sal_base < 300)
   sal_final= sal_base * 1.30;
 else
        if(sal_base >= 300   &&   sal_base < 1000)
          sal_final= sal_base * 1.20;
        else
          sal_final= sal_base* 1.10;

 printf("\n O salario com abono e: %.2f", sal_final);
```

```
 fflush( stdin ); // limpa buffer do teclado, funciona junto com entrada de dados
 getchar(); // parada da tela
}
```

No exemplo acima, não houve necessidade de demarcar blocos, já que para cada comando SE havia sempre um único comando na sua dependência.

5.3 COMANDO CONDICIONAL ESCOLHA

Uma especialização do comando SE é o comando "caso", também chamado de ESCOLHA. O ESCOLHA também é conhecido como comando de seleção de múltipla escolha, foi criado para facilitar o uso de SE encadeados. Geralmente este comando é utilizado em situações com várias opções, como em menus.

Sintaxe: **ESCOLHA** (seletor){

Caso 1: comando ou bloco;

Caso 2: comando ou bloco;

Caso n: comando ou bloco;

DEFAULT: Comando ou Bloco;

}

Sintaxe: **SWITCH** (seletor){

Case 1: comando ou bloco;

Case 2: comando ou bloco;

Case n: comando ou bloco;

DEFAULT: Comando ou Bloco;

}

Onde:

- ESCOLHA e CASO são palavras reservadas do comando;
- Seletor é uma variável ou expressão de tipo escalar (exceto real);
- Caso 1,... Caso n são os valores esperados para *seletor.*

O comando determina que em função do valor contido em *seletor* seja executado o comando ou bloco pertencente ao "caso" que se identifica com o seletor. "Caso" poderá ser um ou mais escalares ou um intervalo de escalares.

Exemplos:

ESCOLHA (N1)**{**

0 .. 10: comando ou bloco;

11 .. 20: comando ou bloco;

DEFAULT: Comando ou bloco;

}

ESCOLHA (Letra)**{**

'A','a': comando ou bloco;

'B', 'b': comando ou bloco;

'C', 'c': comando ou bloco;

DEFAULT: Comando ou bloco;

}

Exemplo: Faça um programa que leia dois número reais (N1 e N2) e um caractere. Conforme o valor lido para caractere o programa deverá calcular e informar:

+ a soma de N1 e N2;
- a diferença entre os dois valores;
* o produto de N1 e N2;
/ a divisão de N1 por N2.

```
Programa calculadora;

real n1 ,n2;
caracter operador;

leia(n1, n2, operador);
```

```
escolha(operador){
            caso '+':
                        escreva("a soma e: ", n1+n2);
            caso '-':
                        escreva("a diferenca e: ", n1-n2);
            caso '*':
                       escreva("\n a multiplicacao e: ", n1*n2);
           caso '/':
                        escreva("\n a divisao e: ", n1/n2);
            default:
                        escreva("operador invalido");
 }

fim.
```

Exemplo em C:

```
#include <stdio.h>
#include <conio.h>

int main(void)
{
 float n1 ,n2;
 char  operador;

 printf("\n Digite n1: ");
 scanf("%f", &n1);
 printf("\n Digite n2: ");
 scanf("%f", &n2);
 printf("\n Digite operador: ");
 operador= getche(); // ou scanf("%c", &operador);

 switch(operador){
            case '+':
                        printf("\n a soma e: %.2f", n1+n2);
                        break;
            case '-':
                        printf("\n a diferenca e: %.2f", n1-n2);
                       break;
            case '*':
                       printf("\n a multiplicacao e: %.2f", n1*n2);
                       break;
            case '/':
                        printf("\n a divisao e: %.2f", (float)n1/n2);
                       break;
            default:
                        printf("\n operador invalido");
 }

 fflush( stdin ); // limpa buffer do teclado, funciona junto com entrada de dados
 getchar(); // parada da tela
}
```

Observe no exemplo acima que os caracteres são sempre escritos entre apóstrofes e que eles são do mesmo tipo da variável operador. Observe ainda que o comando escolha permite que apenas um dentre os vários valores apresentados para a variável seletora seja escolhido, caso nenhum valor seja adequado, será executado o comando senão (DEFAULT).

Identicamente ao comando SE, também o comando ESCOLHA poderá ser utilizado sem o DEFAULT, por exemplo:

ESCOLHA (*seletor*)**{**

Caso 1: comando ou bloco;

Caso 2: comando ou bloco;

Caso n: comando ou bloco

}

5.4 EXERCÍCIOS PROPOSTOS

Exercícios identificados com (*) estão resolvidos ao final do capítulo.
Implemente todos algoritmos construídos.
Este material segue as referências bibliográficas indicadas ao final, principalmente de Lux e Furtado (2010).

1. (*) Faça um programa para calcular a mensalidade a ser paga pelos alunos de uma academia de ginástica que pratica os seguintes descontos:
   - mulheres com idade igual ou inferior a 15 anos terão desconto de 15%;
   - mulheres com idade superior a 15 anos terão desconto de 10%;
   - homens com idade inferior a 22 anos não terão desconto;
   - homens com idade igual ou superior a 22 anos terão desconto de 20%.

   Considerar a mensalidade básica como R$45.00

2. Faça um programa para calcular o resultado de um aluno após três provas, considerando a seguinte tabela:
   - média igual ou superior a 7 – aluno aprovado;
   - média igual ou superior a 5 e inferior a 7 – aluno em exame;
   - média inferior a 5 – aluno reprovado.

   A média do aluno é obtida através do cálculo de média ponderada, considerando peso=3 para a Nota 1, peso=2 para a Nota 2 e peso=5 para a Nota 3.

3. Faça um programa para determinar se um número inteiro lido é par.

4. (*) Faça um programa que lê um código (1, 2, 3 ou 4) e dois valores do tipo inteiro, calcula e fornece:
   A adição dos números para código =1.
   A subtração dos números para código =2.
   A multiplicação dos números para código =3.
   A divisão dos números para código = 4.

5. Faça um programa que calcula as raízes de uma equação do 2º Grau, considerando que não é possível a divisão por zero (informar caso ocorra) e considerando também inviável o cálculo de raízes imaginárias (informar caso ocorra).

6. Faça um programa que lê dois valores numéricos e informa se são múltiplos entre si.

7. Faça um programa para ler três valores numéricos e escrever o maior entre eles. Considere que não haverá valores iguais.

8. (*) Escrever um programa que leia as medidas dos lados de um triângulo e escreva se ele é equilátero, isóscele ou escaleno.
   - Triângulo equilátero: possui os 3 lados iguais;
   - Triângulo isóscele: possui os 2 lados iguais;
   - Triângulo escaleno: possui os 3 lados diferentes.

9. (*) Faça um programa para ler três valores inteiros e escrevê-los em ordem crescente (considere que serão informados apenas valores diferentes).

10. Escreva um programa que leia um valor referente a dinheiro (por exemplo R$540.00) e calcula qual o menor número possível de notas de 50.00, 10.00, 5.00 e 1.00 reais em que o valor lido pode ser decomposto. O programa deverá fornecer o valor lido e a relação de notas necessárias.

11. (*) Escreva um programa que lê a hora de início e fim de um jogo (considere apenas horas inteiras) e calcula a duração do jogo, sabendo-se que o tempo máximo de duração do jogo é inferior a 24 horas e que o jogo pode iniciar em um dia e terminar no seguinte.

12. Faça um programa que lê 2 valores, o primeiro servindo de indicador de operação e o segundo correspondendo ao raio de uma circunferência. Se o valor for 1, calcular e imprimir a área da circunferência. Se o valor lido for 2, calcular e imprimir o perímetro da circunferência, e se o valor lido for diferente destes dois valores, imprimir uma mensagem dizendo que o indicador de operação foi mal fornecido.

13. Faça um programa que lê quatro valores, I, A, B e C onde I é um número inteiro e positivo e A, B e C são quaisquer valores reais. O programa deve escrever os valores lidos e:
    - se I=1 escrever os três valores A, B e C em ordem crescente;
    - se I=2 escrever os três valores A, B e C em ordem decrescente;
    - se I=3 escrever os três valores A, B e C de forma que o maior valor fique entre os outros dois;
    - se I não for um dos três valores acima, dar uma mensagem indicando isto.

14. Uma loja fornece 5% de desconto para funcionários e 10% de desconto para clientes especiais. Faça um programa que calcule o valor total a ser pago por uma pessoa. O programa deverá ler o valor total da compra efetuada e um código que identifique o tipo de cliente.

15. A partir do preço a vista de um determinado produto, calcular o preço total a pagar e o valor da prestação mensal, referentes ao pagamento parcelado. Se o pagamento for parcelado em três vezes deverá ser dado um acréscimo de 10% no total a pagar. Se for parcelado em 5 vezes, o acréscimo é de 20%.

16. O cardápio de uma lanchonete é o seguinte:

Especificação	Preço unitário
100 Cachorro quente	R$3,50
101 Bauru simples	R$4,00
102 Bauru c/ovo	R$4,50
103 Hamburger	R$4,00
104 Cheeseburger	R$3,00

    Escrever um programa que leia o código do item pedido, a quantidade e calcule o valor a ser pago por aquele lanche. Considere que a cada execução somente será calculado um item.

17. Uma empresa concederá um aumento de salário aos seus funcionários, variável de acordo com o cargo, conforme a tabela abaixo. Faça um programa que leia o salário e o cargo de um funcionário e calcule o novo salário. Se o cargo do funcionário não estiver na tabela, ele deverá, então, receber 40% de aumento. Mostre o salário antigo, o novo salário e a diferença.

Código	Cargo	Percentual
101	Gerente	10%
102	Engenheiro	20%
103	Técnico	30%

18. Ler um valor e escrever se é positivo, negativo ou zero.

19. As maçãs custam R$0,50 cada se forem compradas menos do que uma dúzia, recebem 5% de desconto se for comprada uma dúzia e 10% de desconto se forem compradas duas dúzias ou mais. Já as peras, custam R$0.65 cada se for comprada menos do que uma dúzia, recebem 4% de desconto se for comprada uma dúzia e 8% de desconto se forem compradas três dúzias ou mais. Escreva um programa que leia a quantidade comprada de cada fruta, calcule e escreva o custo total da compra.

20. Faça um programa para ler o nome e o número de votos de quatro candidatos em uma eleição. Ao final deverá ser escrito o nome do vencedor.

21. Um posto está vendendo combustível com a seguinte tabela de descontos:
    Álcool: até 30 litros, desconto de 3% por litro; acima de 30 e até 40 litros 5% de desconto; acima de 40 litros, desconto de 7% por litro.

    Gasolina: até 20 litros, desconto de 4% por litro; acima de 20 litros e até 30, desconto de 6% por litro; acima de 30 litros desconto de 8% por litro.

    Escreva um programa que leia o número de litros vendidos, o tipo de combustível (codificado da seguinte forma: A – álcool, G - gasolina), calcule e imprima o valor a ser pago pelo cliente sabendo que o preço da gasolina é de R$ 2,63 o litro e do álcool R$ 1,56.

22. O departamento que controla o índice de poluição do meio ambiente mantém controle sobre 3 grupos de indústrias que são altamente poluidoras. Faça um programa que, após ler o tipo da empresa e o nível de poluição do ar, emite uma notificação adequada conforme os índices da tabela abaixo:

Grupo	Nível aceitável	Ajustar produção	Paralisar produção
1	<0.35	0.35 - 0.45	Acima de 0.45
2	<0.3	0.30 - 0.4	Acima de 0.4
3	<025	0.25 – 0.35	Acima de 0.35

23. Escrever um programa que leia a idade de dois homens e duas mulheres (considere que a idade dos homens será sempre diferente, assim como das mulheres). Calcule e escreva a soma das idades do homem mais velho com a mulher mais nova, e o produto das idades do homem mais novo com a mulher mais velha.

24. Faça um programa que leia uma data (dia, mês e ano), verifique e informe se ela é válida.

25. (*) A jornada de trabalho semanal é de 40 horas. O funcionário que trabalhar mais que 40 horas receberá hora extra, cujo cálculo é o valor da hora regular com um acréscimo de 50%. Escreva um programa que leia o número de horas trabalhadas

em um mês, o salário por hora e escreva o salário total do funcionário (considere que o mês possui 4 semanas exatas).

5.5 EXERCÍCIOS RESOLVIDOS

Exercício 1:

Programa	C

programa mensalidade;
 	
const mens=45;
inteiro idade;
caractere sexo;
real total;
inicio
total= mens;
leia(sexo, idade);
 	
se sexo = 'f' então
se idade <= 15 então
total= mens*0.85
senão
tota= mens*0.9
senão
se idade >= 22 então
total= mens*0.8;
escreva(total);
fim.
	#include <stdio.h>
	#include <stdlib.h>
	#define mens 45
	void main(void) {
	int idade;
	char sexo;
	float total;
	total=mens;
	printf("Digite sexo e idade \n");
	scanf("%c %d", &sexo, &idade);
	if(sexo=='F' || sexo== 'f')
	if(idade <= 15)
	total=mens*0.85;
	else
	total= mens*0.90;
	else
	if(idade >=22)
	total= mens*0.8;
	printf("\n Mensalidade: %.2f \n", total);
	}

Exercício 4:

Programa	C

programa operacoes;
 	
inteiro n1, n2, codigo;
real r;
 	
inicio
leia(codigo, n1, n2);
 	
escolha código
 	
1: r= n1+n2;
 	
2:r= n1-n2;
 	
3:r= n1*n2;
	#include <stdio.h>
	#include <stdlib.h>
	void main(void){
	int n1, n2, codigo;
	float r;
	printf("1- adicao 2-subtracao 3-multiplicacao 4-divisao\n");
	printf("digite a opcao\n");
	scanf("%i", &codigo);
	printf("digite os dois valores inteiros\n");
	scanf("%i%i", &n1, &n2);
	switch (codigo)
	{
	case 1: r= n1+n2;
	break;
	case 2: r= n1-n2;
	break;
	case 3: r= n1*n2;
	break;
	case 4: r= n1/n2;

4:r= n1/n2
senão
 	
escreva('código inválido');
fim;
escreva('resultado: ',r);
fim.
	break;
	default: printf("\n opcao invalida \n");
	}
	printf("resulatdo: %.2f \n", r);
	}

Exercício 8:

Programa	C

programa triangulo;
 	
inteiro a, b, c;
inicio
 	
leia(a, b, c);
se (a = b) e (b = c) então
escreva('Triângulo equilátero!')
senão
se (a <> b) e (b <> c) e (a <> c) então
escreva('Triângulo escaleno!')
senão
escreva('Triângulo isósceles!')
fim.
	#include <stdio.h>
	#include <stdlib.h>
	void main(void){
	int a, b, c;
	printf("Valores: \n");
	scanf("%d %d %d", &a1, &b, &c);
	if((a == b) && (b == c))
	printf("\n Triângulo equilátero");
	else
	if((a <> b) && (b <> c) && (a <> c))
	printf("\n Triângulo escaleno");
	else
	printf("\n Triângulo isósceles");
	}

Exercício 9:

Programa	C

programa ordemCrescente;
 	
inteiro a, b, c;
 	
inicio
 	
leia(a, b, c);
se a > b então
se b > c então
escreva('Ordem: ', c, b, a)
senão
se a > c então
escreva('Ordem: ', b, c, a)
senão
escreva('Ordem: ', b, a, c)
	#include <stdio.h>
	#include <stdlib.h>
	void main(void){
	int a, b, c;
	printf("Valores: \n");
	scanf("%d %d %d", &a1, &b, &c);
	if(a > b)
	if(b > c)
	printf("\n Ordem: %d %d %d", c, b, a);
	else
	if(a > c)
	printf("\n Ordem: %d %d %d" ,b, c, a);
	else
	printf("\n Ordem: %d %d %d", b, a, c);
	else

senão
  se b < c então
  escreva('Ordem: ', a, b, c)
  senão
  se a < c então
  escreva('Ordem: ', a, c, b)
  senão
  escreva('Ordem: ', c, a, b);
fim.
	if(b < c)
	  printf("\n Ordem: %d %d %d", a, b, c);
	else
	  if(a < c)
	  printf("\n Ordem: %d %d %d", a, c, b);
	  else
	  printf("\n Ordem: %d %d %d", c, a, b);
	}

Exercício 11:

Programa	C

programa jogo;
 	
inteiro hi, hf, dh;
inicio
leia(hi, hf);
se hi < hf então
dh= hf - hi
senão
dh= 24 – hi + hf;
escreva(dh,' horas');
fim.
	#include <stdio.h>
	void main(void){
	int hi, hf, dh;
	printf("Digite hora inicial e final (meia noite=0) \n");
	scanf("%i %i", &hi, &hf);
	if (hi < hf)
	dh= hf - hi;
	else
	dh= 24 – hi + hf;
	printf("%i horas \n", dh);
	}

Exercício 25:

Programa	C

programa salario;
 	
real horas_trabalhadas, salario_hora, salario_total, horas_extras;
inicio
leia(horas_trabalhadas, salario_hora);
 	
se horas_trabalhadas <= 160 então
salario_total=salario_hora * horas_trabalhadas;
senão
inicio
horas_extras= horas_trabalhadas - 160;
salario_total= horas_extras * salario_hora * acrescimo + 160 * salario_hora;
fim;
escreva("Salário:", salario_total);
	#include <stdio.h>
	void main(void){
	float horas_trabalhadas, salario_hora, salario_total, horas_extras;
	
	printf("Digite as horas trabalhadas e o salário hora \n");
	scanf("%f %f", &horas_trabalhadas, &salario_hora);
	
	if(horas_trabalhadas <= 160)
	salario_total=salario_hora * horas_trabalhadas;
	else
	{
	horas_extras= horas_trabalhadas - 160;
	salario_total= horas_extras * salario_hora * acrescimo + 160 * salario_hora;
	}
	printf("Salário: %.2f", salario_total);
	}

fim.	

Referências biliográficas:

Ziviani, Nivio. Projeto de Programas em C e Java. 2ª ed. São Paulo: Thomson Pioneira, 2004

Feofiloff, Paulo. Programas em Linguagem em C. 1ª ed. Rio de Janeiro: Campus, 2008

Fertig, Cristina; Medina, Marco. Programas e Programação Teoria e Prática. 1ª ed. São Paulo: Novatec, 2005

Oliveira, Álvaro; Boratti, Isaias. Introdução a Programação de Programas. 3ª ed. Florianópolis: Visual Books, 2007

Garcia, Guto; Lopes, Anita. Introdução à Programação: 500 Programas Resolvidos. 1ª ed. Rio de Janeiro: Campus, 2002

Lux, Beatriz; Furtado, João Carlos. Apostila de Programas e Programação. Departamento de Computação, UNISC, 2010

Parte VI

6. COMANDOS DE REPETIÇÃO

Um de nossos exemplos anteriores foi o cálculo da média de um aluno, onde foi fornecido o resultado final (aprovado, reprovado) em função desta média. Normalmente este cálculo é necessário não apenas para um aluno, mas para toda uma turma, por exemplo, 45 alunos.

Neste caso devemos declarar 45 vezes mais variáveis?! Considerando um conjunto básico de variáveis para cada aluno (nota 1 , nota 2, média, etc.)... imagine o tamanho que este programa terá?
Felizmente isto pode ser feito de forma muito mais ágil e automática (e já era de esperar porque na programação buscamos concisão, agilidade, menor tempo e menor esforço!).

Existem comandos apropriados para determinar a repetição de um bloco de instruções. Tais comandos são: comando **PARA**, comando **ENQUANTO** e comando **REPITA**. Eles se diferenciam tanto pelo uso, ou não, uma expressão lógica para testar a repetição, quanto por testarem a repetição antes ou depois da execução do bloco de instruções.

Vamos estudá-los?

6.1 COMANDO PARA

O comando **PARA** executa a repetição de um bloco de comandos por um número fixo de vezes e o teste para determinar a repetição é executado antes da execução do bloco de comandos.

Sintaxe:

PARA(controle= valor_inicial; controle <= valor_final; controle++**)**

Comando ou bloco de comandos;

Sintaxe em C:

FOR(controle= valor_inicial; controle <= valor_final; controle++**)**

Comando ou bloco de comandos;

Onde:

- **PARA** É palavra reservada do comando.
- **controle** é uma variável ou expressão pertencente a tipo escalar (exceto real).
- **valor_inicial** e **valor_final** são do mesmo tipo de **controle** e delimitam a repetição do comando ou bloco de comandos.

Exemplo: Faça um algoritmo para calcular o resultado final de 45 alunos de uma turma, considerando que cada aluno tem duas notas e que será considerado aprovado aquele aluno cuja média aritmética das notas for igual ou superior a 7. Considere como dados de entrada o nome do aluno e suas notas. Deverá ser fornecido o nome do aluno, sua média e resultado.

