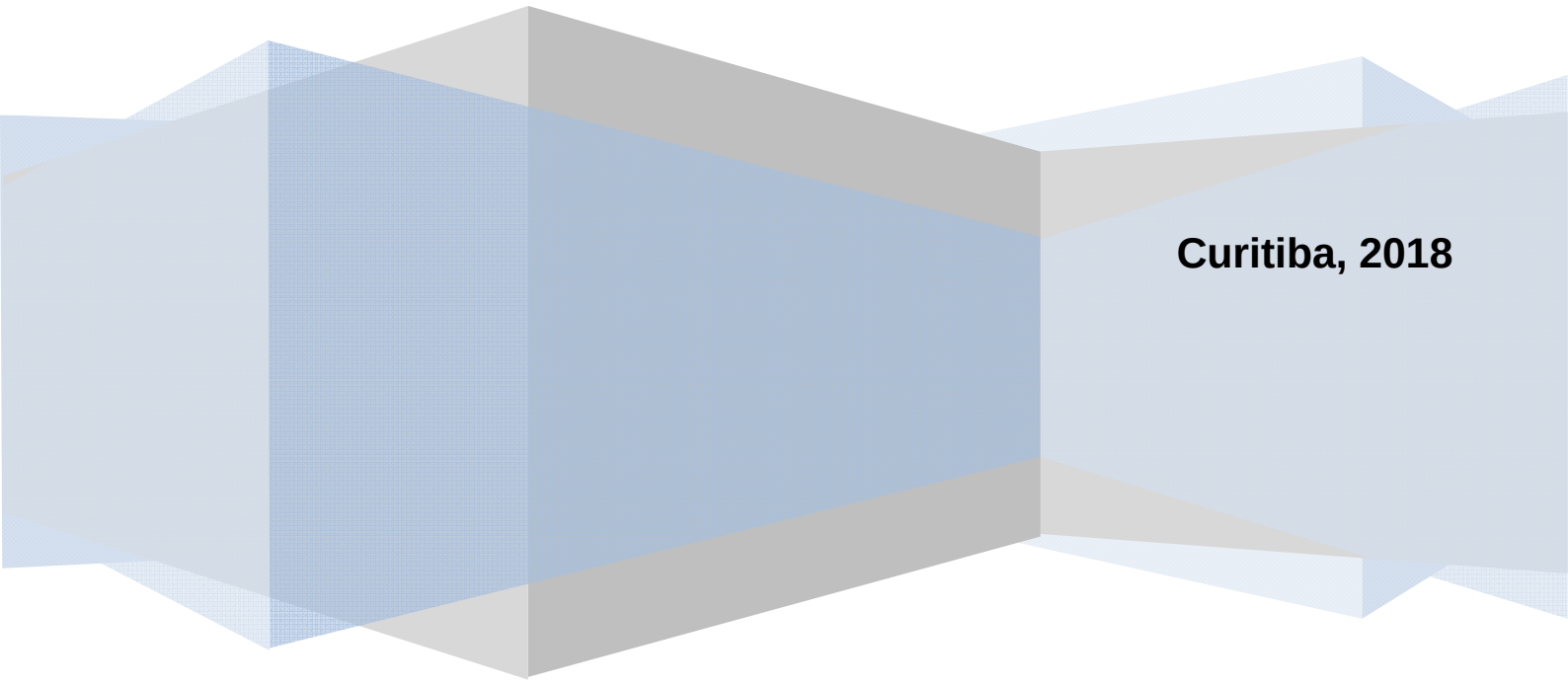
Introdução à programação de computadores

Com exemplos em HTML e PHP

Cicero Aparecido Bezerra



Curitiba, 2018

SUMÁRIO

1 SOBRE O AUTOR 3
2 APRESENTAÇÃO 4
3 ESTRUTURAS BÁSICAS DE PROCESSAMENTO DE DADOS 5
4 VARIÁVEIS, CONSTANTES E COMANDO DE ATRIBUIÇÃO 9
5 COMANDOS DE ENTRADA E SAÍDA DE DADOS 15
6 OPERADORES MATEMÁTICOS 28
7 COMANDOS DE DECISÃO 37
8 COMANDOS DE REPETIÇÃO 49
9 FUNÇÕES 60
10 CONSIDERAÇÕES FINAIS 73

1 SOBRE O AUTOR

Possui graduação em Informática pela Universidade do Vale do Rio dos Sinos (1992), mestrado em Engenharia de Produção pela Universidade Federal de Santa Catarina (2001), doutorado em Engenharia de Produção pela Universidade Federal de Santa Catarina (2007) e estágio pós-doutoral em Gestão Estratégica da Informação e do Conhecimento pela Pontifícia Universidade Católica do Paraná (2012). Atualmente é professor Associado nível II da Universidade Federal do Paraná. Tem experiência em profissional em desenvolvimento, implantação e gestão de Sistemas de Informação e, Processos de Produção. Enquanto docente, leciona disciplinas alinhadas ao desenvolvimento de Sistemas de Informação e Análise de Dados. Como pesquisador, tem voltado sua atenção aos Métodos e Técnicas de Análise de Dados, aplicados à Gestão do Conhecimento e Inovação.

2 APRESENTAÇÃO

O material ao qual vocês estão tendo acesso apresenta, passo a passo, uma maneira para o desenvolvimento de programas computacionais. Foi desenvolvido com a expectativa que sua leitura sequencial conduza, naturalmente, à construção de programas de computador, a partir de uma linguagem coloquial, próxima ao leitor. Além disto, ainda que os exemplos tenham sido implementados em PHP, a demonstração algorítmica permite que, com um mínimo de esforço, os programas possam ser codificados na maior parte das linguagens de computador.

A primeira parte consiste em uma visão geral das estruturas básicas de processamento de dados. A seguir, a ênfase volta-se aos conceitos, atribuições e aplicações de variáveis. Logo na sequência, os comandos de entrada e saída de dados são demonstrados. A próxima seção traz os operadores matemáticos mais empregados. Em seguida, são abordados as estruturas de controle condicionais: decisão e repetição. A próxima seção demonstra o emprego de funções, cujo intuito é modularizar os programas, tornando-os mais econômicos em termos de linhas de código. Finalmente, são colocadas algumas considerações e bibliografias que você, leitor, deve atentar caso desejar se aprofundar em programação de computadores.

Espero que este material possa ser útil na compreensão básica dos conceitos e comandos necessários à programação de computadores.

Bons estudos...

3 ESTRUTURAS BÁSICAS DE PROCESSAMENTO DE DADOS

Gerenciar informações requer do gestor mais do que a habilidade de planejar, organizar, coordenar, controlar e comandar equipes e infra estrutura. É importante que este profissional saiba como as informações são geradas. A partir daí, alinhando esta habilidade à tecnologia de informação, o gestor da informação poderá automatizar estes processos, ganhando agilidade e reduzindo custos para a obtenção da informação correta no prazo adequado.

Sabemos que as informações são geradas pelo processamento de dados. Este processamento é realizado a partir de algumas regras e estruturas básicas. Conhecendo-as, o gestor da informação poderá traduzi-las para a grande maioria das linguagens de programação. De modo geral, é possível afirmar que a estrutura lógica básica para o processamento de dados apresenta, basicamente, as seguintes etapas:

```
1. Início;
2. Declaração de variáveis;
3. Entrada de dados;
4. Processamento de dados;
5. Saída de dados;
6. Fim.
```

Vamos compreender cada uma das etapas:

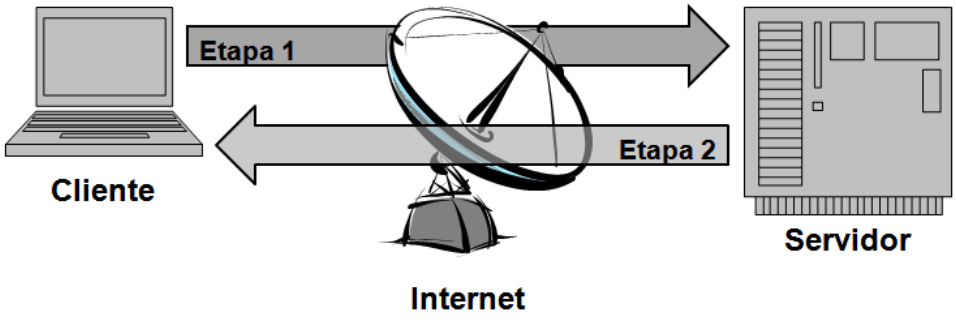
1. Início: todo processamento de dados, deve ter seu início muito bem delimitado. Não pode haver dúvidas sobre em que ponto será iniciado.
2. Declaração de variáveis: podemos entender “variáveis” como sendo os elementos que irão armazenar os dados e, após o processamento, as próprias informações. Antes do processamento propriamente dito, as variáveis devem ser formalmente identificadas – o que significa dizer que devemos saber, de antemão, quais são os elementos que serão processados.
3. Entrada de dados: os dados, que serão transformados em informações, devem ser originados de algum ponto. A etapa de `entrada de dados` indica como estes dados serão obtidos.
4. Processamento de dados: esta é a etapa onde os dados “sofrerão” algumas operações matemáticas ou lógicas até serem transformados em informações.
5. Saída de dados: uma vez obtidas as informações, estas devem ser mostradas a quem de direito.
6. Fim: assim como o processamento tem seu início bem definido, também o final deve ser formalmente identificado.

Combinar estes elementos no intuito de permitir sua execução por computadores (garantindo rapidez) é o que chamamos de desenvolvimento de algoritmos. Um algoritmo pode ser entendido como um conjunto de instruções, comunicadas em uma linguagem natural e que, quando executadas em determinada sequência, atingem um objetivo previamente estabelecido. Que objetivo é este? A geração de informações! Processar dados nada mais é do que traduzir um algoritmo para uma linguagem de programação e submeter os dados a este programa.

Este capítulo irá, portanto, guia-los na construção lógica de algoritmos traduzindo-os em uma linguagem de programação, para que possam ser testados, além de familiariza-los com o desenvolvimento de programas computacionais.

Antes de mais nada, vamos esclarecer que os programas serão testados em uma linguagem de programação, chamada PHP (abreviação de *Personal Home Page*). Escolheu-se esta linguagem pela sua popularidade no desenvolvimento de aplicações Web. Uma aplicação Web é aquela em que existe uma comunicação de dados entre um computador cliente (que, geralmente, informa os dados a serem processados) e um computador servidor (que recebe os dados, processa e envia a informação ao cliente), através da Internet, conforme mostrado na Figura 1:

Figura 1 – Comunicação cliente servidor



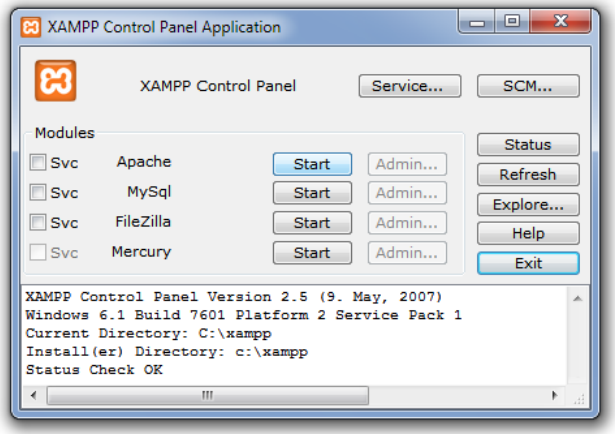
A Figura 1 mostra, na Etapa 1, os dados sendo transmitidos ao servidor que, após tê-los processado, os retorna ao cliente (Etapa 2). Onde entra o PHP? Os programas escritos em PHP ficam residentes no servidor e são disparados por páginas Web nos clientes (escritas em HTML). Trata-se da abreviação de *HyperText Markup Language*, que nada mais é do que uma linguagem empregada para a produção de páginas Web, interpretadas por navegadores Web. Basicamente, esta linguagem funciona com etiquetas (tags) que determina a formatação (ou a inserção) de algum elemento a ser exibido no navegador.

Tudo isto significa que, para testarmos nossos programas, iremos utilizar uma página HTML onde os dados serão fornecidos e um programa PHP (chamado de script) onde serão processados. Mas vocês devem estar se perguntando: onde iremos salvar os programas PHP que, anteriormente, foi explicado que estariam no servidor? Iremos utilizar um

programa que irá fazer com que seu computador pessoal seja, ao mesmo tempo, cliente e servidor! Existem vários programas que fazem esta tarefa – qualquer um pode ser empregado. Em nosso caso, sugerimos o XAMPP, por se tratar de um servidor desenvolvido em software livre, independente de plataforma, rodando em sistemas Microsoft Windows, GNU/Linux, Solaris, e MacOS X. Sua instalação é muito simples: basta fazer o download, executar o arquivo baixado e confirmar as opções solicitadas.

Para que as aplicações desenvolvidas no decorrer deste capítulo possam ser testadas, é necessário que seu computador esteja com serviços de servidor ativos. Para isto, é necessário clicar no Painel de Controle do XAMPP e, no serviço “Apache”, clicar no botão “Start”, conforme mostrado na Figura 2:

Figura 2 – Iniciando serviço de servidor



Se tudo transcorrer normalmente, irá aparecer a palavra “Running” ao lado do serviço “Apache”. Isto indica que o servidor Apache já está rodando em seu computador.

Além disto, é necessário que todos os programas PHP e páginas HTML estejam salvos na pasta C:\xampp\htdocs – iremos lembra-los disto mais adiante. Ao término dos exercícios, recomenda-se acessar o XAMPP e clicar no botão “Stop” ao lado do serviço “Apache” e, em seguida, no botão “Exit”.

Exercícios resolvidos

1. O que é PHP?

   Linguagem de programação empregada em aplicações cliente-servidor.

2. Onde residem os scripts PHP?

   Nos servidores.

3. O que é *HyperText Markup Language*?

   Linguagem de marcação para exibição de páginas Web em navegadores.

4. Qual a relação entre páginas HTML e scripts PHP?

   As páginas HTML recebem dados que são enviados aos scripts PHP para que sejam processados.

4 VARIÁVEIS, CONSTANTES E COMANDO DE ATRIBUIÇÃO

Primeiramente devemos identificar os elementos que irão comportar os dados que serão processados. Estes elementos são chamados de *variáveis*. O fato de serem conhecidos como *variáveis* indica que podem assumir vários valores durante o processamento. Podemos imaginar uma variável como sendo uma caixa que armazena determinado conteúdo (dado) e que, só pode armazenar um e somente um dado naquele momento – caso seja necessário que esta variável armazene outro dado, o atual desaparece para dar lugar ao novo dado.

Seguindo a estrutura lógica para processamento de dados apresentada anteriormente, o primeiro passo é identificar quais variáveis são necessárias para que a tarefa seja cumprida. Em algoritmos, esta etapa é chamada de *declaração de variáveis*. Consiste, basicamente, em nomear as variáveis utilizadas. Existem algumas regras a serem observadas quando necessitarmos informar quais variáveis serão necessárias para processar o que desejamos:

1. Cada variável terá um nome único e, obviamente, não pode ser repetido para outra variável.
2. Recomenda-se que o nome da variável seja iniciado por uma letra e não contenha caracteres especiais, tais como acentos (~ ‘ ^) e espaços em branco.
3. É importante, mas não obrigatório, que o nome dado à variável seja curto e remeta à finalidade da mesma.

Vejamos alguns exemplos de nomes corretos e errados para variáveis:

Quadro 1 – Nomes de variáveis

Exemplo correto	Exemplo errado
numero	número
nome1	1nome
salario	salário
cidadeNatal	cidade natal

Em PHP, servem as mesmas regras, com um pequeno, mas importante, detalhe: as variáveis **devem** iniciar com o sinal de $:

Quadro 2 – Nomes de variáveis em PHP

Exemplo em algoritmo	Exemplo em PHP
numero	$numero
nome1	$nome1
salario	$salario
cidadeNatal	$cidadeNatal

Uma variável armazena valores de diversas naturezas:

- Numérica: representa um valor numérico, indicando, portanto, que é passível de operações matemáticas. Por exemplo: 3; 45,67 (em algumas linguagens de programação, as casas decimais são representadas por ponto – 45.67). Algumas linguagens de programação requerem que variáveis numéricas sejam especificadas, sendo as mais comuns as do tipo "Real" que permitem números representados com casas decimais (por exemplo: 45,67; 5,0; -1,46) e "Inteiro" que, evidentemente, somente permitem armazenar números inteiros (4; -23).
- Alfanumérica: uma variável pode representar um conjunto de caracteres sendo eles letras, símbolos e/ou números – o que indica que não poderemos realizar operações matemáticas com esta variável. Tanto em um algoritmo, quanto em um programa de computador, os valores atribuídos a estas variáveis são representados entre aspas, por exemplo: "sim", "masculino", "12/08/2013"; "22,34".
- Lógica: este tipo de variável armazena apenas o resultado de um teste condicional que pode ser somente "verdadeiro" ou "falso".

Em processamento de dados existe ainda um outro elemento que representa valor, porém, ao contrário das variáveis, este valor permanece inalterado durante a execução do programa – por esta razão, este elemento é chamado de *constante*. A rigor, não é necessário que uma constante seja declarada, mas é importante que saibamos que as constantes apresentam as mesmas naturezas das variáveis.