```
Algoritmo calculoMedia;
real nota1, nota2, media;
caracter nome;
inteiro i;

início
```

```
para(i= 0; i < 45; i++)
início
   leia(nome, nota1, nota2);

   media= (nota1 + nota2)/2; // cálculo da média
   se(media >= 7)
     escreva(nome, "esta aprovado com media ", media);
   senão
     printf(nome, "foi reprovado, sua media e ", media);
fim;
fim.
```

Exemplo em C:

```
int main(void)
{
 float nota1, nota2, media;
 char nome[30]; // nome é uma variável string que pode armazenar 30 caracteres
 int i;

 for(i= 0; i < 45; i++)
 {
   printf("\n Informe nome: ");
   gets(nome); // comando para a leitura de string
   printf("\n Informe nota 1: ");
   scanf("%f", &nota1);
   printf("\n Informe nota 2: ");
   scanf("%f", &nota2);

   media= (nota1 + nota2)/2; // media
   if(media >= 7)
     printf("\n %s esta aprovado com media %.2f ", nome, media);
   else
     printf("\n %s foi reprovado, sua media e %.2f ", nome, media);
 }
 fflush( stdin );
 getchar();
}
```

Vamos analisar o exemplo...

A variável de controle do comando **PARA**, no exemplo representada por "i", inicia com o valor 1 e "i" terá seu valor incrementado em uma unidade até o valor limite estabelecido em 45. A cada novo valor desta variável de controle, será executado o bloco de instruções que lhe segue, onde são lidos os dados para cada aluno e determinado o seu resultado. A repetição dos comandos será executada, portanto, até

que a variável “i” atinja o valor 45, que ultrapassa o limite superior. O acréscimo dado a variável de controle do comando **PARA** é denominado de passo.

Observe: Não deverá existir nenhuma instrução que altere o valor da variável de controle do comando **PARA**. A variável de controle deve ser incrementada e controlada pelo comando **PARA** exclusivamente, de forma automática, como se fosse um contador.

Contador??? Vejamos então o que é um contador.

6.2 CONTADORES E ACUMULADORES

6.2.1 CONTADORES

Muitas vezes para resolução de uma tarefa necessitamos que uma variável seja encarregada de armazenar o número de vezes que determinada condição foi satisfeita, por exemplo, na resolução do seguinte problema: faça um algoritmo para ler 20 valores inteiros e informar quantos destes são pares.

A resolução do algoritmo vai envolver uma repetição por 20 vezes de:

- Leitura de um valor inteiro.
- Testar o valor para verificar se é par e caso seja, acrescentar uma unidade à variável encarregada de contar os valores pares.

Assim, uma variável **CONTADOR** é uma variável que recebe um valor inicial (normalmente zero) e é incrementada posteriormente de algum valor constante (geralmente em unidade).

```
Algoritmo quantidadePares;
inteiro numero, i, contador= 0; // declara e inicializa a variável contador
```

```
início
 para(i= 0; i < 20; i++)
 início
      leia(numero);
      se(numero % 2 == 0)
         contador= contador + 1; // ou contador++;
fim;
escreva("Foram lidos ", contador, " valores pares");
fim.
```

Exemplo em C:

```
int main(void)
{
 int numero, i, contador= 0; // declara e inicializa a variável contador

 for(i= 0; i < 20; i++){
      printf("\n Informe numero: ");
      scanf("%d", &numero);

      if(numero % 2 == 0)
         contador= contador + 1; // ou contador++;
}
printf("\n Foram lidos ", contador, " valores pares");
fflush( stdin ); // limpa buffer do teclado, funciona junto com entrada de dados
getchar(); // parada da tela
} // fim do programa principal
```

Observe a necessidade de atribuir zero inicialmente ao contador, para garantir que a cada nova execução do programa não permaneça valores anteriores, ocasionando um resultado errado.

6.2.2 ACUMULADORES

Outra situação frequente em algoritmos é a necessidade de realizar o somatório de determinados valores. Por exemplo, para calcular a média das notas finais de uma turma de 45 alunos, é necessário ir acumulando a média de cada aluno e após, dividir a

soma das médias pelo número de alunos. A variável encarregada de armazenar este somatório é conhecida em programação como acumulador.

Assim, uma variável utilizada como **ACUMULADOR** é uma variável que recebe um valor inicial (geralmente zero) é e incrementada posteriormente de um valor variável.

```
Algoritmo mediaTurma;
real nota1, nota2, media, acumulador= 0;
caracter nome;
inteiro i;

início
 para(i= 0; i < 45; i++)
 início
    leia(nome, nota1, nota2);

    media= (nota1 + nota2)/2; // cálculo da média
    se(media >= 7)
      escreva(nome, "esta aprovado com media ", media);
    senão
      escreva(nome, "foi reprovado, sua media e ", media);
    acumulador= acumulador + media;
 fim;
 escreva("A media da turma e ", acumulador/45);
fim.
```

Exemplo em C:

```
int main(void)
{
 float nota1, nota2, media, acumulador= 0;
 char nome[30]; // nome é uma variável string que pode armazenar 30 caracteres
 int i;

 for(i= 1; i <= 45; i++)
 {
    printf("\n Informe nome: ");
    gets(nome); // comando para a leitura de string
    printf("\n Informe nota 1: ");
    scanf("%f", &nota1);
    printf("\n Informe nota 2: ");
    scanf("%f", &nota2);

    media= (nota1 + nota2)/2; // media
    if(media >= 7)
      printf("\n %s esta aprovado com media %.2f ", nome, media);
```

```
    else
      printf("\n %s foi reprovado, sua media e %.2f ", nome, media);
    acumulador= acumulador + media;
  }
  printf("\n A media da turma e %.2f ", (float)acumulador/45);
  fflush( stdin ); // limpa buffer do teclado, funciona junto com entrada de dados
  getchar(); // parada da tela
 } // fim do programa principal
```

6.3 COMANDO ENQUANTO

O comando **ENQUANTO** serve para deve ser executar um comando ou um bloco de comandos repetidamente, enquanto uma determinada condição for verdadeira.

O fato de utilizar uma expressão lógica para controlar a repetição amplia bastante a utilização do comando **ENQUANTO** se comparado ao comando **PARA**, pois podemos utilizá-lo para um número previamente conhecido de repetições e também para a situação em que o número de repetições a executar é desconhecido.

Sintaxe:

ENQUANTO (condição)

Comando ou bloco de comando

Sintaxe em C:

WHILE (condição)

Comando ou bloco de comandos;

Onde:

- **ENQUANTO** é palavra reservada do comando.
- Condição é uma expressão lógica.

Exemplo 1: Faça um algoritmo para calcular o resultado final de 45 alunos de uma turma, considerando que cada aluno tem duas notas e que será considerado aprovado aquele aluno cuja média aritmética das notas for igual ou superior a 7. Considere com dados de entrada o nome do aluno e suas notas. Deverá ser fornecido o nome do aluno, sua média e resultado.

```
Algoritmo mediaAluno;
real nota1, nota2, media;
caracter nome;
inteiro i= 0;

início
 enquanto(i < 45)
 início
    leia(nome, nota1, nota2);

    media= (nota1 + nota2)/2; // cálculo da média
    se(media >= 7)
      escreva(nome, "esta aprovado com media ", media);
    senão
      escreva(nome, "foi reprovado, sua media e ", media);
    i= i + 1;
 fim;
fim.
```

Exemplo em C:

```
int main(void)
{
 float nota1, nota2, media;
 char nome[30]; // nome é uma variável string que pode armazenar 30 caracteres
 int i= 0;

 while(i < 45){
    printf("\n Informe nome: ");
    gets(nome); // comando para a leitura de string
    printf("\n Informe nota 1: ");
    scanf("%f", &nota1);
    printf("\n Informe nota 2: ");
    scanf("%f", &nota2);

    media= (nota1 + nota2)/2; // media
```

```
    if(media >= 7)
      printf("\n %s esta aprovado com media %.2f ", nome, media);
    else
      printf("\n %s foi reprovado, sua media e %.2f ", nome, media);
    i++; // ou i= i + 1;
}
fflush( stdin );
getchar();
}
```

Como o comando enquanto controla a repetição através de uma condição e temos um número conhecido (45) de repetições a executar, necessitamos de uma variável para fazer o papel de contador de repetições e sobre o seu valor estabelecermos o teste. Foi escolhida para contador a variável "i", que inicia com um valor igual a zero e é incrementada após cada repetição em uma unidade.

Desafio:

Pense o que seria necessário alterar no algoritmo acima

se a variável contador iniciasse com o valor 1?

Vamos ver outro exemplo!

Exemplo 2: Faça um algoritmo que deverá calcular e informar a média de "n" números inteiros lidos, encerrando a leitura destes números com a entrada de um valor negativo.

```
Algoritmo mediaInteiros;
real media, acum= 0;
inteiro valor, cont= 0;

início
 leia(valor);

 enquanto(valor >= 0)
 início
      acum= acum + valor;
       cont= cont + 1;
      leia(valor);
  fim;
se(cont > 0)
início
   media= acum / cont;
```

```
   escreva("A media entre os valores lidos e: ", media);
 fim;
fim.
```

Exemplo em C:

```
int main(void)
{
 float media, acum= 0;
 int valor, cont= 0;

 printf("\n Valor: ");
 scanf("%d", &valor);

 while(valor >= 0){
      acum= acum + valor; // ou acum+= valor;
       cont++;
      printf("\n Valor: ");
      scanf("%d", &valor);
 }
 if(cont > 0){
   media= (float)acum / cont;
   printf("\n A media entre os valores lidos e: %.2f", media);
 }
 fflush( stdin );
 getchar();
}
```

Observe: para que seja possível a realização do primeiro teste (condição) no comando **ENQUANTO** é necessária a leitura prévia de um valor e a variável valor deverá ser lida novamente antes do final do bloco de repetição, para permitir novo teste no comando **ENQUANTO**.

6.4 COMANDO REPITA

O comando **REPITA** serve para estabelecermos que um ou vários comandos sejam executado repetidamente **ATÉ** enquanto determinada condição seja verdadeira.

O comando **REPITA** diferencia-se dos dois comandos de repetição anteriores por realizar o teste da condição de repetição após a execução de todos os comandos.

Sintaxe:

REPITA{

Comando(s);

}ATÉ (condição);

Sintaxe em C:

DO{

Comando(s);

}WHILE (condição);

Onde:

- **REPITA .. ATÉ** são palavras reservadas do comando.
- Condição é uma expressão lógica.

Vejamos os mesmos exemplos resolvidos com o comando **ENQUANTO** só que agora utilizaremos o **REPITA .. ATÉ**.

Exemplo 1: Faça um programa para calcular o resultado final de 45 alunos de uma turma, considerando que cada aluno tem duas notas e que será considerado aprovado aquele aluno cuja média aritmética das notas for igual ou superior a 7. Considere com dados de entrada o nome do aluno e suas notas. Deverá ser fornecido o nome do aluno, sua média e resultado.

```
Algoritmo mediaAluno;
real nota1, nota2, media;
caracter nome;
inteiro i= 0;
início
 repita
   leia(nome, nota1, nota2);

   media= (nota1 + nota2)/2; // cálculo da média
   se(media >= 7)
     escreva(nome, "esta aprovado com media ", media);
   senão
     escreva(nome, "foi reprovado, sua media e ", media);
   i= i + 1;
 até(i < 45);
fim.
```

Exemplo em C:

```
int main(void)
{
 float nota1, nota2, media;
 char nome[30];
 int i= 0;

  do{
    printf("\n Informe nome: ");
    fflush(stdin);
    gets(nome); // comando para a leitura de string
    printf("\n Informe nota 1: ");
    scanf("%f", &nota1);
    printf("\n Informe nota 2: ");
    scanf("%f", &nota2);

    media= (nota1 + nota2)/2; // media
    if(media >= 7)
      printf("\n %s esta aprovado com media %.2f ", nome, media);
    else
      printf("\n %s foi reprovado, sua media e %.2f ", nome, media);
    i++; // ou i= i + 1;
 }while(i < 45);

 fflush( stdin );
 getchar();
}
```

Como o comando **REPITA .. ATÉ** também controla a repetição através de uma condição e temos um número conhecido (45) de repetições a executar, necessitamos de uma variável para servir como contador de repetições e sobre o seu valor estabelecermos o teste. Foi escolhida para contador a variável "i", que inicia com um valor igual a um e é incrementada após cada repetição em uma unidade.

Exemplo 2: Faça um algoritmo que deverá calcular e informar a média de "n" números inteiros lidos, encerrando a leitura destes números com a entrada de um valor negativo.

```
Algoritmo mediaInteiros;
real media, acum= 0;
inteiro valor, cont= 0;

repita
      leia(valor);
      se(valor >= 0)
      início
            acum= acum + valor;
            cont= cont + 1;
      fim;
até(valor >= 0);

se(cont > 0)
início
  media= acum / cont;
  escreva("A media entre os valores lidos e: ", media);
fim;
fim.
```

Exemplo em C:

```
int main(void)
{
 float media, acum= 0;
 int valor, cont= 0;

 do{
      printf("\n Valor: ");
      scanf("%d", &valor);
```

```
            if(valor >= 0){
                acum= acum + valor; // ou acum+= valor;
                cont++;
            }
}while(valor >= 0);

if(cont > 0){
  media= (float)acum / cont;
  printf("\n A media entre os valores lidos e: %.2f", media);
}
fflush( stdin );
getchar();
}
```

Observe: na resolução do segundo exemplo foi necessário introduzir mais um teste para impedir de acrescentar ao acumulador um valor negativo e para que o mesmo não fosse contado como valor válido. Isto mostra que, devido a sua característica de testar a condição depois de executar os comandos da repetição, nem sempre o comando **REPITA .. ATÉ** uma boa escolha. Neste exemplo o comando **ENQUANTO** proporciona uma solução melhor construída.

Mas observe cada problema, há momentos que o comando **REPITA .. ATÉ** é mais apropriado. Por exemplo: quando se deseja testar se uma entrada de dados é válida. Veja um trecho de algoritmo a seguir. Ele mostra a validação de uma entrada de dados, onde só é possível seguir o processamento do algoritmo se o usuário digitar uma idade válida entre 0 e 150.