As variáveis podem armazenar valores através de comandos de atribuição, observando uma única, porém importante, regra: **sempre representamos o elemento que vai receber o valor** (no caso, a variável) **antes do valor a ser recebido**, conforme exemplos a seguir:

Quadro 3 – Atribuições

Exemplo correto	Exemplo errado

`numero ← 1;`
`nome1 ← "João";`
`salario ← 1234.56;`
`cidadeNatal ← "São Paulo";`
	`1 → numero;`
	`"João" → nome1;`
	`1234.56 → salario;`
	`"São Paulo" → cidadeNatal;`

Percebam que em algoritmos, empregamos uma seta para representar a atribuição de determinado valor para uma variável (`numero ← 1;`). Já em PHP, para o mesmo comando, utilizamos o sinal de igual (`numero = 1;`). Notem que cada variável (`numero`, `nome1`, `salario` e `cidadeNatal`) está recebendo um valor constante (respectivamente, `1`,

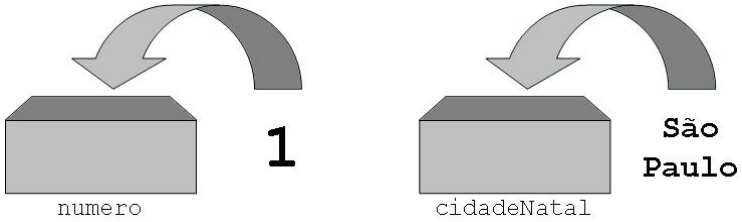
"João", 1234.56 e "São Paulo") que permanecerá o mesmo durante a execução do programa. Outro detalhe fundamental: se atribuirmos um valor alfabético para uma variável, este valor **deve** estar entre aspas. Os exemplos a seguir mostram estas situações:

Quadro 4 – Atribuições em PHP

Exemplo em algoritmo	Exemplo em PHP
numero ← 1; nome1 ← "João"; salário ← 1234,56; cidadeNatal ← "São Paulo";	$numero = 1; $nome1 = "João"; $salário = 1234,56; $cidadeNatal = "São Paulo";

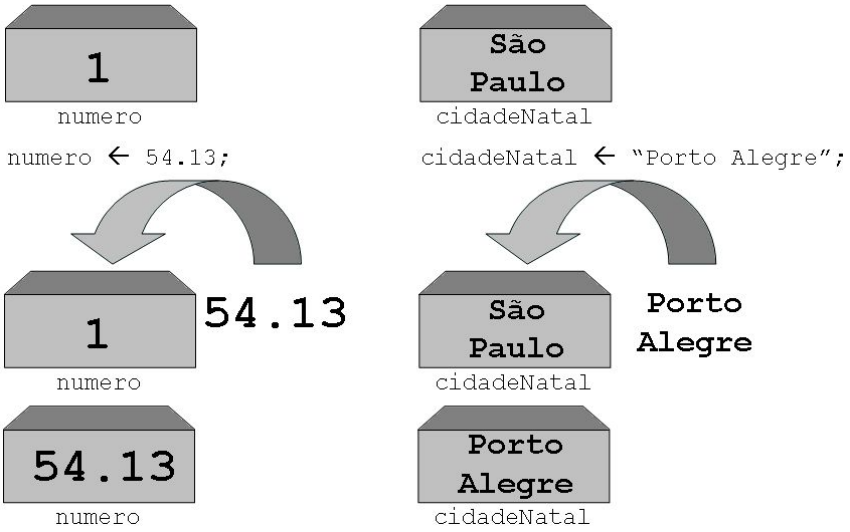
Temos que entender a atribuição como uma ação de estarmos "jogando" algum valor para uma caixa (variável), como na Figura 3:

Figura 3 – Atribuição



Percebam que se a "caixa" (variável) já contiver um valor prévio e atribuirmos outro, perderemos o anterior, conforme Figura 4:

Figura 4 – Atribuindo novos valores



Vale atentarmos para um detalhe: anteriormente vimos que uma variável não pode ser representada utilizando caracteres especiais ou espaços em branco. Porém, nos exemplos, a variável cidadeNatal recebe um valor que contém um acento e um espaço em branco (São Paulo). Como explicar isto? É simples: o **nome** da variável (cidadeNatal), de fato, não pode conter estes elementos, ou seja, não podemos nomear

uma variável empregando tais caracteres, mas nada impede que o **conteúdo** desta variável (`São Paulo`) tenha não somente estes, bem como quaisquer outros caracteres.

É possível ainda atribuirmos o conteúdo de uma variável para o conteúdo de outra variável – isto, de fato, é corriqueiramente utilizado em processamento de dados. Vamos observar os exemplos a seguir com atenção:

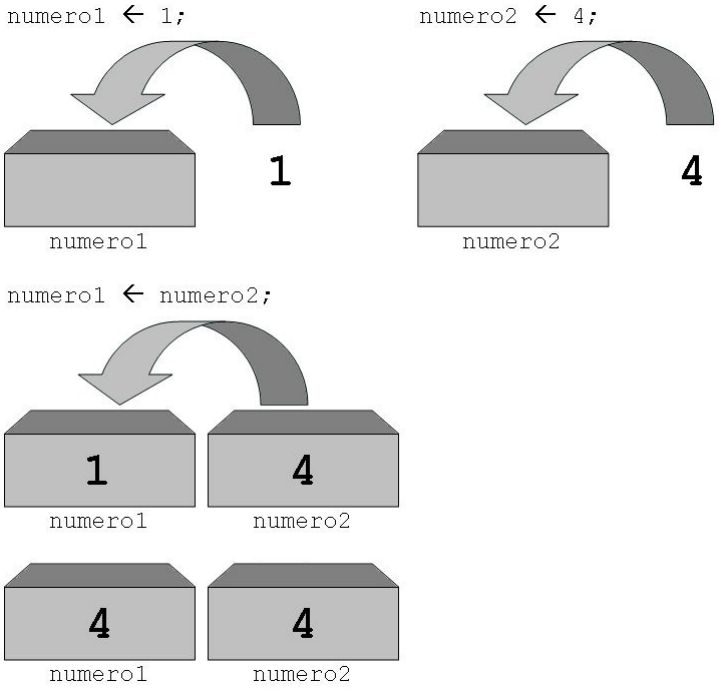
Quadro 5 – Atribuindo conteúdos de variáveis

Exemplo em algoritmo	Exemplo em PHP

`1. numero1 ← 1;`
`2. numero2 ← 4;`
`3. numero1 ← numero2;`
	`$numero1 = 1;`
	`$numero2 = 4;`
	`$numero1 = $numero2;`

Tendo uma vez executado a sequência de instruções, qual será o valor final contido na variável `numero1`? Será `4`, conforme a sequência reproduzida na Figura 5:

Figura 5 – Atribuições de conteúdos de variáveis



Ou seja, sempre que encontrarmos atribuições de variável para variável temos que saber se trata da atribuição do conteúdo de uma variável, para o conteúdo de outra. Mas voltando ao exemplo mostrado na Figura 5... Qual valor terá permanecido na variável `numero2`? Também `4`. **Uma vez que uma variável recebe determinado valor, ela só terá seu conteúdo alterado se receber novo valor** – enquanto isto não ocorrer, ela permanece com o valor atual inalterado.

Exercícios resolvidos

1. Uma das variáveis foi nomeada de forma errada. Qual delas?

   (a) Numero1

   (b) Numero_1

   (c) N1umero

   (d) 1Numero

   (e) Numero

2. Qual será o valor da variável numero1 após a execução dos seguintes comandos:

```
numero1 ← 1;
numero2 ← 3;
aux ← numero1;
numero1 ← aux + numero2;
numero1 ← aux.
```

   (a) 1

   (b) 3

   (c) 4

   (d) 2

   (e) 0

3. Qual será o valor da variável numero1 após a execução dos seguintes comandos:

```
numero1 ← 1;
numero2 ← 3;
aux ← numero1;
numero1 ← aux + numero1.
```

   (a) 1

   (b) 3

   (c) 4

   (d) 2

   (e) 0

4. As variáveis `soma`, `numero1` e `numero2` foram declaradas como alfanuméricas. Qual será o valor armazenado em `soma`, após a execução dos seguintes comandos?

```
numero1 ← "1";
numero2 ← "4";
soma ← numero1 + numero2;
```

(a) "1"

(b) "4"

(c) "5"

(d) "14"

(e) "3"

5. Um destes comandos de atribuição está errado. Qual deles?

(a) Numero1 ← 2;

(b) Numero1 ← "Livro de Programação";

(c) Numero1 ← Numero2;

(d) Numero1 ← Numero1 + Nota;

(e) Numero1 ← Porto Alegre;

5 COMANDOS DE ENTRADA E SAÍDA DE DADOS

Em processamento de dados, algumas vezes temos que permitir que o usuário de nosso programa possa inserir os valores que julgar necessário para as variáveis. Além disto, é comum nos deparamos com a necessidade de apresentar os dados processados – o que conhecemos por informação.

Um comando de entrada de dados significa que o programa de computador ficará aguardando por um valor que será atribuído a uma variável, **vindo externamente do programa** – ao contrário da instrução de atribuição. “Vir externamente” significa que este valor poderá ser obtido do teclado (via digitação), do clique de um mouse sob determinado elemento, de algum dispositivo de leitura ótica (um scanner, por exemplo). Em algoritmos, expressamos esta instrução através do comando `Ler` (o exemplo será apresentado logo a seguir). Já a instrução de saída de dados (representado pelo comando `Mostrar`) indica que naquele ponto do programa, um determinado valor será apresentado em um dispositivo externo, tal como um monitor ou uma impressora.

Vamos supor que desejamos implementar um programa que receba dois números fornecidos pelo usuário, atribua o segundo número ao primeiro e mostre este resultado. Para isto, vamos retomar a estrutura lógica básica de um programa, apresentada anteriormente, para desenvolver o algoritmo que servirá de base para nosso programa:

Quadro 6 – Atribuição de variáveis

Estrutura lógica básica de um programa	Algoritmo para atribuição de números
1. Início;	1. Início;
2. Declaração de variáveis;	2. Variáveis:
	2.1. Inteiro: numero1;
	2.2. Real: numero2;
3. Entrada de dados;	3. Ler numero1;
	4. Ler numero2;
4. Processamento de dados;	5. numero1 ← numero2;
5. Saída de dados;	6. Mostrar numero1;
	7. Mostrar numero2;
6. Fim.	8. Fim.

Já estamos diante de um algoritmo plenamente funcional – o que significa que ele já pode ser codificado para uma linguagem de programação. Mas antes disto, vamos atentar para algumas importantes regras no desenvolvimento de algoritmos:

1. Um algoritmo deve ter seu início muito bem determinado (linha 1), bem como seu final (linha 8).

2. Existe uma sequência lógica para a execução do algoritmo – por isto, as linhas estão numeradas e serão executadas exatamente na ordem. A linha 2, por sua vez, apresenta uma subdivisão, indicado pelas linhas 2.1 e 2.2. Esta subdivisão informa que temos uma variável numéricas do tipo `Inteiro`, ou

seja, a variável `numero1` somente irá armazenar números inteiros e outra variável (`numero2`) que irá armazenar números reais.

3. Cada instrução deve ser delimitada por ponto e vírgula (;) – percebam isto nas linhas 1, 2.1; 2.2; 3; 4; 5; 6 e 7. O ponto e vírgula indica que aquela instrução foi finalizada e, portanto, devemos seguir para a próxima. Se não indicarmos o fim da linha de comando, a ideia transmitida será a de que aquele comando ainda não foi finalizado. Percebam que a linha 2, que informa que as variáveis serão declaradas, não foi finalizada, pois é composta de duas outras linhas 2.1 e 2.2, sendo que cada linha é finalizada pelo ponto e vírgula. Além disto, a última linha do algoritmo é finalizada com ponto final (.).

Vamos agora interpretar o algoritmo, linha a linha:

1. Esta linha indica o início do algoritmo.
2. Logo após o início do algoritmo, devemos declarar as variáveis que serão utilizadas no seu transcorrer. Ainda que certas linguagens prescindam da declaração das variáveis (como o PHP) é altamente recomendável que tenhamos em mente quais serão as variáveis envolvidas. Esta linha 2 está subdividida em outras duas linhas, cada uma delas indicando a natureza de um conjunto distinto de variáveis. Se fosse necessária mais alguma variável inteira, por exemplo, separaríamos esta da primeira por uma vírgula (,) e repetiríamos este processo até não ser necessário mais nenhuma variável inteira, encerrando este conjunto com um ponto e vírgula (;).
3. Nesta linha estamos indicando que o algoritmo ficará suspenso até que receba algum valor externo a ele e que será armazenado na variável `numero1`. Podemos interpretar esta instrução como aguardando que o usuário digite algum valor para a variável `numero1`.
4. Da mesma forma como na linha anterior, o algoritmo requer que se obtenha algum valor externo que será armazenado na variável `numero2`.
5. Uma vez tendo obtido os valores, o programa requer que atribuamos o valor que está armazenado na variável `numero2` para a variável `numero1`.
6. Agora, a instrução desta linha mostra o conteúdo atual da variável `numero1`.
7. Da mesma forma, o conteúdo da variável `numero2` é mostrado.
8. Como não há mais nada a ser efetuado, o algoritmo é finalizado.

Percebam que utilizamos uma estrutura lógica bastante básica, apoiada na ideia de `Entrada → Processamento → Saída`. Porém alguns problemas requerem certa flexibilidade quanto ao momento em que os dados darão entrada, serão processados e, enfim, mostrados. Nada impede, por exemplo, que seja necessário mostrar o conteúdo de alguma variável logo após termos entrado com algum valor para ela e, somente então, efetuarmos algum tipo de processamento.

Agora que já compreendemos a lógica empregada, vamos traduzir as respectivas instruções do algoritmo para comandos utilizados na linguagem PHP:

Quadro 7 – Script PHP gerado a partir de um algoritmo

Algoritmo para atribuição de números	Script PHP para atribuição de números
1. Início;	<?php
2. Variáveis:	/* o PHP prescinde da declaração de variáveis */
2.1. Inteiro: numero1;	
2.2. Real: numero2;	
3. Ler numero1;	$numero1 = $_GET["numero1"];
4. Ler numero2;	$numero2 = $_GET["numero2"];
5. numero1 ← numero2;	$numero1 = $numero2;
6. Mostrar numero1;	echo $numero1." ";
7. Mostrar numero2;	echo $numero2;
8. Fim.	?>

Vamos compreender o script PHP, relacionando cada linha do algoritmo com a instrução correspondente:

1. O início de um script PHP é identificado pela tag `<?php`.
2. O PHP prescinde da declaração de variáveis. Isto indica que no momento em que uma variável é referenciada, se ela já não existir, naquele momento é criada e sua natureza será determinada pela natureza do valor que lhe é atribuída. Por isto é necessária muita atenção no momento de identificarmos as variáveis necessárias.
3. Na linha 3 do algoritmo temos a instrução de entrada de dados. O comando PHP correspondente é `$_GET`. Sua sintaxe indica que o valor contido no objeto `numero1` será armazenado na variável `$numero1`. Muita atenção a este detalhe: percebam que a variável `$numero1` começa com símbolo de cifrão ($), enquanto que o objeto `numero1` não deve ser iniciado por este símbolo. Mas o que é o objeto `numero1`? É um elemento de entrada de dados através de um formulário HTML (que será mostrado logo a seguir).
4. Da mesma forma, recebemos o valor da variável `$numero2` de um elemento identificado por `numero2`, vindo de um formulário HTML.

5. Nesta linha, o valor contido na variável `$numero2` é armazenado na variável `$numero1` – consequentemente, eliminando o valor obtido por esta variável através do comando `$numero1 = $_GET[numero1];`.

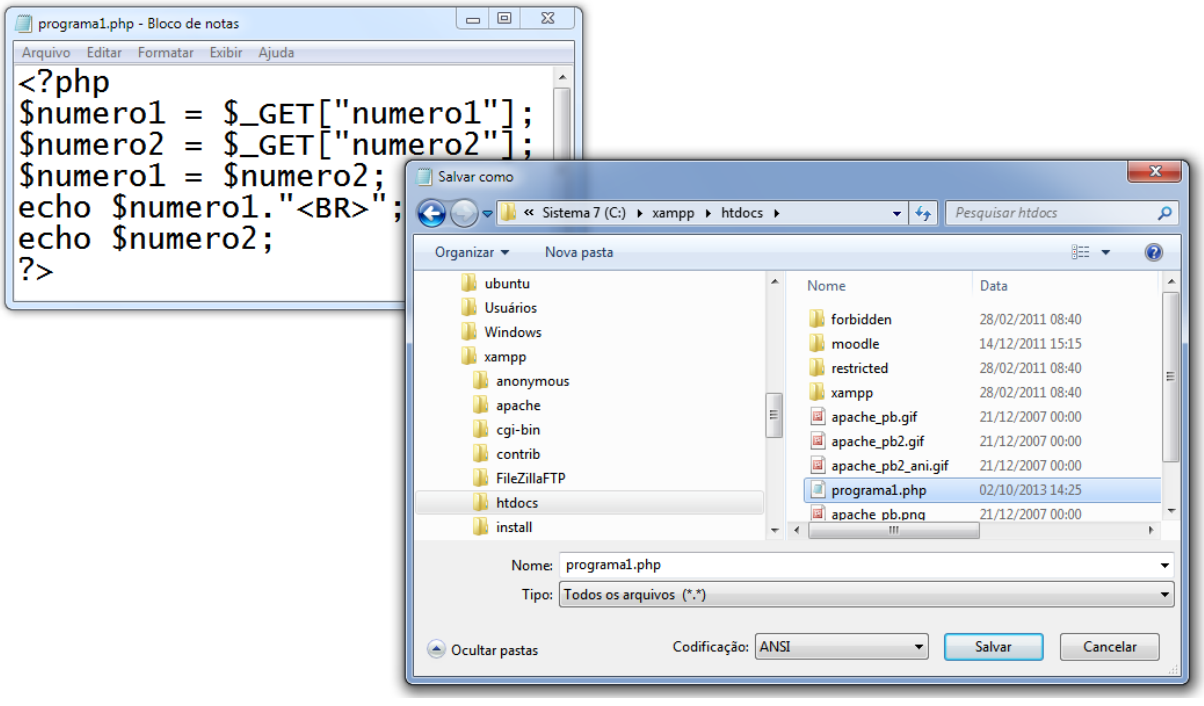
6. O comando PHP que mostra o valor armazenado em uma variável é `echo`. Notem que logo após a variável `$numero1` existe um ponto (.) seguido da instrução `"<BR>"`. O ponto indica que estamos vinculando (o termo técnico é "concatenando") a exibição do conteúdo da variável `$numero1` a uma instrução de quebra de linha (`"<BR>"`), Ou seja, logo após termos mostrado o valor da variável `$numero1` iremos "quebrar a linha" para mostrar o conteúdo da variável `$numero2` na linha seguinte. Se não tivéssemos vinculado o `"<BR>"` ao `echo $numero1`, a linha seguinte iria mostrar o valor da variável `$numero2` imediatamente ao lado da variável `$numero1`.

7. A linha seguinte, tal como a linha anterior, irá mostrar (`echo`) o conteúdo da variável `$numero2`.

8. Finalmente, como não há mais nada a ser processado, o script PHP é encerrado (`?>`).

As instruções do script PHP podem ser digitadas no bloco de notas e devem ser salvas na pasta C:\xampp\htdocs, conforme mostrado na Figura 6:

Figura 6 – Salvando o script PHP



Temos que prestar especial atenção aos seguintes aspectos:

1. O nome do arquivo deve ser acompanhado de `.php`.
2. O arquivo deve ser salvo como tipo “Todos os arquivos (*.*)”.

Apesar de digitado, nosso primeiro programa ainda não pode ser testado. Percebam que as linhas 3 e 4 do algoritmo exigem que sejam obtidos valores para as variáveis `$numero1` e `$numero2` – traduzido pelas instruções `$numero1 = $_GET[numero1]` e `$numero2 = $_GET[numero2]`. Para que possamos informar valores que serão armazenados nas variáveis, iremos implementar uma página Web e a partir dela, digitar os respectivos valores. Esta página Web será desenvolvida em HTML.