```
...
    REPITA
            leia(idade);
    ATÉ (idade >= 0 && idade <= 150);
```

6.5 EXERCÍCIOS PROPOSTOS

Exercícios identificados com (*) estão resolvidos ao final do capítulo.
Implemente todos algoritmos construídos.
Este material segue as referências bibliográficas indicadas ao final, principalmente de Lux e Furtado (2010).

1. (*) Faça um programa que leia cinco valores inteiros, calcule e escreva:

a) a soma dos números pares;
b) o menor entre os valores lidos.

2. Faça um programa que calcule o fatorial de um número lido (n) por repetidas vezes. Encerra com n menor que zero.

3. Faça um programa que apresente a série de Fibonacci, até o décimo quinto termo. A série de Fibonacci é formada pela sequência:
1, 1, 2, 3, 5, 8, 13, 21, 34 (esta série se caracteriza por iniciar com dois valores determinados que formam o primeiro e segundo termo (t1=1 e t2=1) e a partir destes cada termo é obtido através da soma dos seus dois termos anteriores (t3 = t1+t2).

4. Faça um programa que lê um número e informa se ele , é primo.
(Os números primos são divisíveis por um e por si mesmos, somente)

5. Duas tartarugas possuem 3,1 e 4,5 cm respectivamente. A menor tartaruga cresce 0,21 cm ao ano e a maior 0,16 cm ao ano. Faça um programa que calcule o número de anos necessários para que a tartaruga menor fique maior que a maior atualmente.

6. Faça um programa que lê um número indeterminado de valores, todos inteiros, um de cada vez, finalizando a leitura com o número zero e forneça como saída:
a) a soma dos números que estão no intervalo 10(inclui) a 20(inclui);
b) a média dos números que estão no intervalo de 1(inclui) a 10(exclui);
c) o menor entre os valores lidos.

7. Faça um programa que leia um número inteiro repetidas vezes, paralisando a leitura com número <= zero e escreva, para cada valor lido, a soma de todos os números menores e não sub-múltiplos do número lido.

8. (*) Faça um programa que lê 2 notas de alunos e calcula suas médias individuais e a média geral da turma. O programa finaliza com 5 alunos ou quando a média geral for menor ou igual a cinco.

9. Faça um programa que lê números inteiros e finaliza quando a soma destes atinge um valor superior a 30.

10. Faça um programa para ler as notas (N1 e N2) de alunos (para um total de 5 alunos) e calcular suas médias individuais, escrevendo, para cada um:
"aprovado" - se media superior ou igual a sete, "reprovado" - se inferior a sete. O programa também deverá informar a média geral da turma e o percentual de alunos aprovados.

11. Faça um programa que calcule a média aritmética de todos os números pares que forem fornecidos pelo usuário. O valor de finalização deve ser a entrada do

número zero. O usuário pode entrar com números ímpares, mas estes não serão acumulados.

12. Faça um programa para apresentar os quadrados dos números inteiros múltiplos de 5 no intervalo de 15 a 100.

13. Construa um programa que permita fazer um levantamento do estoque de vinhos de uma adega, tendo como dados de entrada tipos de vinho (branco, tinto e rosê). Especifique a porcentagem de cada tipo sobre o total geral de vinhos. Considere que a quantidade de vinhos é desconhecida (usar como finalizador da leitura a constante 'fim').

14. (*) Faça um programa que escreve a tabuada até 10 dos dez primeiros números inteiros positivos permitindo a repetição do cálculo para novo número após ler 'S' ou 'N 'como resposta para: Escrever nova tabuada(S/N)?.

15 Faça um programa que calcule o imposto de renda de um grupo de contribuintes considerando que os dados de cada contribuinte, número do CPF, numero de dependentes e renda mensal são valores fornecidos pelo usuário. Para cada contribuinte será feito um desconto de 5% de salário mínimo por dependente.
Os valores da alíquota para calculo do imposto são:

até 2 sal min (exclusive)	isento
2 .. 3 (inclusive,inclusive)	5%
3 .. 5 (exclusive, inclusive)	10%
5 .. 7 (exclusive,inclusive)	15%
acima de 7	20%

O último valor (para encerrar o programa), que não será considerado, terá CPF igual a zero. Deve ser lido o valor atual do salário mínimo.

16. Construa um programa que, dado um conjunto de valores inteiros e positivos determine qual o menor e o maior valor do conjunto. O final do conjunto de valores é conhecido através do 0 (zero), que não deve ser considerado.

17. A conversão de graus Fahrenheit para centígrados é obtida pela fórmula C = 5/9(f-32). Escreva um programa que calcule e escreva uma tabela de graus centígrados em função de graus Fahrenheit que varie de 50 a 150, de 1 em 1.

18. Elabore um programa que calcule o valor de H, sendo que ele é determinado pela série:

$$H = 1/1 + 3/2 + 5/3 + 7/4 + ... + 99/50$$

19. Elabore um programa que efetue a soma de todos os números ímpares que são múltiplos de três e que se encontram no conjunto dos números de 1 até 500.

20. Considerando a altura dos N alunos de uma turma faça um programa para calcular a média das alturas e determinar a quantidade de alunos com altura >1.60 e <1.75 m.
Encerrar leitura com altura <=0.

21. Um banco utiliza recursos computacionais para efetuar suas transações e uma das necessidades bancárias é saber o total de dinheiro que o banco girou durante o dia. Para tanto a cada débito ou crédito que o caixa executa é digitado o respectivo valor sendo que o débito é sempre precedido do sinal negativo. Faça um programa para calcular o total de créditos e débitos e o total global (movimento do dia) em valor absoluto. Encerrar entradas com zero.

22. Faça um programa que escreva os 30 primeiros termos da seguinte série: 0, 1, 2, 5, 12, 29.

23. Foi realizada uma pesquisa em um edifício para otimizar a utilização dos elevadores, para um número indeterminado de usuários. O edifício possui quatro elevadores (a, b, c, d) que são usados livremente. Os usuários responderam as seguintes questões:

- qual o elevador utilizado (a, b, c, d);
- qual o turno, seguindo os códigos: manhã, tarde e noite (m, t, n)

O programa deve fornecer o elevador mais utilizado e qual o turno em que os elevadores são mais utilizados.
Encerrar com um tipo inválido para elevador.

24. Faça um programa que leia dia, mês e ano respectivamente, ou seja, uma data qualquer e faça consistência desta data (avaliar se é valida ou não). O programa deverá prever teste para ano bissexto. Finalizar a leitura com dia=0.

25. Faça um programa para verificar se um valor lido é número perfeito. Para que um valor seja considerado número perfeito a soma dos seus divisores deve resultar num valor igual a ele mesmo.
Exemplo: os divisores de 6 são : 1, 2, 3 que somados totalizam 6 – 6 é quadrado perfeito.

26. Faça um programa que receba os nomes de 5 produtos de uma loja e o preço unitário de cada um deles (em reais). O programa deverá informar, com mensagens explicativas:

a) o nome dos produtos cujo preço é superior a 20 reais;
b) o nome e o preço correspondente dos produtos cujo preço é inferior a 10 reais;
c) o preço médio dos produtos.

27. Escrever um programa que leia um número inteiro e positivo n por um número indeterminado de vezes. Se n for par, verificar e informar quantos divisores possui. Se n for ímpar e menor que 8 escrever seu fatorial, se n for ímpar e maior ou igual a 8 escrever a soma dos inteiros de 1 a n. Finalizar com n<=0.

28. Escreva um programa que lê um número não determinado de pares de valores a, b, (com a < b) todos inteiros e positivos, um par de cada vez, calcula e escreve a soma dos n inteiros consecutivos a partir de a inclusive até b. Encerrar com valor = 0.

29. Escrever um programa que lê n, inteiro e positivo e calcula e escreve o termo de ordem n da sucessão abaixo:

ordem:	1	2	3	4	5	6	7	8	9	10
termo		-1	0	5	6	11	12	17	18	23

24

30. Fez-se uma pesquisa entre os 2500 habitantes de uma região para coletar os seguintes dados: sexo, idade e altura. Escreva um programa que lê estas informações e informa:

A média da idade do grupo;
A média da altura das mulheres com mais de 21 anos;
A maior altura entre os homens;
O percentual de pessoas com idade entre 18 e 30 anos.

31. Foi feita uma pesquisa nas 20 principais cidades do estado para coletar dados sobre acidentes de trânsito. Foram obtidos os seguintes dados:

Código da cidade (1 a 20);
Número de veículos de passeio do ano 2001;
Número de acidentes com vítimas do ano 2001;

Escreva um programa que lê um conjunto de dados conforme acima descrito para cada cidade, um de cada vez, calcula e escreve:

Qual o maior e o menor número de acidentes e a que cidade pertence.
Qual a média de veículos das cidades pesquisadas.
Qual a média de acidentes entre as cidades pesquisadas.

32. Foi realizada uma pesquisa de algumas características físicas da população de certa região. Foram entrevistadas 500 pessoas e coletados os seguintes dados:

a- sexo: M (masculino) e F (feminino)
b- cor dos olhos: A (azuis), V (verdes) e C (castanhos)
c- cor dos cabelos: L (louros), C (castanhos) e P (pretos)
d- idade

Deseja-se saber:

- a maior idade do grupo;
- a quantidade de indivíduos do sexo feminino, cuja idade está entre 18 e 35 anos e que tenham olhos verdes e cabelos louros.

Elabore um programa para o caso apresentado acima.

33. Faça um programa que leia o nome e a idade dos nadadores de uma competição e classifique-os nas categorias abaixo:

5 a 7 anos : Infantil A
8 a 11 anos: Infantil B
12 a 14 anos: Juvenil A
15 a 17 anos: Juvenil B
O programa termina quando for lida idade = 0, devendo então ser informado:

- quantos nadadores participaram em cada categoria;
- qual o percentual da cada categoria sobre o total de nadadores.

34. Foi realizada uma pesquisa em Santa Cruz, com um grupo de 1000 pessoas. De cada entrevistado foram colhidos os seguintes dados:
a) clube de preferência ('G'- Grêmio; 'I' – Internacional; 'O' – Outros);
b) cidade de origem ('S' – Santa Cruz; 'O'- Outras);
Deseja-se saber:

- número total de torcedores do Grêmio e do Internacional;
- número de pessoas nascidas em Santa Cruz e que não torcem nem pelo Grêmio e nem pelo Inter.

35. A prefeitura de uma cidade fez uma pesquisa entre seus habitantes, coletando dados sobre o salário e número de filhos. A prefeitura deseja saber:
a) média do salário da população;
b) média do número de filhos;
c) maior salário;
d) percentual de pessoas com salário até R$200,00.
Encerrar o programa quando um salário negativo for informado.

36. Em uma eleição presidencial existem quatro candidatos. Os votos são informados através de códigos. Os dados utilizados para a contagem dos votos obedecem à seguinte codificação:
- 1, 2, 3, 4 = voto para os respectivos candidatos;
- 5 = voto nulo;
- 6 = voto em branco;
- 0 = encerrar leitura.

Elabore um programa que leia o código do candidato e um voto. Calcule e escreva:
- total de votos para cada candidato;
- total de votos nulos.

37. Escrever um programa que leia uma quantidade desconhecida de números e conte quantos deles estão nos seguintes intervalos: [0, 25], [26, 50], [51, 75] e [76, 100]. A entrada de dados deve terminar quando for lido um número negativo.

38. Escreva um programa que gere os números de 1000 a 1999 e escreva aqueles que divididos por 11 dão resto igual a 5.

39. Escrever um programa que leia 5 conjuntos de 2 valores, o primeiro representando o número de um aluno, e o segundo representando a sua altura em centímetros. Encontre o aluno mais alto e o mais baixo. Mostre o número do aluno mais alto e do mais baixo, junto com suas alturas.

40. Escrever um programa que calcula e escreva o produto dos números primos entre 92 e 1478.

6.6 EXERCÍCIOS RESOLVIDOS

Exercício 1 resolvido com o comando Para:

Algoritmo	C

algorimo valores;
 	
inteiro v, s= 0, me, x;
inicio
para x= 0 to 4 faça
inicio
leia(v);
se x == 1 então
me= v
senão
se v < me então
me= v;
se v % 2 = 0 então
s= s + v;
fim;
escreva('menor valor: ', me);
escreva('soma dos pares: ', s);
fim.
	#include <stdio.h>
	void main(void)
	{
	int v, s= 0, me, x;
	printf("Digite 5 valores inteiros \n");
	for (x= 0; x <= 4; x++)
	{
	scanf("%i",&v);
	if (x == 1)
	me= v;
	else
	if (v < me)
	me= v;
	if (v%2 == 0)
	s= s + v;
	}
	printf("Menor valor: %i", me);
	printf("\n Soma dos pares: %i \n", s);
	}

Exercício 1 resolvido com o comando Enquanto:

Algoritmo	C
algoritmo valores;	#include <stdio.h>

```
integer    v, s= 0, me, x= 1;
inicio



 enquanto x <= 5 faça
 inicio
  leia(v);
  se x == 1 então
     me= v;
  senão
     se v <me então
        me= v;
  se v  %  2 = 0 então
      s= s + v;
  x= x + 1;
  fim;
  escreva('o menor entre os valores lidos é: ',
me);
  escreva('a soma dos números pares é: ', s);
fim.
```

```
void main(void)
{
int   v, s= 0, me, x= 1;
          printf("Digite 5 valores inteiros \n");
          while (x <= 5)
          {
                    scanf("\n %i", &v);
                    if (x == 1)
                              me= v;
                    else
                              if (v < me)
                                   me= v;
                    if (v%2 == 0)
                       s= s + v;
                    x++;
          }
   printf("\n Menor Valor %i", me);
   printf("\n Soma dos pares: %i \n", s);
}
```

Exercício 1 resolvido com Repita:

Algoritmo	C

algoritmo valores;
var
inteiro   v, s= 0, me, x= 1;
inicio
 	
repita
leia(v);
se x == 1 então
	#include <stdio.h>
	void main(void)
	{
	int v, s= 0, me, x= 1;
	printf("Digite 5 valores inteiros \n");
	do{
	scanf("\n %i", &v);
	if (x == 1)

me= v;
senão
se v < me então
me= v;
se v % 2 = 0 então
s= s + v;
x= x + 1;
ate x <= 5;
escreva('o menor entre os valores lidos é: ', me);
escreva('a soma dos números pares é: ', s);
fim.
	me= v;
	else
	if (v < me)
	me= v;
	if (v%2 == 0)
	s= s + v;
	x++;
	}while (x <= 5);
	printf("\n Menor Valor %i" , me);
	printf("\n Soma dos pares:%i \n", s);
	}

Exemplo 8:

Algoritmo	C

algoritmo ex_8;
real nota1, nota2, mediaa, mediag= 0; real acummedia= 0;
inteiro a= 1;
inicio
acummedia= 0;
 	
enquanto (a < 5) e (mediag > 5) faça
inicio
mediaa= 0;
leia(nota1, nota2);
mediaa= (nota1 + nota2) / 2;
acummedia=acummedia + mediaa;
mediag= acummedia / a;
a= a + 1;
escreva('mediaaluno: ', mediaa);
	#include <stdio.h>
	void main(void)
	{
	float nota1, nota2, mediaa, mediag= 0, acummedia= 0;
	int a= 1;
	
	while (a <= 5 && mediag > 5)
	{
	printf("Digite as 2 notas do aluno %i \n", a);
	mediaa= 0;
	scanf("\n %f%f", &nota1, &nota2);
	mediaa= (nota1 + nota2) / 2;
	acummedia= acummedia + mediaa;
	printf("\n Mediaaluno:%.2f \n",

fim;
escreva('mediageral: ', mediag);
fim.
	mediaa);
	mediag= acummedia / a;
	a= a + 1;
	}
	printf("\n Media geral: %.2f \n", mediag);
	}

Exercício 14:

Algoritmo	C

algoritmo tabuada;
caractere op;
inteiro num, c;
inicio
 	
repita
escreva('digite um número');
leia(num);
 	
para c= 1 to 10 faça
escreva(c,'x',num,'=',c*num);
escreva("escrever a tabuada?(s/n): ");
leia(op);
até (op == 's') ou (op == 'S');
fim.
	#include <stdio.h>
	#include <conio.h>
	void main(void)
	{
	char op;
	int num, c;
	do{
	printf("\n Digite o numero \n");
	scanf("%i", &num);
	for (c= 1; c<= 10; c++)
	printf("\n %ix%i=%i", c, num, c*num);
	printf("\n Escrever a tabuada(s/n)? \n");
	op= getch();
	} while (op == 's' || op == 'S');
	}

Referências biliográficas:

Ziviani, Nivio. Projeto de Algoritmos em C e Java. 2ª ed. São Paulo: Thomson Pioneira, 2004

Feofiloff, Paulo. Algoritmos em Linguagem em C. 1ª ed. Rio de Janeiro: Campus, 2008

Fertig, Cristina; Medina, Marco. Algoritmos e Programação Teoria e Prática. 1ª ed. São Paulo: Novatec, 2005

Oliveira, Álvaro; Boratti, Isaias. Introdução a Programação de Algoritmos. 3ª ed. Florianópolis: Visual Books, 2007

Garcia, Guto; Lopes, Anita. Introdução à Programação: 500 Algoritmos Resolvidos. 1ª ed. Rio de Janeiro: Campus, 2002

Lux, Beatriz; Furtado, João Carlos. Apostila de Algoritmos e Programação. Departamento de Computação, UNISC, 2010

Parte VII

7. TIPO ESTRUTURADO HOMOGÊNEO: VETORES E MATRIZES

7.1 VETORES

Um vetor é um tipo de dado que armazena na memória do computador, de forma estruturada, um número determinado de informações, todas do mesmo tipo.

Sintaxe:

tipo_base identificador[n];

Onde:

- **identificador** é o nome escolhido para a variável;
- **n** identifica o tamanho do vetor ou quantidade de elementos;
- **tipo_base** é o tipo que têm os elementos do vetor.

Exemplo em C:

int V[5];

Podemos desenhar a estrutura de V da seguinte forma:

V =

Após a leitura de dados, os mesmos são armazenados no vetor V da seguinte forma, por exemplo:

V =

5	4	1	8	3

Cada elemento de V tem uma posição bem definida, de tal forma que podemos afirmar que V[**2**] = 1, ou seja, esta estrutura atribui um **índice** a cada um dos seus elementos que identifica a posição do elemento na estrutura. Estes índices serão

sempre valores inteiros, consecutivos e a primeira posição do vetor é sempre indicada pelo índice 0. O acesso a qualquer dos elementos se dará através do seu índice.

Se fizermos as atribuições V[**1**]= 25 e V[**4**]= 30 teremos:

V =

5	**25**	1	8	**30**

Dadas estas características podemos concluir que a leitura dos elementos de um vetor deve ser feita utilizando um comando de repetição, onde o teste de repetição se dará sobre uma variável do tipo inteiro, que iniciará com o valor 0 e será incrementada até atingir o valor que define o tamanho do vetor.

Adotando o vetor V, acima declarado, mais uma variável "i" do tipo inteiro teríamos:

```
for(i= 0; i < 5; i++)
    scanf("%d", &V[i]);
```

Da mesma forma seria feita a apresentação dos elementos do vetor:

```
for(i= 0; i < 5; i++)
    printf("\n Valor do vetor: %d", V[i]);
```

Exemplo: Faça um programa para ler os elementos de um vetor (A) com 10 elementos inteiros e construir um segundo vetor (B) de mesma dimensão e cujos elementos serão os elementos de A multiplicados por 5.

```
#include <stdio.h>
int main(void)
{
   int a[10], b[10], i;

   for (i = 0; i < 10; i++)
      {
         printf("\n Digite o valor: ");
         scanf("%d", &a[i]); // entrada de dados
         b[i]= a[i] * 5;
      }
}
```

7.2 MATRIZES

Como os vetores, as matrizes também permitem que armazenemos na memória do computador um número determinado de dados, todos do mesmo tipo, que ficam organizados de forma bem estruturada, porém cada elemento terá sua posição definida através de dois índices.

Sintaxe:

tipo_base identificador[m] [n];

Onde:

- **identificador** é o nome escolhido para a variável;
- **m** identifica o número de linhas da matriz e **n** identifica o número de colunas;
- **tipo_base** é o tipo que têm os elementos da matriz.

Exemplo em C:

int M[2][3];

Podemos desenhar a estrutura de M da seguinte forma:

M =

Após a leitura dos dados que serão armazenados em M teríamos:

M =

30	5	22
11	42	**13**

Cada elemento de M tem uma posição bem definida, de tal forma que podemos afirmar que M[**1**, **2**] = 13, ou seja, esta estrutura atribui dois índices a cada um dos seus

elementos que identificam a posição do elemento na estrutura. O primeiro índice indica a linha e o segundo a coluna da matriz.

Se fizermos as atribuições M**[0][1]**= 25 e M**[1][1]**= 10 teremos:

M =

30	**25**	22
11	**10**	13

Dadas estas características podemos concluir que a leitura dos elementos de uma matriz deve ser feita utilizando dois comandos de repetição aninhados. Adotando a matriz M, acima declarada, mais duas variáveis "i" e "j", do tipo inteiro teríamos:

```
for(i= 0; i < 3; i++)
    for(j= 0; j < 2; j++)
            scanf("%d", &M[i][j]);
```

Da mesma forma seria feita a apresentação dos elementos da matriz:

```
for(i= 0; i < 3; i++)
    for(j= 0; j < 2; j++)
            printf("\n Valor da matriz: %d", M[i][j]);
```

Exemplo: Faça um programa para ler os elementos de uma matriz (A) com 10 x 5 elementos inteiros e construir uma segunda matriz (B) de mesma dimensão e cujos elementos serão os elementos de A multiplicados por 5.

```
#include <stdio.h>
int main(void)
{
   int a[10][5], b[10][5], i, j;
   for (i = 0; i < 10; i++)
        for (j = 0; j < 5; j++)
           {
              printf("\n Digite o valor: ");
              scanf("%d", &a[i][j]); // entrada de dados
              b[i][j]= a[i][j] * 5;
           }
}
```

7.3 EXERCÍCIOS PROPOSTOS

Exercícios identificados com (*) estão resolvidos ao final do capítulo.
Implemente todos algoritmos construídos.
Este material segue as referências bibliográficas indicadas ao final, principalmente de Lux e Furtado (2010).

1. Faça um programa que leia 200 valores do tipo inteiro e armazene-os em um vetor. A seguir o programa deverá informar:
a) todos os números pares que existem no vetor;
b) o menor e o maior valor existente no vetor;
c) quantos dos valores do vetor são maiores que a média desses valores.

2. Crie um programa para ler 100 nomes e imprimir o nome caso este seja "Adão", informando também sua posição no vetor.

3. (*) Crie um programa para ler 30 números reais e imprimir os números que estão nas posições pares do vetor.

4. (*) Crie um programa para ler 38 números e imprimir a quantidade de números primos existentes neste vetor.

5. Faça um programa para localizar os nomes iguais existentes entre dois vetores A com "n" elementos e B com "m" elementos, imprimindo o resultado para o usuário. Considere n=20 e m=25. Considere que não haverá repetição de nomes no vetor A e nem no vetor B.

6. Fazer um programa para ler 100 números e imprimir um vetor resultante onde os elementos serão o dobro dos elementos lidos.

7. Escrever um programa para ler um vetor com 100 elementos reais e inverter a posição destes elementos, de tal modo que o primeiro elemento venha a ser o último depois da inversão.

8. Dado que para cada aluno de uma turma de 35 alunos se tenha, o seu nome, e as notas de 3 avaliações. Faça um programa que:
   a) Imprima o nome a média de cada aluno.
   b) Calcule a Percentagem de alunos com média acima da média da turma.
   c) Para cada aluno imprima uma mensagem dizendo se o aluno tem ou não notas repetidas.
   d) Determine quantos alunos tem pelo menos duas notas acima de 7.

9. Faça um programa que leia, Nome idade e sexo de 50 pessoas. Após a leitura faça:

a) Imprima o Nome, idade e sexo das pessoas cuja idade seja maior que a idade da primeira pessoa.
b) Imprima o Nome e idade de todas as mulheres.
c) Imprima o Nome dos homens menores de 21 anos.

10. Faça um programa para ler 2 números, n1 e n2 . A soma destes não deve ultrapassar 10, se ultrapassar, os 2 números devem ser lidos novamente.
A seguir, ler um vetor de 10 posições, de modo que contenha n1 valores ímpares e n2 pares, obrigatoriamente.

11. Faça um programa que leia o nome e três notas de "N" alunos de um colégio. O usuário informará qual é o valor de "N", sendo no máximo de 50 alunos. Após a leitura faça:
a) Imprima o Nome e a média de cada aluno aprovado (Média >= 7.0).
b) Imprima o Nome e a média dos alunos reprovados (Média < 5.0).
c) Imprima o percentual de alunos aprovados.

12. Dado Nome, tempo de serviço em anos e salário de 30 funcionários faça:
a) Imprima o Nome e o salário dos funcionários que ganham mais de R$ 500,00
b) Para os funcionários que ganham menos de R$ 200,00 conceda um aumento de 20%.
c) Para os funcionários que trabalham a mais de 3 anos na empresa, forneça um aumento de R$100.
Imprima Nome e o novo salário destes funcionários aumentados

13. Faça um programa para ler 50 valores inteiros. Após imprima tais valores ordenados crescentemente.

14. Faça um programa que leia 25 valores numéricos inteiros em um vetor e três valores inteiros, nas variáveis A, B e C. Após a leitura emita um relatório com cada valor diferente de A, B e C e o número de vezes que o mesmo apareceu no vetor.

15. Faça um programa que:
a) Leia um vetor A com 10 elementos e um vetor B com 15 elementos do tipo real.
b) Intercale estes vetores A e B, formando outro vetor C da seguinte forma.

C[1] = A[1]
C[2] = B[1]
C[3] = A[2]
C[4] = B[2]

Como A tem menos elementos que B, o vetor C deverá ser preenchido ao final com os elementos restantes do vetor B.

16. Elaborar um programa que lê um conjunto de 30 valores e os coloca em 2 vetores conforme forem pares ou ímpares. O tamanho de cada vetor é de 5 posições. Se algum vetor estiver cheio, escrevê-lo. Terminada a leitura escrever o

conteúdo dos dois vetores. Cada vetor pode ser preenchido tantas vezes quantas forem necessárias.

17. Dado um vetor V de 100 elementos inteiros faça um programa que:

Crie outro vetor W contendo os valores de V que estão no intervalo 10 a 40 (incluindo tais números);
Crie outro vetor Z contendo os valores de V que estão em posições múltiplas de 3 ou de 5;

18. Fazer um programa que:

- leia um vetor A com 30 valores inteiros;
- leia um outro vetor B com 30 valores inteiros;
- leia o valor de uma variável X;
- forneça quais os elementos de A que são múltiplos de X e suas posições no vetor;
- Informe o vetor soma de A e B.

19. Escreva um programa que leia um vetor de inteiros com 20 posições e a seguir altere esse vetor de modo que os valores ímpares apareçam intercalados.

20. Faça um programa que leia, Nome idade e sexo de 50 pessoas. Após a leitura faça:

a) Imprima o Nome, idade e sexo das pessoas cuja idade seja maior que a idade da primeira pessoa.
b) Imprima o Nome e idade de todas as mulheres.
c) Imprima o Nome dos homens menores de 21 anos.

Todos os dados devem ser armazenados em vetores.

21. Escrever um programa para ler uma matriz 5 X 5 de elementos inteiros, após somar em um acumulador todos os elementos das linhas pares, e em outro acumulador os elementos das linhas ímpares.

22. Faça um programa para ler os valores inteiros de uma matriz com dimensão 4 x 6 e calcular quantos destes elementos são valores primos.

23. Faça um programa que leia uma matriz 10 X 50 de valores inteiros e acumule somente os elementos que estão posicionados onde ambos os índices sejam múltiplos de 3 ou de 5.

24. Faça um programa para ler os valores de uma matriz com 4 linhas e 4 colunas e fornecer:

- A soma dos elementos da diagonal secundária;
- Os valores pares que compõem a diagonal principal.

25. Faça um programa que leia os valores reais de uma matriz 10 x 15 e forneça a soma dos elementos que encontram-se em posições cuja soma dos índices seja um número ímpar.

26. Faça um programa que leia os elementos de uma matriz 5 x 7 de valores inteiros e forneça o maior e o menor elementos desta matriz e a posição em que estão dentro da estrutura.

27. Faça um programa para ler as notas de uma turma com 30 alunos considerando três notas por alunos. Utilize uma matriz para armazenar estes valores considerando que em cada linha da matriz estarão as três notas de cada aluno. O programa deverá calcular a média aritmética de cada aluno e a média geral da turma.

28. Escreva um programa que leia uma matriz 5 X 6 de valores reais, a seguir divida cada um dos números de cada linha pelo maior número da linha, criando assim uma nova matriz. Escreva a matriz original e a nova matriz.

29. Ler um vetor de 25 elementos inteiros e a partir desse vetor, criar uma matriz 5 X 5 preenchendo-a da seguinte forma: os cinco primeiros elementos do vetor formarão a primeira linha, os próximos 5 elementos formarão a segunda linha e assim por diante. Imprimir o vetor e a matriz no final.

30. Escreva um programa que lê uma matriz M (5 x 5) e troque a seguir, conforme abaixo indicado:

- a linha 2 com a linha 5;
- a coluna 1 com a coluna 4;
- a diagonal principal com a diagonal secundária.

31. Escreva um programa que lê uma matriz M (5 x 5) e cria dois vetores SL e Sc que contenham, respectivamente, as somas das linhas e das colunas de M. Escrever os vetores.
Ler um vetor de 5 elementos inteiros e preencher cada linha de uma matriz de 3 X 5 com esse vetor.

32. Escrever um programa que lê uma matriz A (15 x 5) e a escreva. Verifique, a seguir, quais os elementos de A que estão repetidos e quantas vezes cada um está repetido. Escrever cada elemento repetido com uma mensagem dizendo que o elemento aparece X vezes em A.

33. Ler um número não determinado de valores inteiros até que o número lido seja 0 e, para cada número lido, verificar se ele pertence a uma matriz 8x8, cujos valores, também inteiros, já haviam sido lidos anteriormente. Em caso afirmativo, escreva o valor, e a posição em que foi encontrado, caso contrário, escreva o valor e a mensagem “Não encontrado”.

34. Escreva um programa que leia uma matriz M (5 x 5) e determine o elemento minimax desta matriz, escrevendo o seu valor e a posição em que se encontra na matriz. O elemento minimax de uma matriz é o menor elemento da linha em que se encontra o maior elemento da matriz.

35. Escrever um programa que lê um vetor G com 13 elementos inteiros que é o gabarito de um teste da loteria esportiva. Neste vetor serão lidos os valores 1 (para coluna 1), 2 (para coluna 2) e 3 (para coluna do meio).

36. Ler, a seguir, para cada 50 apostadores, o número de seu cartão e um vetor R, com 13 elementos, onde estarão as suas apostas e cujos valores também serão 1, 2 ou 3. Verificar para cada apostador o número de acertos e escrever o número de seu cartão e seu número de acertos.
Para cada jogador que tiver 13 acertos, acrescentar a mensagem: "GANHADOR". No final escrever quantos apostadores conseguiram 13 pontos e quantos não acertaram nenhum ponto.

37. Escrever um programa que lê uma matriz M 5x5 e um valor A, inteiro e multiplica cada elemento de M pelo valor A e coloca estes resultados em um vetor com 25 elementos. O programa deverá escrever os valores do vetor.

38. Faça um programa que leia uma matriz 4 x 2 de elementos inteiros e calcule a sua transposta, sabendo que a transposta de uma matriz m é uma outra matriz n obtida da seguinte forma: as linhas de m passam a ser as colunas de n.
Ex: sendo m uma matriz 3x3 e n a sua transposta

$$m = \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} \qquad n = \begin{matrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{matrix}$$

39. Deseja-se fazer uma pesquisa a respeito do consumo de energia elétrica em uma determinada cidade. Para isso deve ser lido um vetor com 12 elementos reais, onde cada elemento representa o consumo em KWH de um mês do ano. Calcular e escrever os seguintes resultados para cada habitante:

- o menor e o maior consumo residencial e o mês que ocorreu;
- a média do consumo Residencial nos meses de verão (1, 2, 12) e nos meses de inverno (6, 7, 8).

Considere um total de 100 moradores para a pesquisa.

40. Faça um programa que leia uma matriz A 2 x 3 com elementos inteiros e uma matriz B 3 x 2 também com elementos inteiros, calcule e informe a matriz C, que é o resultado da multiplicação de A por B.

41. Faça um programa que calcule a multiplicação de duas matrizes A e B, sendo inicialmente lidos os valores que correspondem ao número de linhas e colunas destas matrizes. Deverá então ser feito o teste para verificar se pode ocorrer

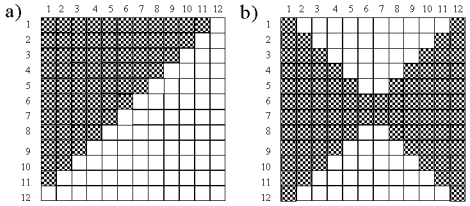
multiplicação entre A e B e caso seja possível, o programa deverá ler as duas matrizes e informar a sua multiplicação.

42. Faça um programa que utiliza uma matriz 4 x 3 para ler os preços de três produtos da cesta básica (colunas) em quatro supermercados (linhas). O programa deverá calcular e informar:

a) a posição (linha e coluna) do maior preço;
b) a soma dos produtos por supermercado;
c) a média dos preços de cada produto.

43. Faça um programa que leia uma matriz 8 x 8 de inteiro e informe a média dos valores que se encontram acima da diagonal principal.

44. Faça um programa que leia uma matriz 12 x 12 e calcule e escreva a soma da área hachurada na letra a e o maior elemento da área hachurada na letra b abaixo:



45. Faça um programa que leia os valores inteiros de uma matriz 3 x 3 e crie a partir destes um vetor cujos elementos sejam o fatorial dos elementos da matriz lida.

46. Faça um programa para ler os valores (inteiros) de um vetor com 10 posições, sendo que o valor lido somente será aceito se for múltiplo do índice correspondente.

7.4 EXERCÍCIOS RESOLVIDOS

Exercício 2:

C