O código HTML que cria uma página onde seja possível informar dois valores é mostrado no Quadro 8:

Quadro 8 – Página HTML

```
1   <HTML>
2       <HEAD>
3           <TITLE>1° programa</TITLE>
3       </HEAD>
5       <BODY>
6           <FORM ACTION = “programa1.php”>
7               Informe número 1<INPUT TYPE = “text” NAME = “numero1”><BR>
8               Informe número 2<INPUT TYPE = “text” NAME = “numero2”><BR>
9               <INPUT TYPE = “submit” NAME = “enviar” VALUE = “Enviar”>
10              <INPUT TYPE = “reset” NAME = “limpar” VALUE = “Limpar”>
11          </FORM>
12      </BODY>
13  </HTML>
```

A linguagem HTML funciona por delimitação de instruções, ou seja, abrimos uma instrução, executamos determinada ação e fechamos a instrução. Tecnicamente, chamamos as “instruções” de tags. Vamos compreender este código HTML, linha a linha:

1. `<HTML>`: indica que foi aberta uma instrução que irá informar ao navegador que todas as instruções subsequentes serão do tipo HTML, ou seja, interpretada por navegadores. Lembre-se que, em algum momento esta tag deverá ser fechada!
2. `<HEAD>`: um documento HTML é dividido, basicamente, em duas partes: cabeçalho (`<HEAD>`) e corpo do documento (`<BODY>`). A instrução da linha 2 indica que estamos implementando o cabeçalho do documento – da mesma forma que a tag anterior, esta também deverá ser fechada.
3. `<TITLE>1° programa</TITLE>`: entre outras informações, no cabeçalho informamos o título do documento – que será mostrado na parte superior do

navegador. Assim sendo, abrimos a tag de título (<TITLE>), informamos o título a ser exibido (1° programa) e fechamos a tag (</TITLE>).

4. </HEAD>: como não temos mais nada a ser informado no cabeçalho do documento HTML, encerramos a tag de cabeçalho, aberta na 2ª linha.

5. <BODY>: indica que estamos abrindo a 2ª parte do documento HTML, ou seja, o corpo. O corpo é, basicamente, tudo o que é mostrado na página HTML. Sendo uma tag aberta, também deverá ser fechada posteriormente.

6. <FORM ACTION = "programa1.php">: todas as tags são importantes (mais do que isto, são necessárias), porém esta apresenta fundamental papel em nosso estudo! Vamos explica-la por partes. O termo FORM indica que estaremos abrindo um formulário, ou seja, a página irá conter elementos que possibilitem ao usuário inserir dados (no nosso exemplo, os valores de $numero1 e $numero2). O termo ACTION = indica o nome do programa (script) que irá processar os dados inseridos no formulário – no caso, um programa chamado programa1.php (não por coincidência, o programa anteriormente digitado!). Não podemos esquecer que esta tag deverá ser fechada em algum ponto do documento.

7. Informe o número 1<INPUT TYPE = "text" NAME = "numero1">
: esta linha representa o primeiro elemento inserido no formulário. Na verdade, nesta linha, temos dois elementos: a frase Informe o número 1 (percebam que ela não está inserida dentro de nenhuma tag específica) e as tags <INPUT TYPE = "text" NAME = "numero1">
. A primeira tag (<INPUT TYPE = "text" NAME = "numero1">) mostra, ao lado da frase, um campo de entrada de dados que possibilita ao usuário digitar algum valor (INPUT TYPE = "text"). Em seguida este campo é identificado (NAME = "numero1"). Finalmente, a outra tag (
) indica que, após o navegador mostrar estes elementos, a linha será quebrada e quaisquer outros elementos seguintes serão apresentados na outra linha. Percebam atentamente que o valor da cláusula NAME é exatamente o mesmo (e deve ser!) empregado na instrução de entrada de dados $numero1 = $_GET["numero1"]; no comando $_GET.

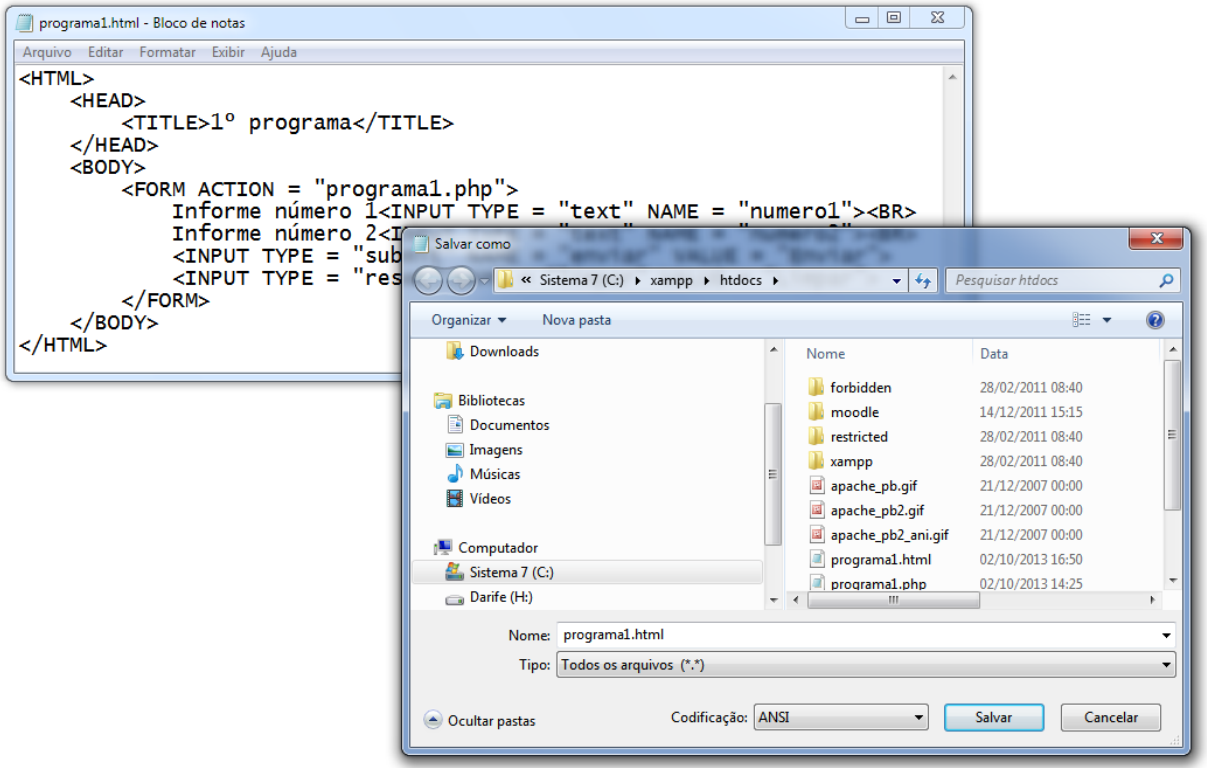
8. Informe o número 2<INPUT TYPE = "text" NAME = "numero2">
: esta linha tem exatamente a mesma função da linha

anterior. Vale atentarmos para o fato de que o `NAME` desta linha é `numero2`! Ou seja, não podemos identificar elementos distintos com o mesmo nome!

9. `<INPUT TYPE = "submit" NAME = "enviar" VALUE = "Enviar">`: como não temos que dar entrada a mais nenhum dado, é o momento de inserirmos um elemento que possibilite enviarmos os dados digitados nos `text` ao `programa1.php`. Esta linha insere um botão que irá enviar os dados. O botão de envio é fornecido na cláusula `INPUT TYPE = "submit"`, cujo nome é identificado em `NAME = "enviar"` e, na cláusula `VALUE = "Enviar"`, estamos colocando a palavra "Enviar" na face do botão (simplesmente por questão de gosto pessoal!).
10. `<INPUT TYPE = "reset" NAME = "limpar" VALUE = "Limpar">`: esta linha insere, logo ao lado do botão de enviar dados (`INPUT TYPE = "submit"`), um botão que "limpa" os dados do formulário. O que diferencia as ações dos botões é a cláusula `TYPE`. Se o conteúdo desta cláusula for `"submit"` então ao clicarmos neste botão, a ação será de envio e, se o conteúdo for `"submit"` então a ação será de limpar os campos do formulário. Tal como a linha anterior, é necessário nomearmos este objeto (`NAME = "limpar"`) e, colocarmos um rótulo sobre ele (`VALUE = "Limpar"`).

Este código deve ser digitado (sem a indicação das linhas) tal como foi o código do script PHP. Utilizaremos o bloco de notas e, da mesma forma, iremos salvar este arquivo na pasta C:\xampp\htdocs, conforme mostrado na Figura 7:

Figura 7 – Salvando a página HTML



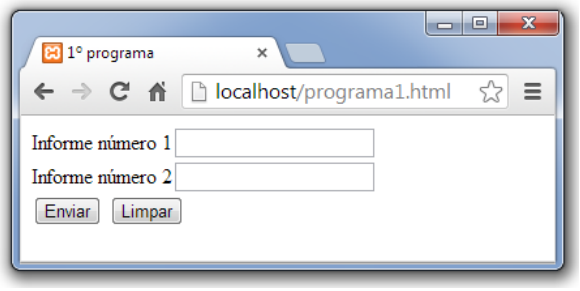
É imprescindível que o arquivo HTML seja salvo com a extensão `.html` e o tipo "Todos os arquivos (*.*)".

Ainda faltam dois passos para podermos testar o programa. Como já exposto anteriormente, estamos programando com uma linguagem cliente-servidor. Ou seja, temos que simular um servidor em nosso computador. Para isto, vamos iniciar o serviço "Apache" no aplicativo XAMPP (já previamente instalado), clicando no botão "Start", conforme mostrado anteriormente, na Figura 2. Agora basta acessarmos qualquer navegador e digitar na barra de endereços, `localhost/programa1.html`, onde:

- `localhost`, indica o endereço do servidor (no nosso caso, nosso próprio computador) e;
- `programa1.html`, o arquivo que contém as tags HTML que permitirão fornecermos dados para que o script PHP os processe e retorne a informação desejada.

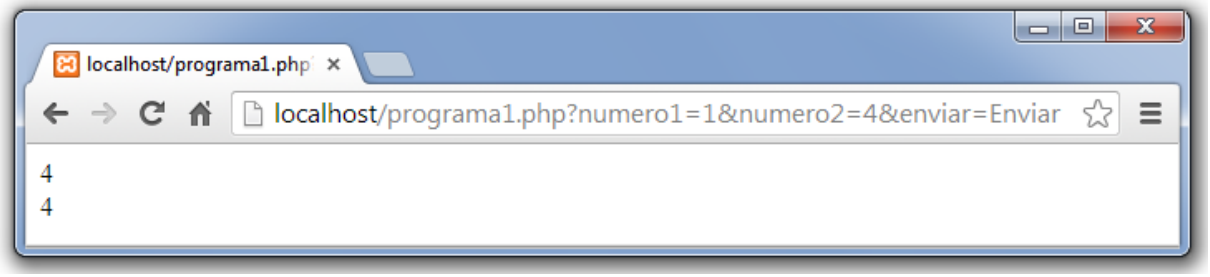
A Figura 8 mostra, no navegador, os elementos HTML que irão permitir o fornecimento e o envio dos dados:

Figura 8 – Rodando o programa1.html



Digitando os números 1 e 4, para as respectivas caixas de texto e clicando no botão "Enviar", teremos o seguinte resultado mostrado no navegador:

Figura 9 – Rodando programa1.php



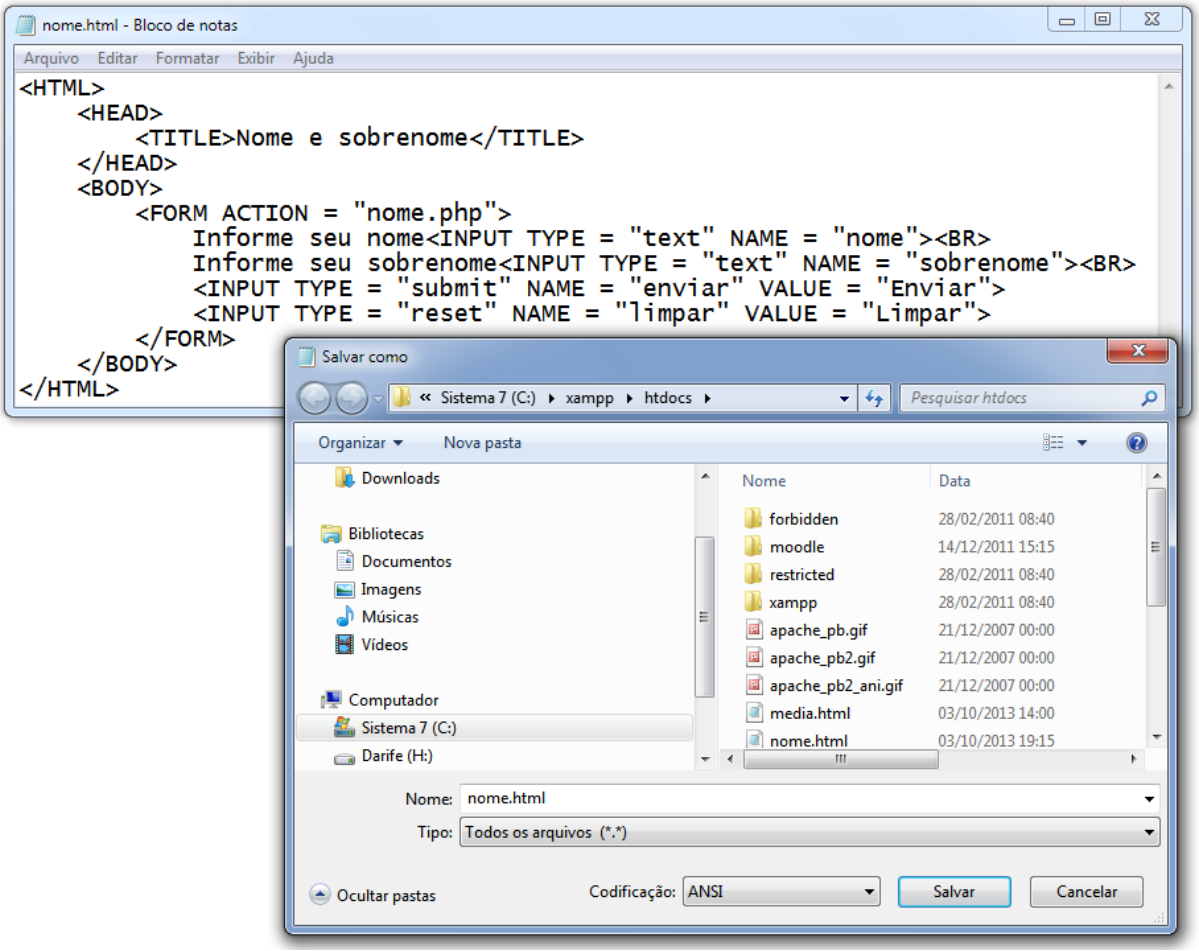
Ou seja, conforme a lógica mostrada na Figura 5 e desenvolvida nos Quadro 7 e Quadro 8, o resultado será que a variável `$numero2` conterá o número 4 e a variável `$numero1` terá o valor da `$numero2` atribuída a ela, perdendo, portanto, seu valor inicialmente digitado (1).

Vamos fazer um aplicativo que mostre seu nome e sobrenome. Qual será a lógica deste programa?

1. Temos que obter o nome da pessoa e armazena-lo em uma variável;
2. Da mesma forma, temos que obter o sobrenome e também armazena-lo;
3. Feito isto basta mostrar ambas as variáveis.

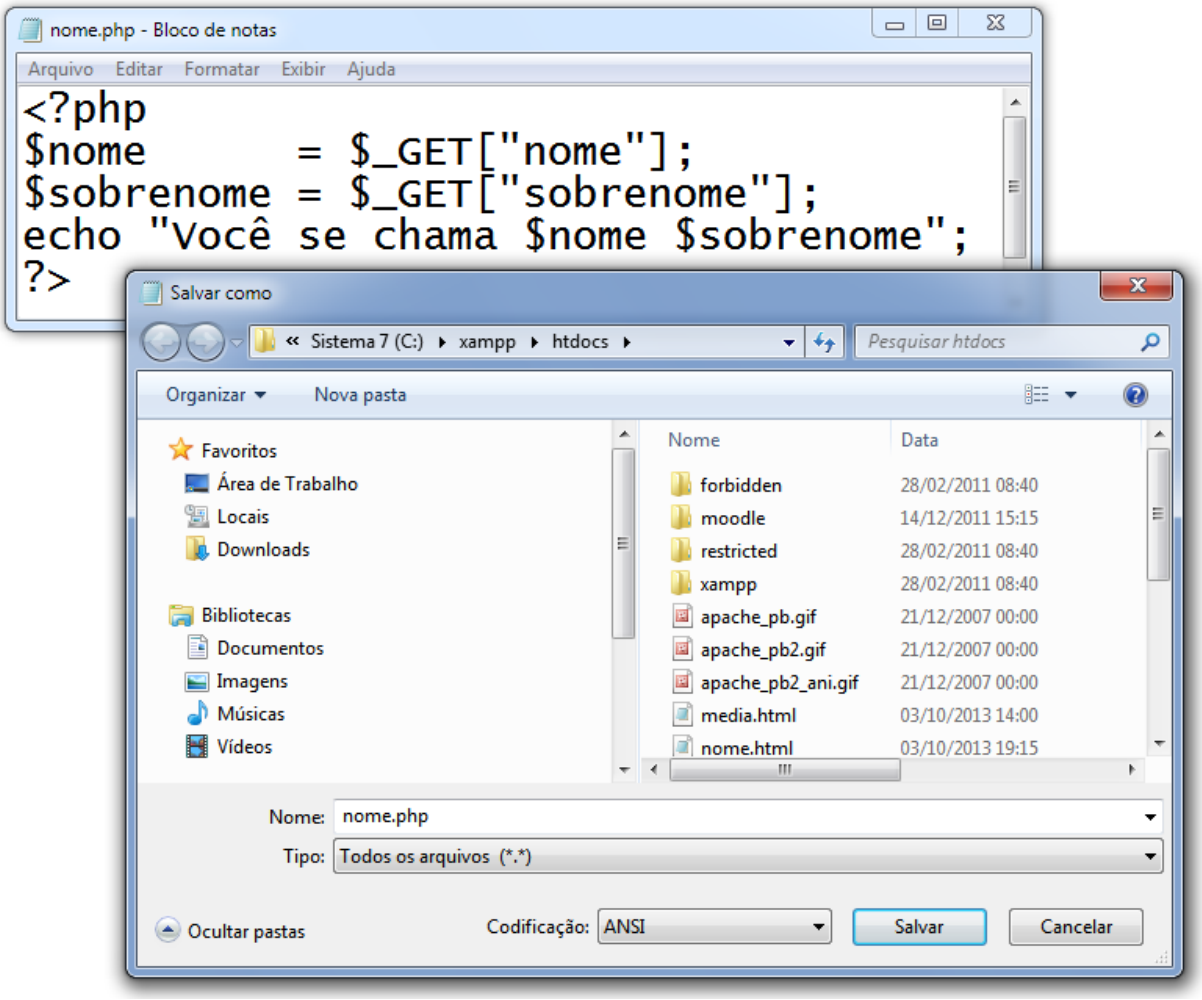
Primeiramente devemos implementar a página HTML que irá solicitar tanto o nome, quanto o sobrenome que, ao clicarmos no botão de enviar, serão enviados a um script PHP que irá armazena-los e mostra-los. Não podemos nos esquecer de salvar este documento com a extensão `.html`, na pasta C:\xampp\htdocs, com o tipo "Todos os arquivos (*.*)", conforme mostrado na Figura 10:

Figura 10 – Página HTML que obtém nome e sobrenome



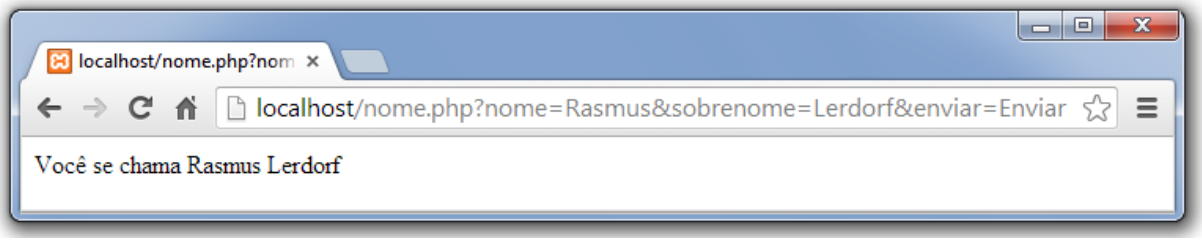
Já o script PHP que receberá os dados e os mostrará, não necessita de maiores explicações, somente que não podemos deixar de salvá-lo como `nome.php`, na pasta C:\xampp\htdocs, com o tipo “Todos os arquivos (*.*)”, tal como apresentado na Figura 11:

Figura 11 – Script PHP que mostra nome e sobrenome



Vamos executar a aplicação. Abra um navegador, digite `localhost/nome.html` na barra de endereços. Fornecendo o nome e sobrenome, nos respectivos campos, basta clicar no botão “Enviar” e o resultado será similar ao mostrado na Figura 12:

Figura 12 – Mostrando nome e sobrenome



Vejam que foi mostrada a frase “Você se chama” e logo após, os conteúdos das variáveis `$nome` e `$sobrenome`. Para obter este efeito, percebam que na linha do comando `echo`, todos os elementos que deveriam ser mostrados no navegador foram colocados entre aspas. O que isto indica? Indica que, quando desejarmos mostrar uma frase ou um texto, estes devem estar entre aspas. No caso de desejarmos mostrar somente o conteúdo de uma variável, as aspas são opcionais no comando `echo`.