```
int main(void)
{
 float vetor[N_MAX];
 int x;

 printf("\n Digite os valores do vetor: ");
 for(x = 0; x <= N_MAX; x++)
            scanf("%f", &vetor[x]);
 printf("\n Estes são os valores das posições pares: ");
 for(x = 0; x <= N_MAX; x++)
           if(x % 2 == 0)
              printf("\n %.2f ", vetor[x]);

 getch();  // parada da tela
```

```
}
```

Exercício 3:

C

```
int main(void)
{
 int   i, j, c, cprimos= 0; // i= variável de controle do for, inicializa variável contadora
 float n[N_MAX];

 for(i = 0; i < N_MAX; i++)
      {
         c = 0;            // inicializa variável
         printf("\n Digite valor: ");
         scanf("%f", &n[i]); // entrada de dados

         for (j= 1; j <= n[i]; j++)
             if (fmod(n[i],j) == 0) // verifica se a divisão tem resto zero
                c++;
         if (c <= 2)        // verifica se é primo
            cprimos++;
      };
   printf("\n Quantidade de primos: %d", cprimos);
   getch();  // parada da tela
}
```

Exercício 46:

C

```
int main(void)
{
  int    n, i, vet[10];

  printf("\n Forneça os valores do vetor: ");
  for(i= 0; I <= 9; i++)
            {           scanf("%i", &n);
                        if(i > 0)
                           while((n <= i) || (n % i != 0))
                                    {
                                                printf("\n Invalido");
                                                scanf("%i", &n);
                                    }
                        vet[i]= n;
            }
  for(i= 0; i <= 9; i++)
        printf("\n Posição %i  Valor = %i  ", i, vet[i]);
}
```

Referências biliográficas:

Ziviani, Nivio. Projeto de Algoritmos em C e Java. 2ª ed. São Paulo: Thomson Pioneira, 2004

Feofiloff, Paulo. Algoritmos em Linguagem em C. 1ª ed. Rio de Janeiro: Campus, 2008

Fertig, Cristina; Medina, Marco. Algoritmos e Programação Teoria e Prática. 1ª ed. São Paulo: Novatec, 2005

Oliveira, Álvaro; Boratti, Isaias. Introdução a Programação de Algoritmos. 3ª ed. Florianópolis: Visual Books, 2007

Garcia, Guto; Lopes, Anita. Introdução à Programação: 500 Algoritmos Resolvidos. 1ª ed. Rio de Janeiro: Campus, 2002

Lux, Beatriz; Furtado, João Carlos. Apostila de Algoritmos e Programação. Departamento de Computação, UNISC, 2010

Parte VIII

8. TIPO ESTRUTURADO HETEROGÊNEO (Registros ou Estruturas)

8.1 REGISTRO

Um **registro é uma estrutura composta** por um número fixo de componentes, chamados **campos**. Trata-se de um grupo de informações relativas a uma mesma entidade. Os campos do registro podem ser de diferentes tipos e a cada campo é dado um nome (identificador do campo), o qual é utilizado para selecioná-lo.

As estruturas são utilizadas para **representar um conjunto de informações** que estão logicamente relacionadas, por exemplo:

- Uma ficha de cadastro de uma empresa que tem todo tipo de informações sobre um funcionário, necessárias para a empresa.
- Uma ficha de um consultório médico que possui alguns dados e características do estado de saúde dos pacientes.

8.2 Declaração

Quando for necessário utilizar um registro é preciso definir um novo tipo:

Sintaxe:

```
struct nome_do_novo_tipo
{
    tipo1     campo11, campo12, ..., campo1N;
    tipo2     campo21, campo22, ..., campo2N;
    ...
    tipoM     campoM1, campoM2, ... campo MN;
};
```

Onde:

- **nome_do_novo_tipo** é o identificador que especifica o nome do registro criado (novo tipo de dados). A partir desse nome, junto com a palavra-chave *struct*, podem ser criadas variáveis desse tipo;
- **tipo1, tipo2, tipo3, ... tipoM** são os tipos de cada um dos campos do registro;
- **campo11, campo12, ..., campoMN** são os nomes dos campos do registro.

Exemplo:

```
// declaração do tipo struct
struct FICHA_CADASTRAL
{
        char cidade[41], estado[31], pais[21], nome[51], profissao[51], escolaridade[21];
        int quantidadeDependentes;
        float salario;
};

// declaração das variáveis do tipo criado struct FICHA_CADASTRAL
struct FICHA_CADASTRAL funcionario1, funcionario2;
```

Outro exemplo:

```
struct ALUNO
{
   char nome[20];
   char curso [25];
   float nota;
};
```

Após a declaração da **struct ALUNO** é possível declarar variáveis do tipo criado, exemplo:

struct ALUNO a1, a2; *// variáveis a1 e a2 são do tipo struct ALUNO*

Uma vez que o tipo de uma estrutura foi declarado, é possível utilizá-lo em outras declarações (variáveis simples, vetores, funções, etc). Exemplo:
struct ALUNO turma[55];

Também é possível informar, logo ao final da declaração da estrutura, as suas variáveis. Veja um exemplo:

```
struct REG_CLIENTE{    // nome da estrutura
    char nome[30];      //campos
    float salario;
    int idade;
}cliente, fornecedor;  // lista de variáveis
```

Os campos de uma estrutura podem ser de qualquer tipo: tipos simples (int, char, float, etc), vetores, ou até mesmo outras estruturas, por exemplo:

```
struct NOME_COMPLETO{
     char primeiro[30];
     char meio[30];
     char sobrenome[30];
};

struct REG_PESSOA{
     NOME_COMPLETO nome;
     int idade;
};

int main(void){
 REG_PESSOA pessoa[N_MAX]; // pessoa é uma variável vetor do tipo REG_PESSOA
 int i;
```

```
 for(i= 0; i < N_MAX; i++){
      printf("\n Digite seu primeiro nome: ");
      fflush(stdin);
      gets(pessoa[i].nome.primeiro);
      printf("\n Digite seu nome do meio.: ");
      fflush(stdin);
      gets(pessoa[i].nome.meio);
      printf("\n Digite seu sobrenome....: ");
      fflush(stdin);
      gets(pessoa[i].nome.sobrenome);
      printf("\n Digite sua idade........: ");
      fflush(stdin);
      scanf("%d", &pessoa[i].idade);

      printf("\n Nome completo: %s %s %s  \n Idade: %d anos\n", pessoa[i].nome.primeiro,
    pessoa[i].nome.meio, pessoa[i].nome.sobrenome, pessoa[i].idade);
     }
     getch();  // parada da tela
}
```

Outros exemplos:

```
struct PONTO
{
      int x;
      int y;
};
struct LINHA
{
      struct PONTO pa;
      struct PONTO pb;
 };
struct LINHA l1,l2;
```

8.3 Acesso aos campos da estrutura

O acesso aos campos é feito através do operador de associação – o **ponto** (**.**).

Exemplos:

```
a1.nome= ”José Carlos dos Santos”;
a1.curso= ”Ciência da Computação”;
a1.nota= 10;
scanf(“%s”, &a1.nome);
scanf(“%d”, &a1.nota);

l1.pa.x= 3.0;
l1.pa.y= 5.0;
l1.pb.x= 15.0;
l1.pb.y= 5.0;
scanf(“%d”, &l1.pb.y);
```

8.4 Atribuição entre variáveis do tipo struct

É possível fazer atribuições entre variáveis de registro, porém elas devem ser do mesmo tipo **struct** declarado.

Exemplos:

```
a2= a1; // as variáveis a1 e a2 devem ser do mesmo tipo
l2 = l1;
```

8.5 Vetor e matriz de estrutura

É possível definir um vetor ou uma matriz do tipo estrutura, por exemplo:

```
#include <stdio.h>
#include <conio.h>
struct CONTA{
    int num;
```

```
        float saldo;
};

void main(void) {
 struct CONTA c[2]; // c é uma variável do tipo vetor de CONTA
 c[0].num = 2;
 c[0].saldo = 20.0;
 c[1].num = 3;
 c[1].saldo = 30.0;
 printf ("Main: Valor de c.num: %d\n", c[0].num);
 printf ("main: Valor de c.saldo: %.1f\n",c[0].saldo);
 printf ("Main: Valor de c.num: %d\n", c[1].num);
 printf ("main: Valor de c.saldo: %.1f\n",c[1].saldo);
 getch();
}
```

Ou ainda, você deve usar:

```
#include <stdio.h>
#include <conio.h>
struct CONTA{
        int num;
        float saldo;
};

void main(void) {
 int i;
 struct CONTA c[100]; // c é uma variável do tipo vetor de CONTA

 for(i= 1; i <= 100; i++){
```

```
    scanf("%d", &c[i].num);
    scanf("%f", &c[i].saldo);
 }
 for(i= 1; i <= 100; i++){
    printf ("Numero: %d \n", c[i].num);
    printf ("Saldo....: %.2f \n", c[i].saldo);
 }
 getch();
}
```

Também podemos definir tipos com **typedef** e, então, definir nomes para estruturas de dados (struct). Desta forma, podemos auto-referenciar a estrutura, ou seja, colocar um tipo de dado struct dentro de outro struct.

Por exemplo:

```
#include <stdio.h>
#include <conio.h>
 typedef struct {
      int num;
      float saldo;
}CONTA;

void main(void) {
 int i;
 CONTA c[100]; // c é uma variável do tipo vetor de CONTA

 for(i= 1; i <= 100; i++){
    scanf("%d", &c[i].num);
    scanf("%f", &c[i].saldo);
 }
 for(i= 1; i <= 100; i++){
```

```
    printf ("Numero: %d \n", c[i].num);
    printf ("Saldo....: %.2f \n", c[i].saldo);
 }
 getch();
}
```

8.6 EXERCÍCIOS PROPOSTOS

Exercícios identificados com (*) estão resolvidos ao final do capítulo.
Implemente todos algoritmos construídos.
Este material segue as referências bibliográficas indicadas ao final, principalmente de Lux e Furtado (2010).

1. (*) Fazer um programa que cria uma estrutura livro, que contém os elementos ISBN (código internacional, um inteiro), ano de edição, número de páginas e preço. Criar uma variável desta estrutura que é um vetor de 5 elementos. Ler os valores para a estrutura e imprimir a média do número de páginas dos livros.

2. Foi realizada uma pesquisa entre 500 habitantes de uma região. De cada habitante foram coletados os dados: idade, sexo, salário e número de filhos. Construa um programa C que armazene as informações da pesquisa em um vetor de estrutura e realize cada um dos requisitos:

- Calcular a média salarial dos habitantes.
- Calcular a média de filhos.
- Calcular o número de habitantes com salário inferior à média.
- Listar os habitantes com mais de 3 filhos.

3. Faça um programa para ler as seguintes informações relativas a 20 alunos da disciplina Algoritmos e Programação de um curso de Computação:

Matrícula (inteiro)
Nota Final (real)

O programa deverá utilizar um vetor de estrutura para armazenar estas informações e, após lê-las, fornecer cada uma das informações abaixo:

- A média da turma.
- O total de alunos aprovados, considerando que estão aprovados os alunos com nota final >=7.
- A matrícula do(s) alunos com maior Nota Final.

4. Faça um programa que leia nome, idade e sexo de 200 pessoas. Após a leitura faça:

- Imprima o nome, idade e sexo das pessoas cuja idade seja maior que a idade da primeira pessoa.
- Imprima o nome e idade de todas as mulheres.
- Imprima o nome dos homens menores de 21 anos.

5. Faça um programa que leia os preços e a descrição dos produtos da cesta básica. O programa deverá calcular e informar:

- o produto com maior preço e a descrição do mesmo;
- o valor total da cesta.

O programa finaliza no momento que o usuário informar que não deseja mais cadastrar produtos.

8.7 EXERCÍCIO RESOLVIDO

Exercício 1:

C