Exercícios resolvidos

1. O que significa o comando a seguir?

   ```
   $numero1 = $_GET["numero1"];
   ```

   A variável `$numero1` recebe o valor armazenado em um objeto chamado `numero1`.

2. O que significa o comando a seguir?

   ```
   echo $numero1."<BR>";
   ```

   O comando `echo` irá mostrar o valor armazenado na variável `numero1` e em seguida, irá mostrar a `tag` de quebra de linha.

3. Elabore um script PHP que receba dois valores, os armazene em duas variáveis e troque os valores entre estas variáveis:

   ```
   <?php
   $variavel1 = $_GET["variavel1"];
   $variavel2 = $_GET["variavel2"];
   echo "O valor da variavel1 é $variável1<BR>";
   echo "O valor da variavel2 é $variável2<BR>";
   $aux = $variavel1;
   $variavel1 = $variavel2;
   $variavel2 = $aux;
   echo "O novo valor da variavel1 é $variável1<BR>";
   echo "O novo valor da variavel2 é $variável2";
   ?>
   ```

4. O que significa o comando a seguir?

   ```
   <INPUT TYPE = "text" NAME = "numero2">
   ```

   É uma instrução HTML que define uma caixa de texto (que irá receber algum valor) com o nome de `numero2`.

5. Qual a relação entre os comandos a seguir?

   ```
   <INPUT TYPE = "text" NAME = "numero2">
   $texto = $_GET["numero2"];
   ```

O primeiro comando indica que um objeto caixa de texto é denominado como `numero2` e o segundo comando armazena o conteúdo digitado nesta caixa de texto em uma variável chamada de `$texto`. O objeto `numero2` é HTML e a variável `$texto` é PHP.

6 OPERADORES MATEMÁTICOS

Além das instruções de entrada (`ler`, ou `$_GET`) e saída (`mostrar`, ou `echo`) de dados, recorrentemente temos que empregar em nossos programas, instruções que realizem operações matemáticas.

Grande parte das linguagens de programação adotam, basicamente, os seguintes símbolos para operadores matemáticos:

Quadro 9 – Operadores matemáticos

Símbolo	Operação	Significado	Exemplo
+	Soma	Somar dois números	$resultado = $numero1 + $numero2;
-	Subtração	Subtrair dois números	$resultado = $numero1 - $numero2;
*	Multiplicação	Multiplicar dois números	$resultado = $numero1 * $numero2;
/	Divisão	Dividir um número pelo outro	$resultado = $numero1 / $numero2;
%	Resto da divisão	Extrair o resto da divisão entre dois números inteiros	$resto = $numero1 % $numero2;

É importante que seja esclarecido que devemos utilizar parênteses quando for necessário quebrar alguma prioridade matemática. Vamos construir um script PHP que receba duas notas de um aluno, calcule e apresente a média:

Quadro 10 – Algoritmo e script PHP para calcular a média

Algoritmo para cálculo da média	Script PHP para cálculo da média
1. Início;	<?php
2. Variáveis: 2.1. Real: nota1, nota2, media;	/* o PHP prescinde da declaração de variáveis */
3. Ler nota1;	$nota1 = $_GET["nota1"];
4. Ler nota2;	$nota2 = $_GET["nota2"];
5. media ← (nota1 + nota2)/2;	$media = ($nota1 + $nota2)/2;
6. Mostrar media;	echo $media;
7. Fim.	?>

Vamos compreender o algoritmo, linha a linha:

1. O algoritmo deve ser inicializado;
2. Devemos identificar as variáveis envolvidas. Como temos duas notas, precisaremos de uma variável para cada nota. Da mesma forma, como devemos calcular a média, é necessário outra variável que armazene o valor da média, obtida através de algum cálculo.
3. Não conseguiremos calcular a média, se não soubermos o valor da primeira nota – por isto, o comando de entrada para a primeira nota.
4. Da mesma forma, é necessário que seja informado também o valor da segunda nota.
5. Tendo uma vez obtido os dados de entrada (`nota1` e `nota2`), já temos condições de calcular a média, primeiramente somando as notas e dividindo esta soma por dois. Percebam que, neste caso, temos que quebrar a

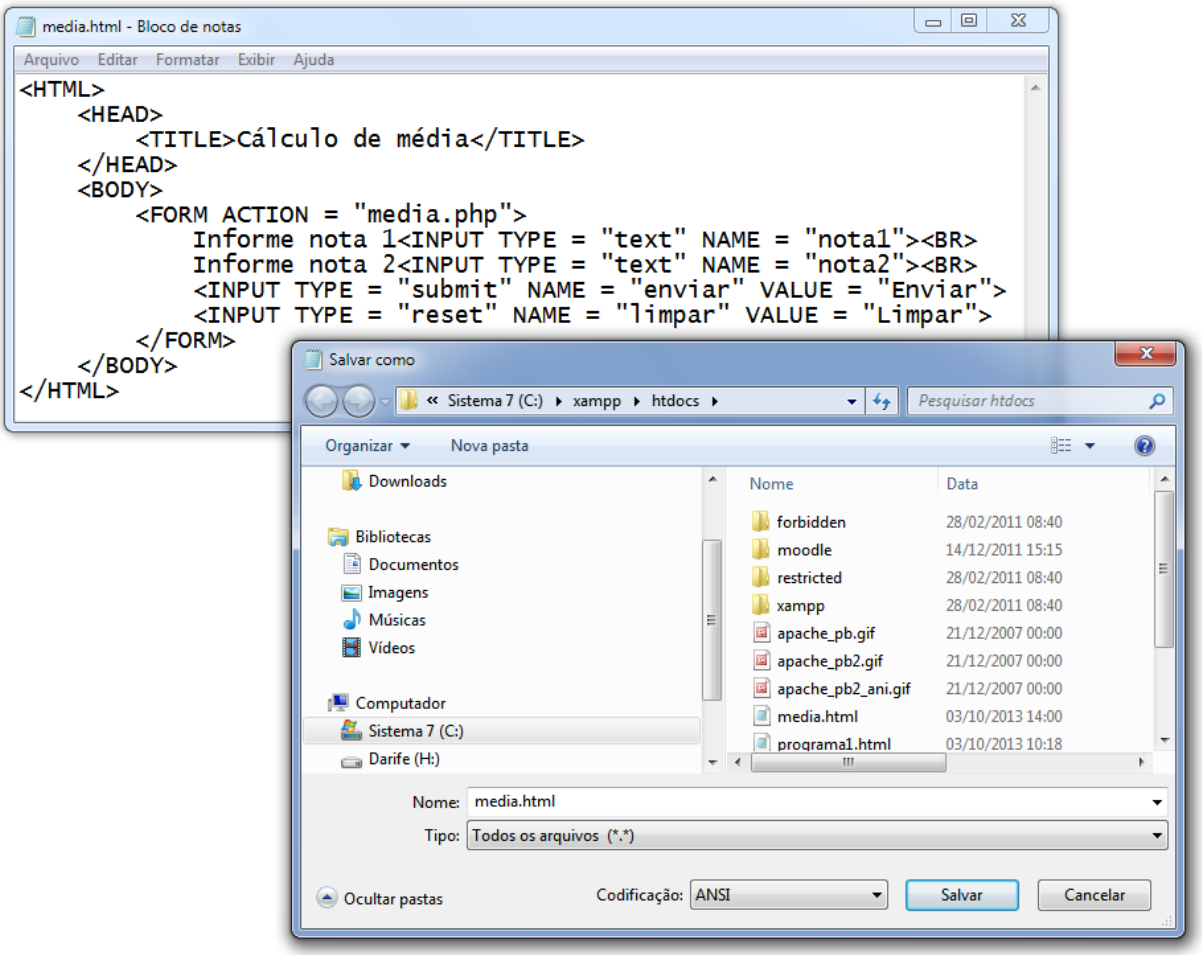
prioridade matemática da multiplicação sobre a soma – para fazer isto, colocamos a soma entre parênteses e, somente após, efetuamos a operação de divisão.

6. Se já obtemos o resultado pretendido, basta mostra-lo – por isto, o comando de saída para a variável `media`.

7. Finalmente, como não há mais nada a ser feito, finalizamos o algoritmo.

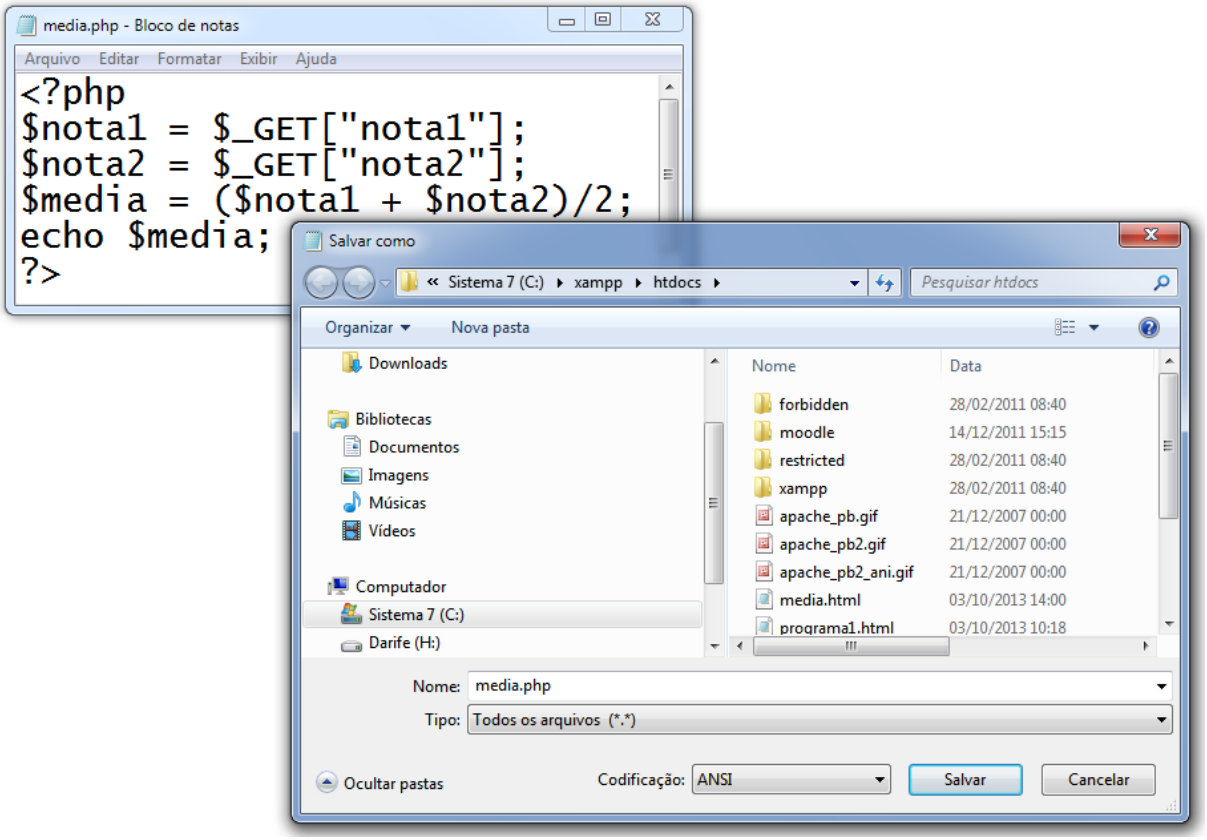
Agora que já sabemos os passos para implementar e testar um programa escrito em PHP, vamos digitar a página HTML que receberá e enviará os dados e o script PHP que irá calcular e mostrar a média. Primeiramente vamos compor a página HTML e salvá-la com o nome de `media.html`, na pasta C:\xampp\htdocs, como tipo "Todos os arquivos (*.*)", conforme visualizado na Figura 13:

Figura 13 – Salvando media.html



Vamos agora digitar os comandos PHP para o script e salvá-lo com a extensão `.php`, como "Todos os arquivos (*.*)", também na pasta C:\xampp\htdocs, de acordo com a Figura 14:

Figura 14 – Salvando media.php



Para verificar o resultado do programa, basta acessar qualquer navegador e, na barra de endereços, digitar `localhost/media.html`, informar dois valores para as notas e clicar no botão “Enviar”.

Além disto, vamos demonstrar o operador que extrai o resto da divisão entre dois números inteiros. Este operador é bastante útil quando desejamos saber se um número é múltiplo de outro. Por exemplo: 21 é múltiplo de 7, pois na divisão do primeiro pelo segundo, não sobra resto; já 4 não é múltiplo de 3, pois na divisão deles, sobra 1. A Figura 15 mostra estas operações:

Figura 15 – Restos de divisões

```
21 | 7        4 | 3
21 | 3        3 | 1
 0             1
```

Vamos desenvolver uma aplicação que mostra o resultado e o resto da divisão entre dois números. A página HTML que recebe estes números e os envia para serem processados é a seguinte:

Quadro 11 – Página HTML para obter dois números

```
<HTML>
    <HEAD>
        <TITLE>Resto da divisão</TITLE>
    </HEAD>
    <BODY>
        <FORM ACTION = "resto.php">
            Informe dividendo<INPUT TYPE = "text" NAME = "dividendo"><BR>
            Informe divisor<INPUT TYPE = "text" NAME = "divisor"><BR>
            <INPUT TYPE = "submit" NAME = "enviar" VALUE = "Enviar">
            <INPUT TYPE = "reset" NAME = "limpar" VALUE = "Limpar">
        </FORM>
    </BODY>
</HTML>
```

Qual deverá ser a lógica empregada para resolver este problema?

1. Obter os dois números a serem divididos;
2. Realizar a operação de divisão e armazenar o resultado;
3. Obter o resto da divisão e armazenar o resultado;
4. Mostrar o resultado da divisão e;
5. Mostrar o resto da divisão.

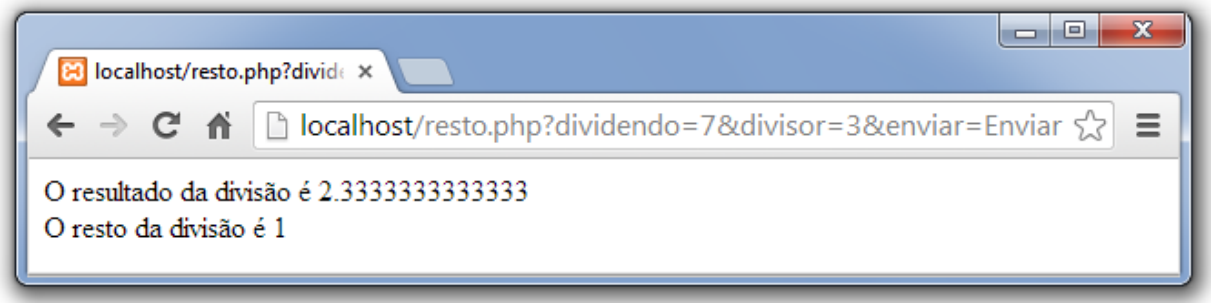
Ou seja, não existem diferenças significativas em relação ao algoritmo apresentado no Quadro 10. Assim sendo, vamos direto ao script PHP:

Quadro 12 – Script PHP para obter e mostrar o resto da divisão

```
<?php
    $dividendo = $_GET["dividendo"];
    $divisor   = $_GET["divisor"];
    $resultado = $dividendo/$divisor;
    $resto     = $dividendo%$divisor;
    echo "O resultado da divisão é $resultado<BR>";
    echo "O resto da divisão é $resto";
?>
```

Tendo salvado a página HTML com o nome de `resto.html` e o script php com o nome `resto.php`, ambos na pasta `C:\xampp\htdocs`, como tipo “Todos os arquivos (*.*)”, basta abrir um navegador e digitar, na barra de endereços, `localhost/resto.html`, informar dois números e clicar no botão “Enviar”. Se os números digitados tiverem sido 7 e 3, respectivamente, o resultado será tal como mostrado na Figura 16:

Figura 16 – Resultado do resto da divisão



Vamos desenvolver uma aplicação onde o usuário informa o preço de venda e o percentual de desconto e o sistema retorna o valor final a ser pago pelo cliente. Primeiramente é importante que pensemos na lógica geral da aplicação:

1. Alguém terá que informar o preço de venda;
2. Da mesma forma, o percentual de desconto também deverá ser fornecido;
3. De posse destes dados, temos que calcular o valor do desconto;
4. A segunda operação a ser feita é subtrair este percentual do preço de venda;
5. Finalmente, temos que mostrar o preço final de venda.