```
#include <stdio.h>
#include <conio.h>
#define N_MAX 3 // 5
struct REG_LIVRO{ // estrutura registro de um livro
    int isbn;
    int ano;
    int numero_paginas;
    float preco;
};
int main(void) // no turbo c utilizar void main(void)
{
 REG_LIVRO livro[N_MAX];
 int     i, acumulador= 0;
 for (i = 0; i < N_MAX; i++)
      {
         printf("\n\n ISBN...: "); // entrada de dados
         fflush(stdin);
         scanf("%d", &livro[i].isbn);
         printf("\n Ano....: ");
```

```
        fflush(stdin);
        scanf("%d", &livro[i].ano);
        printf("\n Paginas: ");
        fflush(stdin);
        scanf("%d", &livro[i].numero_paginas);
        printf("\n Preco..: ");
        fflush(stdin);
        scanf("%f", &livro[i].preco);
        acumulador= acumulador + livro[i].numero_paginas; // acumulador de páginas
    }
 printf("\n\n Media de paginas dos livros: %.2f", (float)acumulador/N_MAX);
 getch();  // parada da tela
 return 0; // no turbo c não usar esta linha
}
```

Referências biliográficas:

Ziviani, Nivio. Projeto de Algoritmos em C e Java. 2ª ed. São Paulo: Thomson Pioneira, 2004

Feofiloff, Paulo. Algoritmos em Linguagem em C. 1ª ed. Rio de Janeiro: Campus, 2008

Fertig, Cristina; Medina, Marco. Algoritmos e Programação Teoria e Prática. 1ª ed. São Paulo: Novatec, 2005

Oliveira, Álvaro; Boratti, Isaias. Introdução a Programação de Algoritmos. 3ª ed. Florianópolis: Visual Books, 2007

Garcia, Guto; Lopes, Anita. Introdução à Programação: 500 Algoritmos Resolvidos. 1ª ed. Rio de Janeiro: Campus, 2002

Lux, Beatriz; Furtado, João Carlos. Apostila de Algoritmos e Programação. Departamento de Computação, UNISC, 2010

Parte IX

9. MODULARIZAÇÃO (Subprogramas)

9.1 Introdução ao conceito de modularização

Nos exemplos e exercícios propostos até então, representamos as soluções para os problemas trabalhados em um único algoritmo. O que ocorre é que muitos problemas exigem soluções complexas e difíceis de entender quando implementadas através de longos algoritmos.

Para tornar a solução mais simples e inteligível utilizamos técnicas como a **divisão de um sistema em sub-rotinas**, permitindo assim **modularizar soluções** por meio de funções e procedimentos.

A divisão de um sistema em módulos apresenta algumas vantagens, como:

- *reduzir a complexidade do problema*, dividindo-o em subproblemas mais simples, que podem inclusive ser resolvidos por equipes independentes;
- é mais simples *alterar a composição de um módulo*, do que alterar a composição de um sistema integrado;
- é mais *fácil detectar problemas e resolvê-los*, pois os módulos são, em princípio, razoavelmente independentes;
- *reutilizar funcionalidade* em outros sistemas (código desenvolvido);
- *testar os módulos* individualmente do que o programa completo;
- é mais fácil *fazer a manutenção* (correção de erros, melhoramentos, etc.) módulo por módulo do que no programa total. Além disso, a modularização aumenta a probabilidade dessa manutenção não ter consequências graves para os outros módulos do programa.

Um programador assume, ao longo do desenvolvimento de um programa, os dois papéis descritos acima: por um lado é fabricante, pois é sua responsabilidade desenvolver módulos; por outro é utilizador, pois fará com certeza uso de outros

módulos, desenvolvidos por outrem ou por ele próprio no passado. É conveniente que um programador possa ser um mero utilizador dos módulos já desenvolvidos, sem se preocupar com o seu funcionamento interno: ele sabe qual a interface do módulo e qual a sua função, e usa-o.

Em nosso estudo, até agora, já entramos em contato com o uso de módulos prontos, como as funções que utilizamos para escrever na tela (*printf*) e fazer entrada de dados (*scanf* ou *gets*).

Existem basicamente duas formas de criar subprogramas: por **procedimentos** ou por **funções**. Um procedimento é um conjunto de instruções acionadas por outro programa ou, normalmente, pelo programa principal. Uma função por sua vez, também segue a definição acima, com a diferença de que deve ter, obrigatoriamente, um valor de retorno para quem a chamou. Funções só não retornam um valor se definirmos como ***void***, neste caso, são chamadas de função sem retorno.

Uma vez definida uma função (ou um procedimento), a mesma pode ser utilizada sem que se precise conhecer o seu funcionamento interno, da mesma forma que não interessa a um usuário de uma aparelhagem de som os circuitos dentro do amplificador, mas simplesmente as suas características. Para que o aparelho funcione adequadamente é preciso saber conectar o equipamento ao sistema e pronto! O mesmo acontece com as funções e os procedimentos.

Tomemos como exemplo uma função que é utilizada para gerar automaticamente números aleatórios, a função *rand* (função do C). Para poder utilizá-la devemos conhecer primeiro seu protótipo (**cabeçalho da função**). O protótipo irá nos mostrar qual **o tipo e os parâmetros** de rand. O protótipo é: **int rand(void)**

Onde:

int informa que a função retorna um valor inteiro, portanto é do tipo int

rand é o nome da função, no caso uma função definida pelo C

(void) indica que ela não requer parâmetros (variáveis) de entrada

Outro exemplo é a função matemática *sqrt* (função do C). Esta função, como já estudamos, informa o resultado (retorno) da raiz quadrada de um número. Para fazer este cálculo devemos colocar entre parênteses (como parâmetro) o valor ou a variável que servirá para fazer o cálculo da raiz quadrada e devemos declarar uma variável apropriada para armazenar o resultado. Veja o exemplo:

```
#include math.h
#include conio.h
void main void
{
        float r, n;
        r= sqrt(16); //r armazena o retorno da função cujo parâmetro de entrada é 16;
        e é do tipo float porque a função sqrt retorna um valor float!
}
```

Para usarmos funções prontas não necessitamos conhecer o seu código, porém é importante conhecer seu protótipo (tipo que retorna, nome da função e parâmetros que requer). Mas nem sempre utilizaremos apenas funções prontas, muitas vezes "fabricaremos" nós mesmos nossas funções! Vamos ver como faremos isso?

9.2 Definição de funções em C

Exemplifico abaixo a estrutura de uma função.

```
tipo_que_retorna nome_da_função (lista_de_parâmetros)
{
        bloco da declarações de variáveis locais;
        função comandos;
        return(variável ou valor); // para funções de tipo diferente de void
}
```

Para entender melhor o conceito de função (com ou sem retorno), vamos ver alguns exemplos, iniciando com um programa bastante simples, veja a seguir.

Exemplo 1: Calcular o fatorial de um número informado pelo usuário.

Sem função:

```
#include <stdio.h>
#include <conio.h> //biblioteca do getch();

int main(void)
{
 int n, fat= 1, i; //n = número informado pelo usuário   fat = valor do fatorial    i = controle da repetição for

 printf("\n Digite um numero: ");
 scanf("%d", &n);

 for(i= n; i > 0; i--)
    fat= fat * i; //calcula fatorial

 printf("\n O fatorial de %d e: %d", n, fat);
 getch();        //parada de tela
}
```

Com função sem retorno:

```
#include <stdio.h>
#include <conio.h> //biblioteca do getch();

// Declaração de procedimentos e funções
void fatorial(int n); //declaração da  função sem retorno - cabeçalho

// Programa principal
int main(void)
{
 int n;     //n = número informado pelo usuário

 printf("\n Digite um numero: ");
 scanf("%d", &n);
```

```
 fatorial(n);//com função sem retorno
 getch();    //parada de tela
}

//-------------------------
// Calcula Fatorial
// Entrada.: número inteiro lido
// Objetivo: calcular e imprimir o fatorial do numero de entrada
// Saída...: nenhuma
//-------------------------
void fatorial(int n) //com função sem retorno
{
 int i, fat= 1;    //i = controle da repetição for   fat = valor do fatorial

 for(i= n; i > 0; i--)
    fat= fat * i; //calcula fatorial

 printf("\n O fatorial de %d e: %d", n, fat);
}
```

Com função com retorno:

```
#include <stdio.h>
#include <conio.h> //biblioteca do getch();

// Declaração de procedimentos e funções
int fatorial(int n); //declaração da função com retorno - cabeçalho

// Programa principal
int main(void)
{
 int n, fat;  //n = número informado pelo usuário   fat = valor do fatorial

 printf("\n Digite um numero: ");
 scanf("%d", &n);

 fat= fatorial(n);//o retorno da função é armazenado em fat
```

```
 printf("\n O fatorial de %d e: %d", n, fat);
 getch();   //parada de tela
}

//-------------------------
// Calcula Fatorial
// Entrada.: número inteiro lido
// Objetivo: calcular o fatorial do numero de entrada
// Saída...: fatorial (tipo inteiro)
//-------------------------
int fatorial(int n) //função com retorno inteiro
{
 int i, fat= 1;   //i = controle da repetição for   fat = valor do fatorial

 for(i= n; i > 0; i--)
    fat= fat * i; //calcula fatorial

 return(fat);     //retorno do tipo inteiro
}
```

Nos exemplos apresentados (**função sem e com retorno**) o programa inicia sua execução na função main (perceba que main também é uma função, só que é do próprio C!) e quando encontra o nome da função, desvia a execução para o bloco de comandos da função. Após encerrar este bloco, retorna para main, fazendo as instruções seguintes e encerrando o programa.

Uma **função sem retorno é do tipo void** (sem valor para retorno). É importante observar que, antes da função main, foi feita a declaração do protótipo da função, que consiste em colocar seu **cabeçalho** seguido de ponto e vírgula. Isto é feito para informar ao compilador a existência do código da função após main. Para evitar a declaração do protótipo (cabeçalho), a função poderia ser implementada antes da main.

Portanto, lembre-se: *sempre que implementarmos o código de funções após main, deveremos fazer a declaração de seu protótipo (cabeçalho) antes de main*!

9.3 Escopo de uma variável (região de validade), tipos de parâmetros e retorno de funções

Variáveis Globais (Estáticas): são declaradas fora das funções e são reconhecidas em todo o programa (no main e nos demais subprogramas).

Variáveis Locais: são declaradas no bloco da função e são válidas apenas na função. Tais variáveis permanecem na memória somente durante a execução da função, deixando de existir quando a função é encerrada.

O exemplo a seguir ilustra o uso de variáveis locais, declaradas na própria função.

Exemplo 2: Faça um programa que leia três valores reais que representam a base maior, base menor e altura de um trapézio, calcule e informe a área do trapézio.

Sem função:

```
#include <stdio.h>
#include <conio.h>
int main (void)
{
float bmai, bmen, h, area;
printf("\n Digite a base maior: "); scanf("%f", &bmai);
printf("\n Digite a base menor: "); scanf("%f", &bmen);
printf("\n Digite altura do trapézio: "); scanf("%f", &h);
area= (bmai + bmen)/2 * h;
printf("\n Área do trapezio:5.2f", area);
getch();
}
```

Com função sem retorno:

```
#include <stdio.h>
#include <conio.h>
void calculo (float ma, float me, float altura); //cabeçalho da função
```

```
int main (void)
{
float bmai, bmen, h;
printf("\n Digite a base maior: "); scanf("%f", &bmai);
printf("\n Digite a base menor: "); scanf("%f", &bmen);
printf("\n Digite altura do trapézio: "); scanf("%f", &h);
calculo(bmai, bmen, h);
}

void calculo (float ma, float me, float altura)
{
float area;
area= (ma + me)/2 * altura;
printf("\n Área do trapezio:5.2f", area);
getch();
}
```

A função “calculo” foi implementa com 4 **variáveis locais**:

- ma, me, altura (declaradas entre parênteses) são parâmetros da função.
- area: declarada no corpo da função não é parâmetro.

Como a função “calculo” **não precisa retornar** nada para quem a chamou (no caso, a main), então seu tipo é **void**.

Ao ser acionada uma função com parâmetros, devemos informar os valores, entre parênteses e estes ficarão armazenados nas variáveis locais declaradas dentro dos parênteses, na mesma ordem em que são colocados na chamada da função.

No exemplo temos a variável “bmai” passando seu valor para o parâmetro “ma”, a variável “bmen” passando seu valor para o parâmetro “me” e a variável h passando seu valor para o parâmetro “altura”. Portanto, temos uma **passagem de parâmetros por valor**, variáveis locais recebem os valores enviados na chamada da função.

Assim, existirá uma comunicação de informações entre main e a função “calculo” que é a passagem destes valores, lidos em main e sem os quais a função não poderia efetuar o cálculo. Esta comunicação se dá somente no **sentido de main para função (e não vice-versa)** e, reforço... é conhecida como **passagem de parâmetros por valor**.

No exemplo 3, a função nada retorna, visto que ela mesma informa o resultado do cálculo. No entanto, poderíamos entender que seria melhor se a função retornasse a área calculada para quem a acionou. O exemplo abaixo apresenta uma forma de resolução onde a área calculada é o retorno da função.

Exemplo 3: resolução com retorno da função

```
#include <stdio.h>
#include <conio.h>
float calculo (float ma, float me, float altura); //cabeçalho da função que tem retorno do tipo float

int main (void)
{
float bmai, bmen, h, a;
printf("\n Digite a base maior: "); scanf("%f", &bmai);
printf("\n Digite a base menor: "); scanf("%f", &bmen);
printf("\n Digite altura do trapézio: "); scanf("%f", &h);
a= calculo(bmai, bmen, h);
printf("\n Área do trapezio:5.2f", a);
getch();
}

float calculo (float ma, float me, float altura)
{
float area;
area= (ma+me)/2 * altura;
return(area)
}
```

Observe a utilização do comando **return** antes de encerrar a função. Através deste comando estamos informando qual o retorno desta função, no caso o valor da variável

local “area”, que por isso deve ser do **mesmo tipo da função** (no exemplo, ambos são do tipo float).

Outra forma passarmos valores para “fora” de uma função é através dos parâmetros. Podemos declarar parâmetros de forma que eles não só recebam um valor de quem aciona a função, mas ao mesmo tempo retornem um valor para as variáveis que correspondem os parâmetros. Esta forma de passagem de parâmetros é chamada de **passagem por referência** e consiste em informarmos o endereço da variável no acionamento da função. Por sua vez, o parâmetro correspondente deverá ser declarado como **tipo ponteiro (identificado pelos símbolos & e *)**, para poder armazenar este endereço. Vamos ao exemplo:

Exemplo 4: Faça um programa que leia 3 valores inteiros (a, b e c) e escreva-os ordenados, de forma que em “a” esteja o menor valor e em “c” o maior.