A lógica está estabelecida. Quais são as variáveis que devemos empregar para que este problema seja resolvido? Certamente, uma que irá armazenar o preço de venda, outra para o percentual de desconto, mais outra para o desconto concedido e, finalmente, uma que armazene o peço final a ser pago. Agora que já sabemos qual a lógica a ser utilizada, bem com as variáveis, basta desenvolver o algoritmo e traduzi-lo para uma linguagem de programação. O Quadro 13 mostra tanto o algoritmo, como o script PHP que resolve o problema:

Quadro 13 – Algoritmo e script PHP para calcular o desconto

Algoritmo para cálculo do desconto	Script PHP para cálculo do desconto
1. Início;	<?php
2. Variáveis: 2.1. Real: precoVenda, percDesc, desconto, precoFinal;	/* o PHP prescinde da declaração de variáveis */
3. Ler precoVenda;	$precoVenda = $_GET[“precoVenda”];
4. Ler percDesc;	$percDesc = $_GET[“percDesc”];
5. desconto ← precoVenda * (percDesc / 100);	$desconto = $precoVenda * ($percDesc / 100);
6. precoFinal ← precoVenda - desconto;	$precoFinal = $precoVenda - $desconto;
7. Mostrar precoFinal;	echo $precoFinal;
7. Fim.	?>

Vamos compreender o algoritmo linha a linha:

1. O algoritmo é iniciado;
2. As variáveis são declaradas;
3. O preço de venda é obtido;
4. O percentual de desconto é obtido;
5. O desconto é calculado e armazenado na variável `desconto`. A lógica empregada no cálculo é a seguinte: multiplicamos o preço de venda pelo percentual de desconto dividido por 100. Por que dividir por 100? Porque, computacionalmente, o valor fornecido na linha 4 não é uma unidade percentual, mas sim um número real qualquer. Imaginemos que o preço de venda seja 100 (linha 3), e o percentual de desconto seja 5 (5 por cento na verdade – linha 4). Qual será o desconto?

   ```
   desconto ← precoVenda * (percDesc / 100);
   desconto ← 100 * (5 / 100);
   desconto ← 100 * (0,05);
   desconto ← 5;
   ```

   Ou seja, R$ 5,00 é exatamente 5 por cento de R$ 100,00.
6. Esta linha subtrai o desconto calculado na linha anterior do preço inicial de venda (obtido na linha 3) e armazena o resultado na variável `precoFinal`.
7. O preço final é mostrado.
8. O algoritmo é encerrado.

A página HTML que permitirá a digitação de preço de venda e do percentual de desconto é mostrada no Quadro 14:

Quadro 14 - Página HTML para obter preço e desconto

```
<HTML>
    <HEAD>
        <TITLE>Desconto</TITLE>
    </HEAD>
    <BODY>
        <FORM ACTION = "desconto.php">
            Informe preço venda<INPUT TYPE = "text" NAME = "precoVenda"><BR>
            Informe % desconto<INPUT TYPE = "text" NAME = "percDesc"><BR>
            <INPUT TYPE = "submit" NAME = "enviar" VALUE = "Enviar">
            <INPUT TYPE = "reset" NAME = "limpar" VALUE = "Limpar">
        </FORM>
    </BODY>
</HTML>
```

A página deve ser salva com o nome `desconto.html` e o script php (do Quadro 13) com o nome `desconto.php` – não se esquecer que ambos devem serem salvos como o tipo "Todos os arquivos (*.*)". Para executar a aplicação, basta digitar `localhost/desconto.html` na barra de endereços do navegador, informar os valores e clicar no botão "Enviar".

Agora que entendemos esta aplicação, vamos aperfeiçoa-la, ou seja, resolver o mesmo problema utilizando menos variáveis e menos linhas de código. Como isto é possível?

1. Vamos empregar somente duas variáveis: preço de venda e percentual de desconto – que, como efetuado anteriormente, devem ser obtidas externamente (alguém deve digita-las).
2. O preço final de venda será o preço de venda menos o percentual de desconto obtido sobre o próprio preço de venda.
3. Mostrar o preço de venda depois de subtraído o desconto dele mesmo.

O algoritmo e, consequentemente, o script PHP assumem a seguinte sequência de instruções:

Quadro 15 – Algoritmo e script PHP otimizados para calcular o desconto

Algoritmo para cálculo do desconto	Script PHP para cálculo do desconto
1. Início;	<?php
2. Variáveis: 2.1. Real: precoVenda, percDesc;	/* o PHP prescinde da declaração de variáveis */
3. Ler precoVenda;	$precoVenda = $_GET["precoVenda"];
4. Ler percDesc;	$percDesc = $_GET["percDesc"];
5. precoVenda ← precoVenda - (precoVenda * (percDesc / 100));	$precoVenda = $precoVenda - ($precoVenda * ($percDesc / 100));
6. Mostrar precoVenda;	echo $precoVenda;
7. Fim.	?>

Vale explicarmos a 5ª linha do algoritmo. Esta linha deve ser analisada primeiramente pela operação efetuada entre os parênteses. Nela, é calculado o desconto sobre o preço de venda (`precoVenda * (percDesc / 100)`) – conforme foi efetuado na linha 5 do algoritmo mostrado no Quadro 13 – e depois subtraído do próprio preço de venda (`precoVenda - (precoVenda * (percDesc / 100))`) e, finalmente, armazenado na mesma variável (`precoVenda ← precoVenda - (precoVenda * (percDesc / 100));`). Qual a consequência desta instrução? Ao executarmos a linha 5 deste algoritmo, teremos perdido o preço inicial de venda (fornecido na linha 3). Porém, como neste problema, não é necessário que ele seja mantido até o final, não teremos maiores implicações. Para testar esta solução, renomeie o script PHP anterior, digite este e o salve com o mesmo nome (`desconto.php`) e execute a página HTML em seu navegador.

O uso de operadores matemáticos segue, basicamente, as mesmas regras das operações matemáticas, não havendo nenhuma particularidade que deva ser mencionada – basta o bom senso e o emprego de uma lógica que permita que o problema seja resolvido computacionalmente.

Exercícios resolvidos

1. Desenvolva um script PHP que receba dados referentes ao tempo gasto em uma viagem de automóvel, a velocidade média e, a taxa de consumo do veículo. Recebido estes dados, o script deve calcular e mostrar (1) a distância percorrida e (2) a quantidade de litros de combustível consumidos.

```
<?php
  $tempo      = $_GET["tempo"];
  $velocidade = $_GET["velocidade"];
  $consumo    = $_GET["consumo"];
  $distancia  = 0;
  $litros     = 0;
  $distancia  = $tempo * $velocidade;
  $litros     = $distancia / $consumo;
  echo "A velocidade média foi de $velocidade km/h<BR>";
  echo "O tempo gasto na viagem foi de $tempo horas<BR>";
  echo "A distancia percorrida foi de $distancia quilômetros<BR>";
  echo "O consumo foi de $litros litros";
?>
```

2. Faça um script PHP que receba duas datas (ambos no formato dia, mês e ano), calcule e mostre quantos dias transcorreram-se entre elas (considere que todos os meses apresentam somente 30 dias, e que a primeira data é menor do que a segunda):

```
<?php
  $dia1      = $_GET["dia1"];
  $mes1      = $_GET["mes1"];
  $ano1      = $_GET["ano1"];
  $dia2      = $_GET["dia2"];
  $mes2      = $_GET["mes2"];
  $ano2      = $_GET["ano2"];
  $diferenca = (($ano2 * 360) + ($mes2 * 30) + $dia2) -
               (($ano1 * 360) + ($mes1 * 30) + $dia1);
  echo "A diferença em dias é $diferenca";
?>
```

3. Faça um script PHP para receber o valor do custo e o valor de venda de um produto. O script deverá mostrar o lucro bruto e o percentual obtido pela venda deste produto.

```
<?php
  $custo      = $_GET["custo"];
  $venda      = $_GET["venda"];
  $bruto      = $venda - $custo;
  $percentual = ($bruto/$custo)*100;
  echo "O lucro bruto foi de $bruto<BR>";
  echo "O lucro percentual foi de $percentual";
?>
```

4. A nota final de um aluno é calculada a partir de 2 notas atribuídas a uma prova individual, e um trabalho em grupo, com diferentes pesos. Faça um script PHP que receba as duas notas, os dois pesos, calcule e mostre a média ponderada.

```
<?php
  $prova    = $_GET["prova"];
  $trabalho = $_GET["trabalho"];
  $peso1    = $_GET["peso1"];
  $peso2    = $_GET["peso2"];
  $media    = (($prova*7)+($trabalho*3))/($peso1 + $peso2);
  echo "média foi de $media";
?>
```

5. Sabe-se que o índice de massa corpórea (IMC) de uma pessoa é calculado a partir da relação entre peso (p) e altura (a), ou seja, IMC = p/a^2. Elabore um script PHP para receber o peso e a altura de uma pessoa e calcular e mostrar seu IMC:

```
<?php
  $peso   = $_GET["peso"];
  $altura = $_GET["altura"];
  $imc    = $peso/($altura * $altura);
  echo "Seu IMC é de $imc";

?>
```

7 COMANDOS DE DECISÃO

Até o momento, vimos problemas cujas soluções foram desenvolvidas de forma sequencial, ou seja, uma instrução após a outra, sendo todas executadas. Porém, existe uma categoria de comandos que possibilita escolhermos qual instrução deve ser executada – sempre em função de determinada condição. Vamos compreender o principal comando desta categoria: se (ou `if`, em PHP).

Este comando apresenta a seguinte estrutura:

Quadro 16 – Estrutura comando Se

`Se (condição)`
`  Então instrução 1;`
`  Senão instrução 2;`
`Fim se;`
	`if (condição)`
	`  instrução 1;`
	`  else instrução 2;`

Qual o seu mecanismo? A `instrução` somente será executada se a `condição` for verdadeira. Quais seus elementos?

- `Se`: indica que uma `condição` será testada.
- `condição`: é um teste lógico, cujo resultado somente pode ser `verdadeiro` ou `falso`. Os testes são efetuados a partir dos seguintes operadores:

Quadro 17 – Operadores lógicos

Operador	Significado	PHP
=	Igual	==
≠	Diferente	!= ou <>
>	Maior que	>
<	Menor que	<
≥	Maior ou igual a	>=
≤	Menor ou igual a	<=

Uma condição é efetuada SEMPRE entre dois elementos, podendo ser quaisquer combinações entre variável e variável e, variável e constante. Exemplos:

Variável e variável: (`nota1 > nota2`)

Variável e constante: (`media ≥ 70`)

- `Então`: define que uma `instrução` será executada, CASO A CONDIÇÃO TESTADA TENHA SIDO VERDADEIRA. Trata-se de uma cláusula OBRIGATÓRIA. Em PHP, não existe uma cláusula para o `Então`, ou seja, a instrução que sucede o teste da condição é justamente executada caso este teste tenha resultado verdadeiro.

- `Senão`: define a instrução a ser executada, CASO A CONDIÇÃO TESTADA TENHA APRESENTADO O RESULTADO FALSO. Ao contrário da cláusula `Então`, NÃO É OBRIGATÓRIA. Em PHP, `else`.
- `instrução 1`: é a instrução a ser executada, caso a condição tenha sido verdadeira. Esta instrução pode ser formada por qualquer comando, seja ele de entrada de dados, alguma operação matemática, saída de dados ou, até mesmo, um novo comando de decisão – tudo vai depender do problema e da lógica empregada. Se esta instrução for executada, a instrução do `Senão`, NÃO É EXECUTADA.
- `instrução 2`: é a instrução a ser executada, caso o resultado da condição seja falso. Da mesma forma como a `instrução 1`, pode ser qualquer outro comando, desde que atenda à solução. Além disto, se esta instrução for executada, é porque a `instrução 1`, do `Então`, não foi executada!
- `Fim se`: indica o final da instrução, ou seja, os comandos subsequentes não estão vinculados à condição testada.

Além disto, é importante que se note que o comando `Se` NÃO É FINALIZADO COM PONTO E VÍRGULA! Na verdade, não faz sentido finaliza-lo, pois não é um comando que realiza uma instrução em si – apenas testa uma condição, sendo que a instrução será realizada caso a condição tenha sido verdadeira (ou falsa, quando se emprega a cláusula `Senão`).

Vamos a um exemplo: imaginemos uma aplicação de uma loja onde, se uma compra for paga a prazo o valor final terá um acréscimo de 10%. Inicialmente, vamos estabelecer a lógica da aplicação:

1. Temos que saber qual o preço inicial do produto;
2. Temos que saber a forma de pagamento;
3. Caso a forma seja a prazo, o preço inicial será acrescido de 10%;
4. Finalmente, o preço final deverá ser mostrado.

E quais serão as variáveis envolvidas? Uma que armazene o preço inicial e outra que armazene a forma de pagamento. Percebam que, caso a forma seja a prazo, utilizaremos a mesma variável do preço inicial para armazenar o preço final – como foi efetuado no exemplo do Quadro 15. Vamos ao algoritmo (e script PHP):

Quadro 18 – Algoritmo e script PHP para calcular o acréscimo

Algoritmo para cálculo do acréscimo	Script PHP para cálculo do acréscimo
1. Início;	<?php
2. Variáveis: 2.1. Real: precoVenda, forma;	/* o PHP prescinde da declaração de variáveis */
3. Ler precoVenda;	$precoVenda = $_GET["precoVenda"];
4. Ler forma;	$forma = $_GET["forma"];
5. Se (forma = "prazo") Então 5.1. precoVenda ← precoVenda + (precoVenda * 10/100);	if ($forma == "prazo") $precoVenda = $precoVenda + ($precoVenda * 10/100);
6. Fim se;	/* O if é finalizado pela instrução que o sucede */
7. Mostrar precoVenda;	echo $precoVenda;
8. Fim.	?>

Vamos entender as linhas 4, 5 e 6:

4. Nesta linha, é obtido o valor da variável `forma`.

5. Após termos recebido o valor que informa a forma de pagamento, o comando `Se` testa se o conteúdo da variável `forma` é igual ao termo "prazo". Se for verdade que a palavra digitada na linha 4 seja "prazo", então será executado o comando vinculado à cláusula `Então`. Nesta cláusula, a variável `precoVenda` tem seu valor atualizado para 10% além de seu próprio valor (a lógica empregada é exatamente a mesma encontrada na linha 5 do Quadro 15). Vale notar que no script PHP, não foi informado nenhuma cláusula correspondente ao `Então`. O PHP "entende" que tudo que vem logo após a condição, finalizando no primeiro ponto e vírgula, é o que deve ser executado, caso a condição seja verdadeira.

6. Como não há mais nenhum comando a ser efetuado, esta linha finaliza o comando `Se`. Ou seja, o comando da linha 7 será executado independentemente da condição testada na linha 5. É importante perceber que não existe um comando, em PHP, correspondente ao `Fim se`. O fim de uma instrução `if` é determinado pelo ponto e vírgula, tanto do comando associado à condição verdadeira ou pelo comando da cláusula `else` (comando associado à condição falsa).

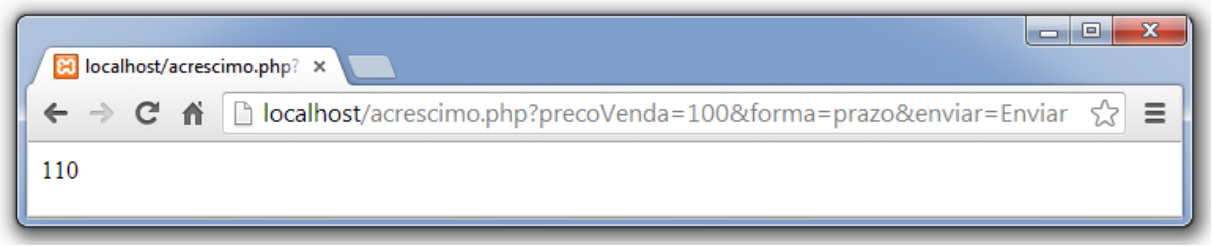
O script PHP deve ser digitado e salvo com o nome de `acréscimo.php`. Já a página HTML que permitirá a digitação de preço de venda e da forma de pagamento é mostrada no Quadro 19 e sugere-se que seja salva com o nome de `acréscimo.html`:

Quadro 19 - Página HTML para obter preço e forma de pagamento

```
<HTML>
    <HEAD>
        <TITLE>Acréscimo</TITLE>
    </HEAD>
    <BODY>
        <FORM ACTION = "acrescimo.php">
            Informe preço venda<INPUT TYPE = "text" NAME = "precoVenda"><BR>
            Informe forma pagto<INPUT TYPE = "text" NAME = "forma"><BR>
            <INPUT TYPE = "submit" NAME = "enviar" VALUE = "Enviar">
            <INPUT TYPE = "reset" NAME = "limpar" VALUE = "Limpar">
        </FORM>
    </BODY>
</HTML>
```

Executem o aplicativo e digitem “100” (sem aspas) na caixa de texto que solicita o preço de venda e, na caixa de texto que solicita a forma de pagamento, digite “prazo” (sem aspas) e verifiquem o resultado. Será exatamente como mostrado na Figura 17:

Figura 17 – Resultado comando Se



Agora executem novamente a aplicação (`localhost/acrescimo.html`) e digitem qualquer palavra no campo “`Informe forma pagto`” (mantenha 100 para o preço de venda) e clique em “Enviar”. Percebam que o resultado será sempre “100”. O que isto significa? Que o comando `Se` irá verificar se o conteúdo da variável `forma` conterá única e exclusivamente o valor “prazo” – qualquer valor diferente desta palavra fará com que a condição seja falsa, ou seja, o comando vinculado ao `Se` não é executado! Portanto, muita atenção com o que está sendo testado!

Vocês devem ter notado que não empregamos a cláusula `Senão`. Neste exemplo, não foi necessário – lembrem-se que o `Senão` é opcional! Vamos a um exemplo onde teremos que empregar o `Senão`. Vamos imaginar que, no mesmo aplicativo, teremos que acrescentar o fato de que, se a forma de pagamento for “vista” então teremos um desconto de 5% (“prazo” ainda manterá acréscimo de 10%). Tanto o algoritmo quanto o script PHp são mostrado no Quadro 20:

Quadro 20 – Algoritmo e script PHP para calcular o acréscimo e desconto

Algoritmo para cálculo do acréscimo	Script PHP para cálculo do acréscimo
1. Início;	<?php
2. Variáveis: 2.1. Real: precoVenda, forma;	/* o PHP prescinde da declaração de variáveis */
3. Ler precoVenda;	$precoVenda = $_GET["precoVenda"];
4. Ler forma;	$forma = $_GET["forma"];
5. Se (forma = "prazo") Então 5.1. precoVenda ← precoVenda + (precoVenda * 10/100); Senão Se (forma = "vista") Então 5.2. precoVenda ← precoVenda – (precoVenda * 5/100);	if ($forma == "prazo") $precoVenda = $precoVenda + ($precoVenda * 10/100); else if ($forma == "vista") $precoVenda = $precoVenda - ($precoVenda * 5/100);
6. Fim se;	/* O if é finalizado pela instrução que o sucede */
7. Mostrar precoVenda;	echo $precoVenda;
8. Fim.	?>

Percebam que na linha 5, acrescentamos a cláusula `Senão` e vinculamos a ela, outro comando `Se` – que testa se o conteúdo da variável `forma` é igual à palavra "vista". Se for, então a linha 5.2. é executada. Notem que no script PHP, a cláusula `Senão` é expressa pelo comando `else`. Renomeiem o script PHP anterior e salvem este com o mesmo nome e executem a aplicação. Testem primeiro com a palavra "prazo", depois com a palavra "vista" e depois, com qualquer palavra que não seja "prazo" ou "vista". Notem que, nesta situação, o preço de venda não se altera, pois a palavra digitada não é nem "prazo" nem "vista"!

Existem situações onde será necessário que mais de uma instrução seja executada, tanto no `Então`, quanto no `Senão`. Em PHP, este conjunto de instruções deve estar delimitado por { } (chaves). Para ilustrar esta situação, vamos desenvolver uma página HTML que permita que sejam digitados três valores de uma equação de 2º grau que, enviados para um script PHP, calcula e mostre as raízes desta equação, empregando a fórmula de Bhaskara:

$$x = \frac{-b \pm \sqrt{\Delta}}{2a} \quad (1)$$

Sendo que:

$$\Delta = b^2 - 4ac \quad (2)$$

Ou seja, se Δ for igual a zero, haverá apenas uma única raiz; se for maior que zero, haverão duas raízes e; menor que zero, não haverão raízes. A lógica de desenvolvimento para este problema, seguirá os seguintes passos:

1. Temos que obter os valores de a, b e c;
2. Tendo obtido os valores de a, b e c, temos que calcular o valor de Δ;
3. Comparamos o valor calculado de Δ com zero e, se o valor for menor que zero nenhuma raiz é calculada e mostramos uma mensagem equivalente;

4. Caso o valor de Δ não tenha sido menor que zero, temos que testar para verificar se é igual a zero. Neste caso, se esta condição for verdadeira, calculamos uma única raiz e mostramos este valor;

5. Se o valor não for menor ou igual a zero, ele só pode ser maior que zero e, neste caso, calculamos as duas raízes e as mostramos.

6. Não tendo mais nada a ser efetuado, finalizamos o programa.

As variáveis estão presentes na própria lógica para a solução do problema: a, b, c (passo 1), delta (passo 2), x1 (passo 4) e x2 (passo 5). Vamos primeiramente ao algoritmo:

Quadro 21 – Algoritmo para calcular as raízes da equação de 2º grau

```
1. Início;
2. Variáveis:
   2.1. Real: a, b, c, delta, x1, x2;
3. Ler a;
4. Ler b;
5. Ler c;
6. delta ← (b*b) - 4*a*c ;
7. Se (delta < 0)
       Então 7.1. Mostrar "não há raízes";
       Senão 7.2. Se (delta = 0)
                  Então
                  {
                     7.2.1. x1 ← (-b + √delta)/(2*a);
                     7.2.2. Mostrar x1;
                  }
                  Senão
                  {
                     7.2.3. x1 ← (-b + √delta)/(2*a);
                     7.2.4. x2 ← (-b - √delta)/(2*a);
                     7.2.5. Mostrar x1;
                     7.2.6. Mostrar x2;
                  }
              7.3. Fim se;
8. Fim se;
9. Fim.
```

Duas linhas merecem nossa atenção:

6. A variável `delta` recebe o valor de `b` ao quadrado menos 4 x `a` x `c`. Como não temos uma operação matemática que eleve um número a uma determinada potência (iremos resolver este problemas nas próximas seções), multiplicamos o valor de `b` por ele mesmo.

7. Esta linha testa se o valor de delta é menor que zero. Se for verdade, a linha 7.1 é executada e os comandos do `Senão` (da linha 7.2), não são executados, desviando o controle do algoritmo para a linha seguinte do `Fim`

`se` da linha 8. Por outro lado, se o valor de delta não for menor que zero (então só pode ser igual ou maior que zero), a linha 7.1 não é executada e o controle será desviado para a linha 7.2. Nesta linha, outro comando `Se` é executado, testando se o valor de delta é igual a zero. Se esta condição for verdadeira, os comandos do `Então` (linhas 7.2.1 e 7.2.2 que estão agrupadas por chaves) são executados, um após o outro. Como este `Então` é efetuado, os comandos do `Senão` (linhas 7.2.3, 7.2.4, 7.2.5 e 7.2.6) não são executados e o controle para a linha subsequente ao `Fim se` da linha 7.3. Como esta linha é uma instrução `Fim se`, a próxima linha, que finaliza o algoritmo, é executada. Porém, se o valor de delta, testado na linha 7.2, não for igual a zero, este valor só pode ser maior que zero, ou seja, os comandos do `Senão` (que estão agrupados por chaves) são executados. Neste caso, após a instrução da linha 7.2.6 ser efetuada, o controle passa para a linha 7.3, em seguida 8 e, finalmente 9, encerrando o algoritmo.

A tradução do algoritmo para o script PHP é quase simultânea

Quadro 22 – Script PHP para calcular as raízes da equação de 2º grau

```
<?php
   $a = $_GET["a"];
   $b = $_GET["b"];
   $c = $_GET["c"];
   $delta = ($b * $b) - (4 * $a * $c);
   if ($delta < 0)
      echo "Não há raízes";
   else if ($delta == 0)
        {
           $x1 = (-$b + sqrt($delta))/(2 * $a);
           echo "A única raiz é $x1";
        }
   else
        {
           $x1 = (-$b + sqrt($delta))/(2 * $a);
           $x2 = (-$b - sqrt($delta))/(2 * $a);
           echo "Uma das raízes é $x1<BR>";
           echo "A outra raiz é $x2";
        }
?>
```

Vale atentarmos para os cálculos do `x1` e `x2`. A linguagem PHP (assim como várias outras) não possui um operador matemático que extraia a raiz quadrada de algum número ou expressão. Porém, esta linguagem possui uma função (mais adiante, abordaremos este conceito) que realiza esta tarefa. A função é `sqrt()` que retorna o valor da raiz quadrada do elemento (variável, constante, expressão) que estiver entre os parênteses. Convém notar que, da mesma forma que o `if` não apresenta ponto e vírgula, também a cláusula `else` não

apresenta este finalizador de instrução. Por que? Porque o `else` não é uma instrução em si – ele apenas indica a instrução (ou o conjunto de instruções) que será executada caso a condição testada pelo `if` anterior tenho sido falsa!

Para testar a aplicação, vamos salvar o script PHP com o nome de `raizes.php` e criar uma página HTML que possibilite ao usuário informar valores para a, b e c e salvá-la como `raizes.html`. O código utilizado para implementar a página HTML é mostrado no Quadro 23:

Quadro 23 - Página HTML para obter valores de a, b e c

```
<HTML>
    <HEAD>
        <TITLE>Raízes da equação de 2° grau</TITLE>
    </HEAD>
    <BODY>
        <FORM ACTION = "raizes.php">
            Informe a<INPUT TYPE = "text" NAME = "a"><BR>
            Informe b<INPUT TYPE = "text" NAME = "b"><BR>
            Informe c<INPUT TYPE = "text" NAME = "c"><BR>
            <INPUT TYPE = "submit" NAME = "enviar" VALUE = "Enviar">
            <INPUT TYPE = "reset" NAME = "limpar" VALUE = "Limpar">
        </FORM>
    </BODY>
</HTML>
```

Até o momento mostramos exemplos onde apenas uma única condição é testada. Porém, podem existir situações onde é necessário testarmos mais de uma condição. Para isto empregamos os conectivos lógicos `E` e `OU`:

- `E`: quando todas as condições estão conectadas por E, OS RESULTADOS DE TODAS DEVEM SER VERDADE, para que os comandos vinculados ao `Então` sejam executados. Caso uma única condição apresente `Falso` como resultado, os comandos associados ao `Então` já não são executados.
- `OU`: basta que APENAS UMA DAS CONDIÇÕES conectadas por `OU` SEJA VERDADEIRA para que os comandos vinculados ao `Então` sejam executados. As instruções do `Então` só não serão executadas SE TODAS AS CONDIÇÕES FOREM FALSAS.

Vamos a um exemplo empregando tanto o `OU`, quanto o `E`. Imaginemos um aplicativo que receba duas notas de alunos, bem como o percentual de falta. Estes devem serão enviados para um script PHP que irá:

1. Calcular a média;
2. Irá apresentar a mensagem “Aprovado” caso a média tenha sido maior ou igual a 70 e o percentual de faltas tenha sido menor ou igual a 25.

3. Irá apresentar a mensagem “Em exame” caso a média tenha sido maior ou igual a 40 (e menor que 70) e o percentual de faltas tenha sido menor ou igual a 25.

4. Irá apresentar a mensagem “Reprovado” caso a média tenha sido menor que 40 ou o percentual de faltas tenha sido maior que 25.

A página HTML (salva com o nome de `aproveitamento.html`) que receberá e enviará os dados é mostrada no Quadro 24:

Quadro 24 – Página HTML para dados de aluno

```
<HTML>
    <HEAD>
        <TITLE>Aproveitamento acadêmica</TITLE>
    </HEAD>
    <BODY>
        <FORM ACTION = "aproveitamento.php">
            Informe nota1<INPUT TYPE = "text" NAME = "nota1"><BR>
            Informe nota2<INPUT TYPE = "text" NAME = "nota2"><BR>
            Informe faltas<INPUT TYPE = "text" NAME = "faltas"><BR>
            <INPUT TYPE = "submit" NAME = "enviar" VALUE = "Enviar">
            <INPUT TYPE = "reset" NAME = "limpar" VALUE = "Limpar">
        </FORM>
    </BODY>
</HTML>
```

Como a lógica já está descrita no próprio problema, vamos diretamente ao algoritmo:

Quadro 25 – Algoritmo para cálculo de aproveitamento de aluno

```
1. Início;
2. Variáveis:
   2.1. Real: nota1, nota2, faltas, media;
3. Ler nota1;
4. Ler nota2;
5. Ler faltas;
6. media ← (nota1 + nota2)/2;
7. Se ((media < 40) OU (faltas > 25))
      Então 7.1. Mostrar "Reprovado";
      Senão 7.2. Se ((media ≥ 70) E (faltas ≤ 25))
                    Então 7.2.1. Mostrar "Aprovado";
                    Senão 7.2.2. Mostrar "Em exame";
            7.3. Fim se;
8. Fim se;
9. Fim.
```

A linha 7 merece ser discutida. Tendo obtido o percentual de faltas na linha 5 e calculado e armazenado a média na linha 6, o comando `Se`, da linha 7, testa duas condições (média ser menor que 40 e faltas ser maior que 25). Como estas condições estão conectadas por `OU`, basta que uma delas seja verdade para que a instrução da cláusula `Então` seja executada (linha 7.1). Somente se ambas forem falsas, a linha 7.1. não é executada e o controle do algoritmo é desviado para a cláusula `Senão` da linha 7.2. Nesta

linha, novo teste é efetuado sobre duas condições (média ser maior ou igual a 70 e faltas ser menor ou igual a 25). Somente se os resultados de ambas forem verdadeiros, a linha 7.2.2., da cláusula `Então` é executada; caso contrário, ou seja, uma delas apresentou resultado falso, a cláusula `Senão` é executada.

Vamos transformar este algoritmo em um script PHP e salvá-lo como `aproveitamento.html`:

Quadro 26 – Script PHP para cálculo de aproveitamento de aluno

```
<?php
$nota1 = $_GET["nota1"];
$nota2 = $_GET["nota2"];
$faltas = $_GET["faltas"];
$media = ($nota1 + $nota2)/2;
if (($media < 40) OR ($faltas > 25))
    echo "Reprovado";
else if (($media >= 70) AND ($faltas <= 25))
    echo "Aprovado";
else
    echo "Em exame";
?>
```

Efetuem todos os testes necessários para verificarem as situações decorrentes do fato de uma e/ou outra condição ser verdadeira.

Vale registrar que existem mais comandos de decisão, porém com o comando `Se`, é possível realizar quaisquer processamentos que exijam tais funções. Da mesma forma, relacionado aos comandos de decisão, existem mais operadores lógicos do que os abordados nesta seção – ainda que as combinações de `E` e `OU` permitam que se desenvolvam grande parte das necessidades de testes de condições conjuntas.

Exercícios resolvidos

1. Vamos retomar o exercício do cálculo do IMC. O resultado indica o grau de obesidade, conforme o Quadro 27:

Quadro 27 – IMC

IMC	Situação
Abaixo de 18,5	Magreza
De 18,5 e abaixo de 25	Normal
Maior ou igual a 25	Obesidade

Elabore um script PHP que receba o peso e a altura de uma pessoa, calcule e mostre o IMC, bem como a situação em que a pessoa se encontra:

```
<?php
  $peso   = $_GET["peso"];
  $altura = $_GET["altura"];
  $imc    = $peso/($altura * $altura);
  echo "Seu IMC é de $imc<BR>";
```

```
  if ($imc < 18.5)
    echo "Sua situação é de magreza.";
  else if ($imc >= 55)
    echo "Sua situação é de obesidade.";
  else
    echo "Sua situação é de normalidade.";
?>
```

2. Faça um script PHP que receba os valores de 3 ângulos de um triângulo. Após estes valores serem obtidos, o script irá informar qual o tipo do triângulo formado: retângulo (quando um dos ângulos é de 90º), obtuso (quando um dos ângulos for maior que 90º) ou agudo (quando todos os ângulos forem menores que 90º). O script deve levar em consideração que os 3 ângulos fornecidos podem não formar um triângulo (para ser um triângulo, a soma dos ângulos deve ser igual a 180º).

```
<?php
  $a = $_GET["a"];
  $b = $_GET["b"];
  $c = $_GET["c"];
  if (($a <= 0) OR ($b <= 0) OR ($c <= 0))
    echo "Os ângulos não formam um triângulo";
  else if (($a + $b + $c) <> 180)
    echo "Os ângulos não formam um triângulo";
  else if (($a == 90) OR ($b == 90) OR ($c == 90))
    echo "O triângulo é retângulo";
  else if (($a > 90) OR ($b > 90) OR ($c > 90))
    echo "O triângulo é obtusângulo";
  else
    echo "O triângulo é acutângulo";
?>
```

3. Faça um script PHP que receba os valores de três lados de um triângulo. Após estes valores serem obtidos, calcular e mostrar a classificação segundo seus lados, sabendo-se que se os três lados forem iguais teremos um triângulo equilátero; se dois lados forem iguais teremos um triângulo isósceles e; se os três lados forem diferentes teremos um triângulo escaleno. O script deve levar em consideração que os 3 lados fornecidos podem não formar um triângulo (se algum lado for maior ou igual à soma dos demais, então não teremos um triângulo).

```
<?php
  $a = $_GET["a"];
  $b = $_GET["b"];
  $c = $_GET["c"];
  if (($a <= 0) OR ($b <= 0) OR ($c <= 0))
    echo "Os lados não formam um triângulo";
  else if (($a >= $b + $c) OR ($b >= $a + $c) OR ($c >= $a + $b))
    echo "Os lados não formam um triângulo";
  else if (($a == $b) AND ($b == $c))
    echo "O triângulo é equilátero";
  else if (($a <> $b) AND ($b <> $c) AND ($a <> $c))
    echo "O triângulo é escaleno";
  else
    echo "O triângulo é isósceles";
?>
```

4. A Prefeitura de um município abriu uma linha de crédito para os funcionários estatutários. O valor máximo da prestação não poderá ultrapassar 30% do salário bruto. Elaborar um script PHP que, a partir de dados recebidos de salário bruto e o valor da prestação, informa se o empréstimo pode ou não ser concedido.

```
<?php
  $salario   = $_GET["salario"];
  $prestacao = $_GET["prestacao"];
  if ($prestacao > $salario*30/100)
    echo "Empréstimo não pode ser concedido";
  else
    echo "Empréstimo pode ser concedido";
?>
```

5. Elaborar um script PHP que receba o salário de uma pessoa, calcule e mostre o desconto de INSS a partir da tabela a seguir:

Menor ou igual a R$ 600,00	Isento
Maior que R$ 600,00 e menor ou igual a R$ 1200,00	20%
Maior que R$ 1200,00 e menor ou igual a R$ 2000,00	25%
Maior que R$ 2000,00	30%

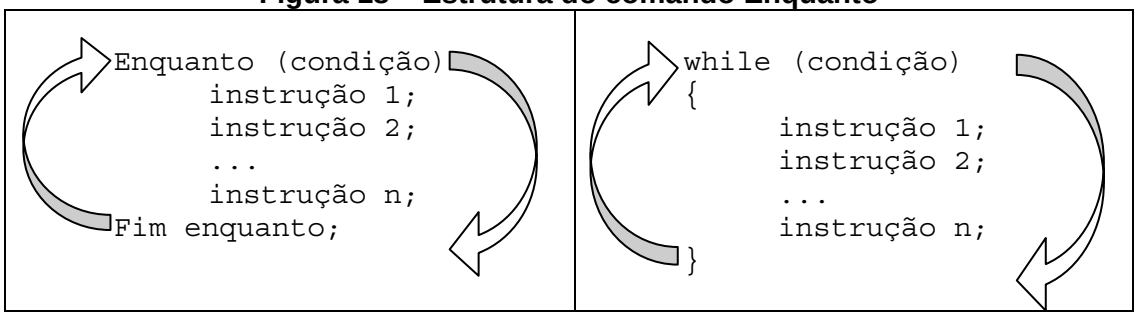
```
<?php
  $salario  = $_POST["salario"];
  $desconto = 0;
  if ($salario <= 600)
    $desconto = 0;
  else if (($salario > 600) AND ($salario <= 1200))
    $desconto = $salario * 20/100;
  else if (($salario > 1200) AND ($salario <= 2000))
    $desconto = $salario * 25/100;
  else if ($salario > 2000)
    $desconto = $salario * 30/100;
  echo "Para seu salário, o desconto é de $desconto";

?>
```

8 COMANDOS DE REPETIÇÃO

Até o momento vimos que cada linha de um programa de computador é executada uma única vez. Se quiséssemos executar novamente algum trecho não havia possibilidade a não ser que executássemos novamente toda a aplicação. Felizmente existe uma categoria de comandos que possibilita que trechos do programa sejam repetidos enquanto uma condição for verdadeira: são os comandos de repetição.