```
#include <stdio.h>
int main (void)
{
int a, b, c, aux;
printf("\n Informe os três inteiros: "); scanf("%d %d %d", &a, &b, &c);
if (a > b){
        aux= a;
        a= b;
        b= aux; }
if (a > c){
        aux= a;
        a= c;
        c=aux; }
if (b > c){
        aux= b;
        b=c;
        c= aux; }
printf("\n a=%d b=%d c=%d", a, b, c);
}
```

Percebe-se que a cada vez que o teste entre as variáveis for verdadeiro será realizado um mesmo bloco de instruções, que troca as variáveis envolvidas no teste. Este trecho de código pode ser implementado em uma função que receba os valores e devolva-os trocados, como é mostrado abaixo, realizando uma **passagem de parâmetros por referência**.

```
#include <stdio.h>
void troca(int *x, int *y);

int main (void)
{
int a, b, c;
printf("\n Informe os três inteiros: "); scanf("%d %d %d", &a, &b, &c);
if (a > b)
        troca(&a, &b);
if (a > c)
        troca(&a, &c);
if (b > c)
        troca(&b, &c);
printf("\n a=%d b=%d c=%d", a, b, c);
}

void troca (int *x, int *y) //x e y são variáveis ponteiros(referências para endereços de memória) para inteiros
{
int aux;
aux= *x;
*x= *y;
*y= aux;
}
```

Agora vamos supor que você queira passar um registro para uma função!

Você terá duas opções:

1. enviar para a função um elemento do registro ou;
2. enviar para a função todos os elementos.

Observe nos exemplos a seguir que a maior diferença entre eles é que o primeiro utiliza passagem por referência (*) de um registro. O segundo, como envia todo o vetor de registro para a função, já pressupõe a referência (indicando o endereço do primeiro registro do vetor para que se tenha acesso a todos os elementos).

Vamos ver um exemplo para a opção 1:

```
#include <stdio.h>
#include <string.h>
#include <conio.h>

#define N_ESTADOS    3      // numero máximo de estados
#define N_CANDIDATOS 5   // numero máximo de candidatos

typedef struct {       // registro para um candidato
     char nome [ 30 ];
     int  votos[ N_ESTADOS ];
} REG_CANDIDATO;

void entrada_dados( REG_CANDIDATO *cand, int *tv )
{
   int j, v; // j = indice

   printf( "\n\n Nome : " );
   fflush( stdin ); // limpa buffer do teclado, funciona junto com entrada de dados
   gets( cand->nome );

   for( j = 0; j < N_ESTADOS; j++ ){
       printf( "\n Votos: " );
       fflush( stdin );
       scanf( "%d", &cand->votos[ j ] );
       *tv = *tv + cand->votos[ j ];
   }
}

void imprime_dados( char nome[ 30 ], int tv )
{
```

```
  printf( "\n Nome do candidato: %s", nome);
  printf( "\n Total de votos   : %d", tv);
}

int main( void )
{
   REG_CANDIDATO candidato[ N_CANDIDATOS ]; // vetor de registros
   int i, j, total_votos;              // i, j = indice, total_votos = votos de um candidato

   for( i = 0; i < N_CANDIDATOS; i++ ){
       total_votos = 0;
       entrada_dados( &candidato[ i ], &total_votos );
       imprime_dados( candidato[ i ].nome, total_votos);
    }
    getch( ); // parada na tela
}
```

Vamos ver um exemplo para a opção 2:

```
#include <stdio.h>
#include <string.h>
#include <conio.h>

#define N_ESTADOS    3      // numero máximo de estados
#define N_CANDIDATOS 5   // numero máximo de candidatos

typedef struct {       // registro para um candidato
     char nome [ 30 ];
     int  votos[ N_ESTADOS ];
} REG_CANDIDATO;

void entrada_dados( REG_CANDIDATO cand[] ){ // leitura dos dados de entrada
 int i, j;

   for(i= 0; i < N_CANDIDATOS; i++){
```

```
        printf("\n Nome: ");
        fflush(stdin);
        gets(cand[i].nome);
        for(j= 0; j < N_ESTADOS; j++){
           printf("\n Votos: ");
           fflush(stdin);
           scanf("%d", cand[i].votos[j]);
        }
   }
}

void imprime_dados ( REG_CANDIDATO cand[] ){
 int i, j;

    for(i= 0; i < N_CANDIDATOS; i++){
        printf("\n Nome: %s", cand[i].nome);
        for(j= 0; j < N_ESTADOS; j++){
           printf("\n Votos: ");
           fflush(stdin);
           scanf("%d", cand[i].votos[j]);
        }
   }
}

int main( void )
{
   REG_CANDIDATO candidato[ N_CANDIDATOS ]; // vetor de registros

   entrada_dados( candidato ); // leitura dos dados de entrada, passagem de todo vetor de
registro
   imprime_dados( candidato ); // impressão de todos os dado, passagem de todo vetor de
registro

    getch( ); // parada na tela
    printf( "\n\n" );
}
```

9.4 Inclusão de arquivos de cabeçalho com funções criadas pelo usuário

Podemos criar um arquivo cabeçalho (***.h**) com as funções que acharmos apropriadas, geralmente aquelas que são muito utilizadas. Abaixo, apresenta-se o mesmo exemplo utilizado para demonstrar passagem de parâmetros por referência, visto anteriormente. Agora, porém, a função que faz a troca entre os valores está implementada em um arquivo externo, chamado **uteis.h**. Veja:

```
#include <stdio.h>
#include <conio.h>
#include "d:\tc\bin\A09_01\ Ex_incl\uteis.h"

int main (void)
{
int a, b, c;
printf("\n Informe os três inteiros: "); scanf("%d %d %d", &a, &b, &c);
if (a > b)
        troca(&a, &b);
if (a > c)
        troca(&a, &c);
if (b > c)
        troca(&b, &c);
printf("\n a=%d b=%d c=%d", a, b, c);
getch();
}

//arquivo .h que possui a função para trocar dois inteiros
void troca(int *x, int *y); //cabeçalho da função
// desenvolvimento da função
void troca(int *x, int *y)
{
 int aux;
aux= *x;
*x= *y;
*y= aux;
}
```

Observação: quando delimitamos o arquivo cabeçalho < > estamos direcionando sua busca na pasta padrão de inclusão (a pasta include), quando utilizamos " " significa que a pasta é corrente, ou podemos especificar um caminho.

9.5 EXERCÍCIOS PROPOSTOS

(Exercícios identificados com (*) estão resolvidos ao final do capítulo)

1. Faça um programa para ler valores inteiros até ser digitado o valor -1 (tarefa 1) e fornecer o fatorial de cada valor lido (tarefa2).

2. Faça um programa que verifica quais dentre os 1000 primeiros números naturais são número perfeito.
   Número perfeito é todo número que, somando seus divisores, esta soma resulta nele mesmo. Utilize uma função para a verificação de cada um dos números. A função main deve informar o resultado. Ex: 6 é divisível por 1 6 é divisível por 2 6 é divisível por 3 1+2+3 = 6 é quadrado perfeito.

3. Faça um programa que lê os seguintes dados relativos a 30 alunos de uma escola de futebol: altura; data de nascimento (tarefa de main). O programa deverá fornecer como saída: a idade média dos alunos (tarefa2); a altura média dos alunos (tarefa 3).

4. A prefeitura de uma cidade fez uma pesquisa entre seus habitantes, coletando dados sobre o salário e o número de filhos. A prefeitura deseja saber:
   • média do salário da população (utilize 1 função);
   • média do número de filhos ((utilize 1 função);
   • maior salário (utilize 1 função);
   • percentual de pessoas com salário até R$420,00 (utilize 1 função).
   Encerrar a leitura dos dados com salário negativo.

5. Faça um programa onde main imprima na tela os ‘n’ primeiros números primos, onde ‘n’ será fornecido pelo usuário. Utilize uma função para verificar se o número é primo.

6. Dado n e p inteiros, n, p >= 0, calcular as combinações de n elementos p a p, isto é: n! / (p! * (n-p)!). Utilize funções.

7. (*) A sequência de Fibonacci é a seguinte: 1, 1, 2, 3, 5, 8, 13, 21, ... os dois primeiros termos são iguais a 1. Cada termo seguinte é igual à soma dos dois anteriores. Escreva um programa onde main solicita ao usuário o número do termo e o cálculo dos demais termos seja apresentado por uma função.

8. Escreva um programa em que main solicite ao usuário três números inteiros a, b, e c onde a é maior que 1. Uma outra função deve somar todos os inteiros entre b e c que sejam divisíveis por a. Main deve informar a soma.

9.6 EXERCÍCIO RESOLVIDO

Exercício 7:

C

```
/* Definição das bibliotecas */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>

/* Cabeçalhos de funções/procedimentos */
void fibonacci(int termo);

/* Programa principal */
int main(void)
{
  int termo, i, op; //op = opção continuar o programa ou finalizar

  do{
```

```
    printf("\n Digite o termo: ");
    scanf("%d", &termo);

    fibonacci(termo);

    getch(); //parada da tela
    printf("\n Digite [c] para continuar: ");
    op= toupper(getche()); //lê um caracter e converte a entrada para maiúsculo, não é necessário digitar <enter>
  }while(op == 'C');
}

/*-----------------------
| Função fibonacci
| Entrada: valor lido da quantidade de termos
| Saída  : impressão em tela do resultado da série
+----------------------*/
void fibonacci(int termo)
{
   int i, a= 1, b= 1, c;

   printf("\n Sequencia de Fibonacci: ");
   printf("%d  %d ", a, b);

   for(i = 0; i < termo-2; i++){
            c= a + b;
            printf(" %d ", c);
            a= b;
            b= c;
        }
}
```

C

RESOLUÇÃO UTILIZANDO ALGORITMO RECURSIVO

```
/* Definição das bibliotecas */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
```

```
#include <ctype.h>

/* Cabeçalhos de funções/procedimentos */
int fibonacci(int n);

/* Programa principal */
int main(void)
{
  int termo, i, op; //op = opção continuar o programa ou finalizar

  do{
     printf("\n Digite o termo: ");
     scanf("%d", &termo);

     printf("\n Sequencia de Fibonacci: ");
     for(i = 0; i < termo; i++)
          printf(" %d ", fibonacci(i + 1));

     getch(); //parada da tela

     printf("\n Digite [c] para continuar: ");
     op= toupper(getche()); //lê um caracter e converte a entrada para maiúsculo, não é necessário digitar <enter>
  }while(op == 'C');
}

/*-----------------------
| Função fibonacci
| Entrada: valor lido
| Saída   : resultado da série
+----------------------*/
int fibonacci(int n)
{
   if(n == 1 || n == 2)
      return 1;
   else
      return fibonacci(n - 1) + fibonacci(n - 2); //recursividade (a função chama ela própria!)
}
```

Referências biliográficas:

Ziviani, Nivio. Projeto de Algoritmos em C e Java. 2ª ed. São Paulo: Thomson Pioneira, 2004

Feofiloff, Paulo. Algoritmos em Linguagem em C. 1ª ed. Rio de Janeiro: Campus, 2008

Fertig, Cristina; Medina, Marco. Algoritmos e Programação Teoria e Prática. 1ª ed. São Paulo: Novatec, 2005

Oliveira, Álvaro; Boratti, Isaias. Introdução a Programação de Algoritmos. 3ª ed. Florianópolis: Visual Books, 2007

Garcia, Guto; Lopes, Anita. Introdução à Programação: 500 Algoritmos Resolvidos. 1ª ed. Rio de Janeiro: Campus, 2002

Lux, Beatriz; Furtado, João Carlos. Apostila de Algoritmos e Programação. Departamento de Computação, UNISC, 2010