Basicamente, temos duas instruções de repetição: `Enquanto` e `Para`. Vamos compreender a estrutura de um comando `Enquanto`:

Figura 18 – Estrutura do comando Enquanto



Os elementos desta instrução apresentam as seguintes características:

- `Enquanto`: indica que, a partir daquele ponto, uma instrução (ou um conjunto de instruções) será repetida DESDE QUE O RESULTADO DE UMA `condição` SEJA VERDADEIRO. Caso este resultado seja falso, o controle do programa é desviado para a linha subsequente ao `Fim enquanto`, encerrando, portanto, a repetição das `instruções 1, 2, n`. Em PHP, empregamos o comando `while`.
- `condição`: apresenta exatamente as mesmas características de uma condição empregada em comandos de decisão – ou seja, nada mais é do que um teste lógico realizado entre dois elementos (variáveis e/ou constantes) cujo resultado só pode ser `verdadeiro` ou `falso`.
- `instrução 1, 2, n`: são as instruções a serem repetidas enquanto a condição testada for verdadeira. Estes instruções podem ser de entrada ou saída de dados, decisões e, até mesmo, nova instrução de repetição.
- `Fim enquanto`: indica o final da instrução, ou seja, os comandos subsequentes não estão vinculados à condição testada. Diferentemente da forma como as instruções empregadas até o momento se sucedem, a instrução executada após o `Fim enquanto` é o `Enquanto` (e não a linha subsequente), retornando, portanto, à `condição`. Se esta for verdadeira,

repetem-se as `instruções 1, 2` e `n`; caso contrário, o controle é desviado para a linha seguinte ao `Fim enquanto`, formando um laço de comandos. Em PHP, não existe uma cláusula correspondente. O que determina o conjunto de instruções a serem repetidas por um comando `while` são as chaves { } – da mesma forma como empregado no comando `if`.

O exemplo a seguir é bastante ilustrativo quanto ao uso do `Enquanto`. Vamos resgatar a situação onde precisávamos elevar um número ao quadrado (o exemplo apresentado no Quadro 21). Como, naquele ponto, não tínhamos conhecimento de instruções de repetição, apenas multiplicamos o número por ele mesmo (ainda bem que era elevado ao quadrado – imaginem se fosse elevado à centésima potência!). Agora temos condições de criar uma aplicação que receba um número e um expoente e mostre o resultado desta operação. Inicialmente é necessário que entendamos o mecanismo desta situação:

2^5 = 2 x 2 x 2 x 2 x 2 = 32, ou seja, o número (2) é multiplicado por ele mesmo e, este resultado é novamente multiplicado por ele mesmo e, a cada vez que uma multiplicação acontece, este resultado volta a sofrer a mesma operação, até que isto tenha sido executado tantas vezes quanto for o valor do expoente (5).

1ª repetição	=	2	x	2	x	2	x	2	x	2
2ª repetição	=		4		x	2	x	2	x	2
3ª repetição	=			8			x	2	x	2
4ª repetição	=				16				x	2
5ª repetição	=					32				

Ao utilizarmos uma instrução de repetição, a primeira ação é perguntarmos a nós mesmos: existe alguma operação, neste problema, que é efetuada mais de uma vez? Se sim, então CERTAMENTE teremos que empregar uma instrução de repetição (caso contrário, não há sentido em emprega-la). No caso da potenciação existe alguma operação sendo efetuada mais de uma vez? Sim – a multiplicação de um número por ele mesmo. Agora cabe uma segunda pergunta: quantas vezes esta operação é executada? Tantas vezes quanto for o número que representa o expoente! Ou seja, a operação deve ser repetida EXATAMENTE o número de vezes representado pelo expoente. Se repetirmos esta operação mais (ou menos) vezes do que o expoente, o resultado estará errado. Assim sendo, temos que contar cada operação realizada e comparar com o expoente. Se o resultado desta conta for menor que o expoente, então a operação de multiplicação deve ser efetuada novamente. O algoritmo pode ser expresso da seguinte forma:

Quadro 28 – Algoritmo para cálculo de potenciação

```
1. Início;
2. Variáveis:
   2.1. Inteiro: numero, expoente, potencia, contador;
3. Ler numero;
4. Ler expoente;
5. potencia ← 1;
6. contador ← 0;
7. Enquanto (contador < expoente)
      7.1. potencia ← potencia * numero;
      7.2. contador ← contador + 1;
8. Fim enquanto;
9. Mostrar potencia;
10. Fim.
```

É importante que as linhas 5, 6, 7 e 9 sejam detalhadas – vamos tomar como ponto de partida, que nas linhas 3 e 4 foram obtidos os valores 3 e 2, respectivamente para as variáveis `numero` e `expoente`, ou seja, queremos obter o resultado de 3^2 (que é 9):

5. A variável `potencia` é a responsável por armazenar cada resultado oriundo da multiplicação do número por ele mesmo. Até aí tudo bem, não é? Mas na primeira vez que formos efetuar esta multiplicação, o valor da variável `numero` deve ser multiplicado por qual valor? Como se trata uma multiplicação deve ser por um valor que não altere aquele resultado SOMENTE NA PRIMEIRA VEZ. Ora, que valor multiplicado por um segundo número mantém somente o resultado do segundo? A resposta é 1. Se, por exemplo, este valor fosse 0 (zero), como as operações subsequentes serão multiplicações, o resultado seria sempre 0.

6. A variável `contador` é inicializado com o valor 0 (zero). O que isto significa? Que antes de começarmos as repetições, não as efetuamos nenhuma vez, ou seja, 0 (zero).

7. A instrução desta linha indica que uma condição será testada e que, "enquanto" o resultado da mesma for verdadeiro, as instruções das linhas que estão agrupadas entre o `Enquanto` e o `Fim enquanto` serão repetidas. Inicialmente será testado se o valor inicial de `contador` é menor que o valor de `expoente`. Como `contador`, neste momento, vale 0 (zero) e `expoente`, 3, o resultado da comparação é verdadeiro e, portanto, a alinha 7.1. é executada. A instrução desta linha armazena o valor atual de `potencia` (que é 1) multiplicado pelo valor de `numero` (que é 3), na própria variável `potencia` (que, portanto, assume o valor 3). A próxima linha (7.2.) armazena o valor atual de `contador` (que, neste momento é 0) mais 1, na

própria variável `contador` (passando a valor 1). Como não existem mais linhas a serem executadas, o controle passa para o `Fim enquanto` que, por sua vez, retorna ao `Enquanto` na linha 7. Neste ponto, é efetuado novo teste com o valor atual de `contador` (que agora vale 1) e `expoente` (que, como não teve seu valor alterado, continua valendo 2). Como é verdade que 1 é menor que 2, a linha 7.1. é novamente executada, ou seja, a variável `potencia` recebe seu valor atual (que é 3) multiplicado pelo valor da variável `numero` (que também é 3), resultando em 9. A linha seguinte soma 1 ao valor que está armazenado na variável `contador` (atualmente valendo 1) e atribui este resultado (2) à própria variável `contador`. A próxima linha é o `Fim enquanto` que desvia a execução para a linha 7 que, novamente, verifica se o conteúdo da variável `contador` é menor que `expoente`. Porém, nesta vez, o resultado é falso, pois `contador` armazena o valor 2 e este valor NÃO É MENOR (é igual!) que o valor da variável `expoente` (que também vale 2). Desta forma, o controle é desviado para a linha subsequente ao comando `Fim enquanto`. Cada variável tem seu valor alterado conforme as operações, de cada linha, são efetuadas, de acordo com a sequência mostrada no Quadro 29:

Quadro 29 – Sequência de operações

Sequência	numero	expoente	potencia	contador	contador < expoente?
1	3 (linha 3)	2 (linha 4)	1 (linha 5)	0 (linha 6)	Verdadeiro (linha 7)
2			3 (linha 7.1)	1 (linha 7.2)	Verdadeiro (linha 7)
3			9 (linha 7.1)	2 (linha 7.2)	Falso (linha 7)

9. Esta linha é executada somente após o resultado da condição testada na linha 7 ter resultado falso. Sua execução mostra o valor atual da variável `potencia`. – como é possível verificar no Quadro 29, este valor é 9, ou seja 3^2!

Para testarmos este exemplo, vamos criar uma página HTML que possibilitará a digitação de um número e um expoente e os enviará a um script PHP que irá calcular e mostrar o resultado:

Quadro 30 – Página HTML para dados de potenciação

```
<HTML>
    <HEAD>
        <TITLE>Potenciação</TITLE>
    </HEAD>
    <BODY>
        <FORM ACTION = "potenciacao.php">
            Informe número<INPUT TYPE = "text" NAME = "numero"><BR>
            Informe expoente<INPUT TYPE = "text" NAME = "expoente"><BR>
            <INPUT TYPE = "submit" NAME = "enviar" VALUE = "Enviar">
            <INPUT TYPE = "reset" NAME = "limpar" VALUE = "Limpar">
        </FORM>
    </BODY>
</HTML>
```

Vamos salvar esta página HTML com o nome de `potenciacao.html` e digitar o script PHP (e salvá-lo com o nome de `potenciacao.php`):

Quadro 31 – Script PHP para cálculo de potenciação

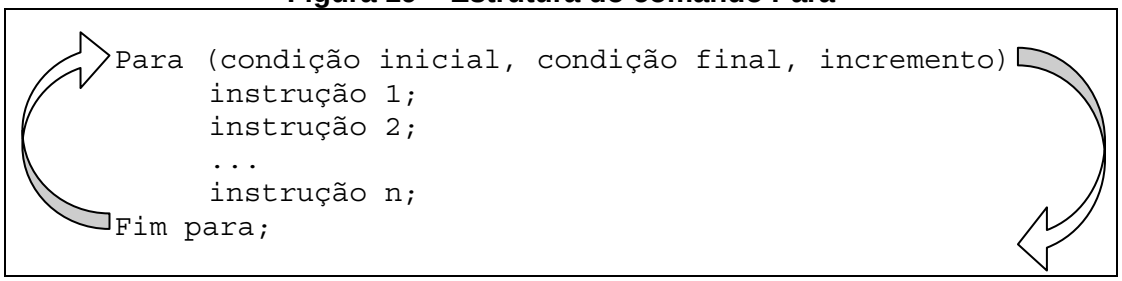
```
<?php
$numero = $_GET["numero"];
$expoente = $_GET["expoente"];
$potencia = 1;
$contador = 0;
while ($contador < $expoente)
{
    $potencia = $potencia * $numero;
    $contador = $contador + 1;
}
echo "$numero elevado a $expoente é $potencia";
?>
```

Se compararmos o script PHP com o algoritmo, é possível perceber que os comandos que serão repetidos não estão delimitados por um comando específico (em algoritmos, temos o `Fim enquanto`). Quem faz o papel de delimitar estes comandos são as chaves { }.

Evidentemente que este exemplo não considera o fato de que o expoente fornecido possa ser um número negativo. Mas esta situação é facilmente contornável utilizando um comando de decisão que realize a operação de potenciação somente com números positivos – fica o desafio!

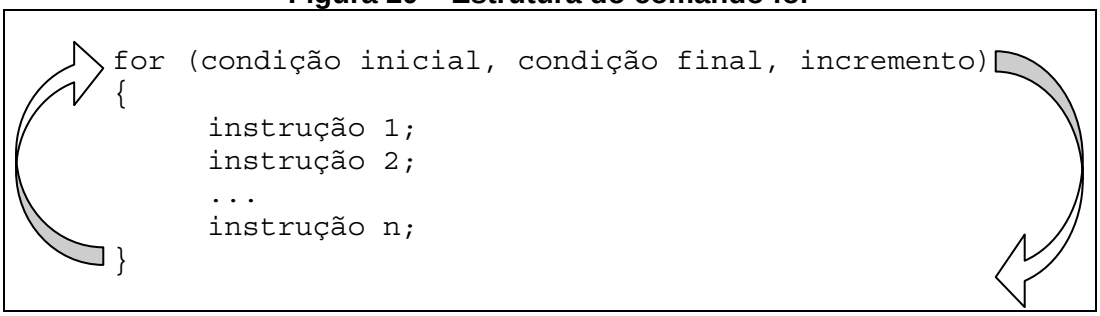
Agora que já entendemos o mecanismo de funcionamento do comando `Enquanto`, vamos para o comando `Para`. Antes de visualizarmos sua estrutura é importante sabermos que este comando tem exatamente a mesma função do `Enquanto` – como, da mesma forma, possui exatamente os mesmos elementos, porém dispostos de forma alternativa. A Figura 19 apresenta sua estrutura lógica:

Figura 19 – Estrutura do comando Para



Já em PHP, a estrutura correspondente é mostrada na Figura 20:

Figura 20 – Estrutura do comando for



Vamos compreender os elementos da estrutura do `Para`:

- `Para`: indica o início da estrutura de repetição. Este início, porém, é condicionado ao número de vezes determinado pelos argumentos do comando: `condição inicial`, `condição final`, `incremento`.
- `Condição inicial`: representa o valor inicial que determinada variável deve apresentar antes das instruções serem repetidas. Fazendo um paralelo com o exercício anterior, trata-se da linha 6, do Quadro 28.
- `Condição final`: determina a condição que permite a repetição das instruções. É exatamente a condição que seria expressa em um comando `Enquanto`. No exemplo anterior, linha 7 do Quadro 28.
- `Incremento`: indica a contagem das repetições das instruções – no comando Enquanto do exercício anterior, é a instrução da linha 7.2 do Quadro 28.
- `Instrução 1`, `2` e `n`: são as instruções que serão repetidas a cada `incremento` até que o resultado da `condição final` seja falso.
- `Fim para`: indica o final do bloco de instruções a serem repetidas.

Na verdade, trata-se de uma instrução `Enquanto`, com todos os elementos de controle desta, porém expressos em uma mesma linha! Vamos mostrar o uso da instrução `Para` para mostrarmos os oito primeiros termos da série de Fibonacci. Um pouco de

história: Leonardo Fibonacci foi um comerciante italiano que se destacou por seus estudos de matemática, tendo proposto, em 1202, uma série numérica que representava o crescimento da população de coelhos, geração após geração (o que pode parecer uma brincadeira resultou em estudos de fractais – se ficou curioso, faça uma pesquisa pelos termos “fractais” e “Fibonacci”!). Esta série é iniciada com os números 0 e 1 e, a partir daí cada novo termo é gerado a partir da soma dos dois termos anteriores. Como qualquer aplicação computacional, o primeiro passo é compreender o mecanismo do problema:

1. O primeiro termo inicia-se com o valor 0 (zero);
2. O segundo termo também é iniciado com o valor 1;
3. O terceiro termo é formado pela soma dos dois anteriores, ou seja, será 1;
4. Para gerar o próximo termo, o primeiro termo passa a ser o segundo e o segundo passa a ser o terceiro (recém calculado). A soma destes dois será 2. Esta lógica será repetida oito vezes, gerando a seguinte sequência: 0, 1, 1, 2, 3, 5, 8, 13.

Vamos ao algoritmo:

Quadro 32 – Algoritmo para gerar a série de Fibonacci

```
1. Início;
2. Variáveis:
   2.1. Inteiro: termo1, termo2, termo3;
3. termo1 = 0;
4. termo2 = 1;
5. Para (contador ← 0; contador < 8; contador++)
      5.1. Mostrar termo1;
      5.2. termo3 ← termo2 + termo1;
      5.3. termo1 ← termo2;
      5.4. termo2 ← termo3;
6. Fim para;
7. Fim.
```

É importante que alguns pontos sejam detalhados:

- Percebam que não temos nenhum comando de entrada de dados. Se o `termo1` é 0 e o `termo2` é 1 (são valores já fixados nas linhas 3 e 4), não é preciso que ninguém os forneça.
- Na linha do `Para` (linha 5), temos os seguintes argumentos: `contador ← 0; contador < 8; contador++`. O primeiro deles (`contador ← 0`) indica que ANTES de termos repetido qualquer instrução, a variável que irá controlar o número de repetições terá o valor 0 (zero) – óbvio, pois nada foi repetido ainda. O segundo argumento (`contador < 8`) expressa que o valor da variável `contador` será sempre comparado com 8 e enquanto for

verdadeiro que o valor desta esta variável é menor do que 8, os comandos desta estrutura (linhas 5.1, 5.2, 5.3 e 5.4) serão repetidos – caso contrário o controle será desviado para a linha subsequente ao `Fim para` (linha 7). O terceiro argumento (`contador++`) vai somar 1 à variável `contador` a cada repetição (poderíamos ter substituído este argumento por `contador` ← `contador + 1`).

- A linha 5.1 mostra o primeiro termo e as demais fazem exatamente como definido no passo 4 (mostrado anteriormente). Este conjunto de linhas será repetido até que o valor da variável contador seja maior ou igual a 8 (segundo parâmetro da linha 5).
- A linha do Fim para (linha 6) indica que o controle retornará á linha 5 – tal como ocorre no comando `Enquanto`.

As variáveis vão assumir, sequencialmente, os valores mostrados no Quadro 33:

Quadro 33 – Sequência de operações

Seq.	termo1	termo2	termo3	contador	contador < 8?
1	0 (linha 3)				
2		1 (linha 4)			
3				0 (linha 5)	Verdadeiro (linha 5)
4	Mostrou 0 (linha 5.1)				
5			1 (linha 5.2)		
6	1 (linha 5.3)				
7		1 (linha 5.4)			
8				1 (linha 5)	Verdadeiro (linha 5)
9	Mostrou 1 (linha 5.1)				
10			2 (linha 5.2)		
11	1 (linha 5.3)				
12		2 (linha 5.4)			
13				2 (linha 5)	Verdadeiro (linha 5)
14	Mostrou 1 (linha 5.1)				
15			3 (linha 5.2)		
16	2 (linha 5.3)				
17		3 (linha 5.4)			
18				3 (linha 5)	Verdadeiro (linha 5)
19	Mostrou 2 (linha 5.1)				
20			5 (linha 5.2)		
21	3 (linha 5.3)				
22		5 (linha 5.4)			
23				4 (linha 5)	Verdadeiro (linha 5)
24	Mostrou 3 (linha 5.1)				
25			8 (linha 5.2)		
26	5 (linha 5.3)				
27		8 (linha 5.4)			
28				5 (linha 5)	Verdadeiro (linha 5)
29	Mostrou 5 (linha 5.1)				
30			13 (linha 5.2)		
31	8 (linha 5.3)				
32		13 (linha 5.4)			
33				6 (linha 5)	Verdadeiro (linha 5)
34	Mostrou 8 (linha 5.1)				
35			21 (linha 5.2)		
36	13 (linha 5.3)				
37		21 (linha 5.4)			
38				7 (linha 5)	Verdadeiro (linha 5)
39	Mostrou 13 (linha 5.1)				
40			44 (linha 5.2)		
41	21 (linha 5.3)				
42		44 (linha 5.4)			
43				8 (linha 5)	Falso (linha 5)

Para implementar este algoritmo não teremos necessidade de desenvolver uma página HTML, visto que não é necessário que sejam fornecidos externamente dados para processados. Assim sendo, basta salvarmos o código a seguir com o nome de `fibonacci.php` e executa-lo digitando na barra de endereços de navegador, `localhost/fibonacci.php`:

Quadro 34 – Script PHP para a série de Fibonacci

```
<?php
$termo1 = 0;
$termo2 = 1;
for ($contador = 0; $contador < 8; $contador++)
{
    echo "$termo1<BR>";
    $termo3 = $termo2 + $termo1;
    $termo1 = $termo2;
    $termo2 = $termo3;
}
?>
```

Finalmente, temos que informar que, da mesma forma como nos comandos de decisão, existe mais uma estrutura de repetição – esta, com teste no final. Porém, com as estruturas apresentadas é possível desenvolver qualquer aplicativo onde seja necessário repetir trechos do programa.

Exercícios resolvidos

1. Elabore um script PHP que calcule e mostre o fatorial de um número recebido:

```
<?php
  $numero = $_GET["numero"];
  $fatorial = 1;
  for($contador = 1; $contador <= $numero; $contador++)
    $fatorial = $fatorial * $contador;
  echo "O fatorial de $numero é $fatorial";
?>
```

2. Elabore um script PHP que calcule e mostre os divisores de um número recebido:

```
<?php
  $numero = $_GET["numero"];
  for($divisor = $numero; $divisor > 0; $divisor--)
    if ($numero % $divisor == 0)
        echo $divisor."   ";
?>
```

3. Uma rainha requisitou os serviços de um monge e disse-lhe que pagaria qualquer preço. O monge, necessitando de alimentos, indagou à rainha se o pagamento poderia ser feito com grãos de trigo dispostos em um tabuleiro de xadrez, de tal forma que o primeiro quadro deveria conter apenas 1 grão e os quadros subsequentes, o dobro do quadro anterior. A rainha achou o trabalho barato e pediu que o serviço fosse executado. Faça um script PHP que calcule e mostre apenas as quantidades de grãos que estarão em cada casa branca do tabuleiro.

```
<?php
for($linha = 1; $linha <= 8; $linha++)
  for($coluna = 1; $coluna <= 8; $coluna++)
  {
    if(($linha == 1) AND ($coluna == 1))
      $quadro = 1;
    else
```

```
      $quadro = $quadro * 2;
    if (((($linha % 2 == 0) AND ($coluna % 2 == 0)) OR
        ((($linha % 2 != 0) AND ($coluna % 2 != 0)))
      echo "$linha, $coluna = $quadro<BR>";
  }
?>
```

4. Faça um script PHP que receba o valor de uma data inicial e de uma data final (no formato DD, MM e AAAA), considerando que a primeira data é anterior à segunda e mostre quantos dias se passaram entre as datas (considere apenas existência de meses com 28, 30 e 31 dias).

```
<?php
$ddi = $_GET["ddi"];
$mmi = $_GET["mmi"];
$aai = $_GET["aai"];
$ddf = $_GET["ddf"];
$mmf = $_GET["mmf"];
$aaf = $_GET["aaf"];
$mes = $mmi;
$ano = $aai;
$quantidadeMeses = (($aaf*12) - ($aai*12)) + ($mmf - $mmi);
$quantidadeDias   = (($aaf*360) + ($mmf*30) + $ddf) - (($aai*360) +
($mmi*30) + $ddi);
$contadorMeses   = 1;
while ($contadorMeses <= $quantidadeMeses)
{
  if (($mes == 1) OR ($mes == 3) OR ($mes == 5) OR ($mes == 7) OR
($mes == 8) OR ($mes == 10) OR ($mes == 12))
    $quantidadeDias = $quantidadeDias + 1;
  else if ($mes == 2)
    $quantidadeDias = $quantidadeDias - 2;
  $mes++;
  if ($mes == 13)
  {
    $mes = 1;
    $ano++;
  }
  $contadorMeses++;
}
echo $quantidadeDias;
?>
```

5. Elabore um script PHP para receber um número de uma página HTML e mostre quais são seus divisores.

```
<?php
  $numero = $_GET["numero"];
  for($divisor = $numero; $divisor > 0; $divisor--)
      if ($numero % $divisor == 0)
          echo $divisor."   ";

?>
```

9 FUNÇÕES

Em termos de comandos empregados em processamento de dados, já vimos todas as categorias. Evidentemente, cada linguagem de programação apresenta outros elementos, muitas vezes distintos entre si, e que realizam outras tarefas – ainda que as mesmas sejam possíveis de serem realizadas com os comandos que estudamos até o momento. Seja com for, é importante que saibamos empregar as instruções estudadas da maneira mais efetiva possível. Neste sentido, é importante que saibamos identificar e projetar funções.

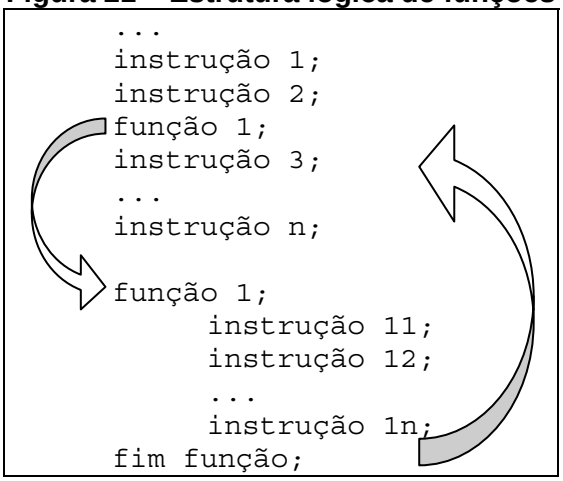
Computacionalmente falando, uma função nada mais é do que um trecho de processamento que por ser executado mais de uma vez no código, é passível de ser implementado à parte do programa principal e ser chamado (para executar a tarefa) tantas vezes quanto necessário for. É uma forma de modularizar a solução.

O emprego de funções apresenta o seguinte mecanismo:

1. São desenvolvidos dois programas: um chamado de “principal” e outro que é a “função”.
2. O programa “principal” segue seu fluxo normal até chamar a “função”.
3. Esta, por sua vez, foi implementada à parte do programa principal e só será executada se este o chamar.
4. Uma vez acionada, ela executará suas tarefas (o programa principal fica “suspenso” enquanto a função está sendo processada) e, ao finalizar, retorna para o ponto em que foi chamada no programa principal.
5. O programa principal segue seu fluxo normal até chamar (se necessário) a mesma função novamente (ou outra função implementada).

A Figura 21 ilustra o mecanismo de chamada, execução e retorno de uma função:

Figura 21 – Estrutura lógica de funções



Para mostrar o funcionamento de uma função, vamos desenvolver uma aplicação onde um usuário forneça dois números e estes são enviados a um script PHP que: (1) calcule e mostre o fatorial do primeiro número, (2) eleve o primeiro número à potência representada pelo segundo número e mostre o resultado e, (3) some o fatorial e o resultado da potenciação e mostre este valor. O Quadro 35 mostra um algoritmo que resolve este problema – ressalta-se que existem maneiras mais otimizadas de resolvê-lo, porém a forma como apresentado, é bastante ilustrativa quanto à necessidade do uso de funções:

Quadro 35 – Algoritmo sem funções

```
01. Início;
02. Variáveis:
    2.1. Inteiro: numero1, numero2, soma, resultado, contador;
03. Ler numero1;
04. Ler numero2;

05. resultado ← 1;
06. Para (contador ← 1; contador <= numero1; contador++)
07.         resultado ← resultado * contador;
08. Fim para;
09. Mostrar "O fatorial de ".numero1." é ".resultado;

10. resultado ← 1;
11. Para (contador ← 1; contador <= numero2; contador++)
12.         resultado ← resultado * numero1;
13. Fim para;
14. Mostrar numero1." elevado a ".numero2." é ".resultado;

15. soma ← resultado;

16. resultado ← 1;
17. Para (contador ← 1; contador <= numero1; contador++)
18.         resultado ← resultado * contador;
19. Fim para;

20. soma ← soma + resultado;
21. Mostrar "Por sua vez, a soma das duas operações anteriores é ".soma;
22. Fim.
```

Qual a lógica empregada na construção deste algoritmo?

- As linhas 01 a 04 são óbvias: declaramos as variáveis e demos entrada naquelas necessárias à execução das operações.
- O bloco formado pelas instruções que se encontram nas linhas 05 a 08 calcula o fatorial do `numero1`, armazenando este valor na variável `resultado`.
- A linha 09 mostra o valor de `resultado` – ou seja, o fatorial de `numero1`.
- As linhas 10 a 13 calculam a potenciação de `numero1` elevado a `numero2`, sendo que este valor é armazenado na variável `resultado`. Notem que na linha 09, esta variável guardava o valor do fatorial e, portanto, ao armazenar o valor da potenciação, perdeu-se o conteúdo anterior (fato que ocorreu já na linha 10).
- A linha 14 mostra o novo valor da variável `resultado` (potenciação).

- A variável `soma`, na linha 15, recebe o valor atual da variável `resultado` (que é o valor obtido pelas instruções das linhas 10 a 13). Por que? Para que, após obtermos NOVAMENTE o valor do fatorial (uma vez que foi perdido na linha 10), podermos adicionar este resultado ao conteúdo já existente na variável `soma` (que é a terceira operação solicitada no exercício).
- Notem com atenção que o bloco formado pelas linhas 16 a 19 apresenta EXATAMENTE as mesmas instruções das linhas 05 a 08, ou seja, obtiveram novamente o valor do fatorial – armazenado em `resultado` que, até o momento, continha o valor da operação de potenciação.
- A linha 20 adiciona à variável `soma` (que continha o resultado da potenciação – linha 15) o valor de `resultado`.
- A linha 21, finalmente, mostra a soma das operações de potenciação e fatorial.

Percebam que, apesar de funcional, no mínimo é um algoritmo ineficiente, pois requer que a mesma operação seja repetida duas vezes (linhas 05 a 08 e, 16 a 19). Agora vejamos o mesmo problema sendo solucionado a partir de um algoritmo que emprega funções:

Quadro 36 – Algoritmo com funções

```
01. Início;
02. Variáveis:
    2.1. Inteiro: numero1, numero2, soma;
03. Ler numero1;
04. Ler numero2;
05. Mostrar "O fatorial de ".numero1." é ".fatorial(numero1);
06. Mostrar "Já ".numero1. " elevado a ".numero2. " é ".potencia(numero1, numero2);
07. soma ← fatorial(numero1) + potencia(numero1, numero2);
08. Mostrar "Por sua vez, a soma das duas operações anteriores é ".soma;

09. Função fatorial(Inteiro: numero)
10. Variáveis:
    10.1. Inteiro: resultado, contador;
11.    resultado = 1;
12.    Para(contador ← 1; contador <= numero; contador++)
13.        resultado ← resultado * contador;
14.    Fim para;
15.    Retornar resultado;
16. Fim função;

17. Função potencia(Inteiro: numero1, numero2)
18. Variáveis:
    18.1. Inteiro: resultado, contador;
19.    resultado = 1;
20.    Para(contador ← 1; contador <= numero2; contador++)
21.        resultado = resultado * numero1;
22.    Fim para;
23.    Retornar resultado;
24. Fim função;

25. Fim.
```

Este algoritmo funciona da seguinte maneira:

- Da mesma forma como no algoritmo anterior, as linhas de 01 a 04, as variáveis são declaradas e os valores obtidos. Notem que na declaração de variáveis, apenas identificamos o `numero1`, `numero2` e `soma`. São as variáveis que serão empregadas no programa principal (que vai da linha 01 a 08).

- A linha 05 apresenta um comando de saída de dados. Inicialmente mostrará uma frase (`O fatorial de ".numero1." é "`) e logo após, chama a função que irá calcular o fatorial do `numero1`. É importante atentar para os seguintes detalhes: (1) o nome da função chamada na linha 05 deve ser exatamente o nome da função declarada na linha 09; (2) o fato da variável `numero1` estar entre parênteses na chamada da função, indica que estamos passando um parâmetro para a função `fatorial` – que parâmetro é este? Ora, é exatamente o número que desejamos extrair o fatorial! Ou seja, a função

recebe o parâmetro (linha 05), vai para a linha 09, executa todas as instruções até encontrar seu final (linha 16) e volta no ponto de chamada que, por sua vez, está concatenada a uma instrução que irá mostrar algo (`Mostrar "O fatorial de ".numero1." é ".fatorial(numero1);`). Isto indica que após a execução da função, esta irá armazenar um resultado que será mostrado pela instrução de saída de dados. É como se a função, neste momento, desempenhasse o papel de uma variável.

- A linha 06 apresenta exatamente o mesmo mecanismo da linha anterior, com o detalhe de que está sendo passado, à função `potencia`, dois parâmetros ao invés de somente um.
- A linha 07 merece especial atenção. Vejam que a variável `soma` está armazenando o valor retornado pela função fatorial (calculado a partir do valor da variável numero1) adicionado ao valor retornado pela função `potencia` (calculado sobre os valores de `numero1` e `numero2`). O que está acontecendo? As instruções que efetuam o cálculo do fatorial são executadas novamente (na primeira vez foram executadas a partir da linha 05), da mesma forma ocorre com as instruções que calculam a potenciação (executadas anteriormente na linha 06). Está é uma das vantagens de se utilizar funções, pois não é necessário replicar os comandos a cada vez que forem necessários – basta implementa-los uma única vez, sob a forma de função, e sempre que for necessário executa-lo, é só fazer uma chamada a ela!
- A linha 08 mostra o resultado armazenado em `soma` e o algoritmo principal é finalizado! Notem que a partir deste ponto, são implementadas somente as funções que serão chamadas pelo algoritmo – tantas quantas forem necessárias.
- A linha 09 dá início à função `fatorial`. Vejam que ela é calculada a partir de um parâmetro recebido (linhas 05 e 07) e armazenado em uma variável inteira chamada de `numero`. Notem, porém, que tanto na linha 05, quanto na linha 09, a variável enviada chama-se `numero1`. O que isto significa? Significa que não importa o nome da variável que esteja passando o parâmetro na chamada – quando a função receber este valor, o mesmo será armazenado em uma variável que pode ter outro nome (ou não – veremos isto na próxima função).

- Nas linhas 10 e 10.1, são declaradas as variáveis que a função fatorial irá manipular durante sua execução – o algoritmo principal não as conhece. Na verdade, uma função é como se fosse um subalgoritmo, ou um algoritmo dentro de outro algoritmo.
- Na linha 11, inicializamos a variável `resultado` com o valor 1. Como esta variável irá "sofrer" sucessivas multiplicações, ela deve iniciar com um valor neutro para estas operações (no caso, 1). Se vocês não estão lembrados, o fatorial de 6, por exemplo é 6! = 1 x 2 x 3 x 4 x 5 x 6.
- Como são sucessivas multiplicações, a linha 12 apresenta um comando de repetição, indicando que a variável que irá controlar estas repetições (`contador`) inicia-se valendo 1 (`contador ← 1`); terá seu valor alterado enquanto for menor ou igual a `numero` (`contador <= numero`) que é justamente o valor lido na linha 03 e repassado à função nas linhas 05 e 07 e; finalmente, a cada repetição, seu valor será incrementado em um unidade (`contador++`).
- Na linha 13, a variável `resultado` (cujo valor inicial é 1) receberá seu próprio valor multiplicado pelo conteúdo da variável `contador` (que também, na primeira repetição também vale 1).
- Na linha 14, após a linha 13 ter sido executada, o controle retorna à linha 12, onde a variável `contador` é incrementada em uma unidade, passando a valer 2. Ainda na linha 12, é verificado se o valor atual de `contador` é menor ou igual ao valor de `numero`. Se esta condição for verdadeira, a linha 13 é novamente executada, porém desta vez, a variável `contador` estará valendo 2, que será multiplicado pelo valor de `resultado` (que ainda é 1) e armazenado na própria variável `resultado` – ou seja, ficará com o valor 2. O processo é efetuado tantas vezes quanto for o valor armazenado em `numero`.
- A linha 15 só será executada quando o resultado do teste da linha 12 for falso. Neste caso, a função `fatorial` irá retornar (à linha onde foi chamada) o valor armazenado na variável `resultado`.
- A linha 16 indica que a função `fatorial` é finalizada.
- As linhas 17 a 24 apresentam o mesmo mecanismo descrito nas linhas 09 a 16 (contudo, a lógica para a potenciação é outra – já mostrada na seção anterior). Porém, é importante notar que as variáveis que irão armazenar os

parâmetros repassados, ao contrário da função anterior, apresentam os mesmo nomes das variáveis originais (no algoritmo principal). Isto significa que na função estas são outras variáveis que, coincidentemente, possuem o mesmo nome. Sejam quais forem os valores que estas variáveis armazenarão na função NÃO IRÃO AFETAR AS VARIÁVEIS ORIGINAIS. Por sua vez, AS VARIÁVEIS ORIGINAIS SOMENTE AFETAM AS VARIÁVEIS DA FUNÇÃO, QUANDO DA PASSAGEM DE PARÂMETROS.

- A linha 25 finaliza todo o algoritmo.

Vamos implementar o algoritmo, primeiramente, pela página HTML que irá receber os valores de `numero1` e `numero2`, conforme mostrado no Quadro 37:

Quadro 37 – Página HTML que recebe dados para funções

```
<HTML>
    <HEAD>
        <TITLE>Funções</TITLE>
    </HEAD>
    <BODY>
        <FORM ACTION = "funcoes.php">
            Informe número 1<INPUT TYPE = "text" NAME = "numero1"><BR>
            Informe número 2<INPUT TYPE = "text" NAME = "numero2"><BR>
            <INPUT TYPE = "submit" NAME = "enviar" VALUE = "Enviar">
            <INPUT TYPE = "reset" NAME = "limpar" VALUE = "Limpar">
        </FORM>
    </BODY>
</HTML>
```

Recomenda-se salvar a página HTML com o nome de `funcoes.html`. Já o algoritmo, propriamente dito, está traduzido em um script PHP com o nome de `funcoes.php`, cujos comandos empregados são mostrados a seguir (o número das linhas não deve ser digitado):

Quadro 38 – Script PHP para funções

```
01. <?php
02. $numero1 = $_GET["numero1"];
03. $numero2 = $_GET["numero2"];

04. echo "O fatorial de $numero1 é ".fatorial($numero1)."<BR>";
05. echo " $numero1 elevado à $numero2 é ".potencia($numero1, $numero2)."<BR>";
06. $soma = fatorial($numero1) + potencia($numero1, $numero2);
07. echo "Por sua vez, a soma das duas operações anteriores é $soma";

08. function fatorial($numero)
09. {
10.    $resultado = 1;
11.    for($contador = 1; $contador <= $numero; $contador++)
12.        $resultado = $resultado * $contador;
13.    return $resultado;
14. }

15. function potencia($numero1, $numero2)
16. {
17.     $resultado = 1;
18.     for($contador = 1; $contador <= $numero2; $contador++)
19.         $resultado = $resultado * $numero1;
20.     return $resultado;
21. }
22. ?>
```

Alguns detalhes merecem ser discutidos:

- Como o PHP não requer declaração de variáveis é óbvio que as funções também não empreguem este tipo de sintaxe.
- O conjunto de instruções de cada função é delimitado por {} (chaves), conforme visto nas linhas 09 a 14 e 16 a 21.
- Percebam que os comandos `for` (linhas 11 e 18) não apresentam {} (chaves) indicando que um bloco de instruções serão repetidas. Nestes casos, as chaves são opcionais pois somente um único comando será repetido "`$resultado = $resultado * $contador;`" na linha 12 e "`$resultado = $resultado * $numero1;`" na linha 19.

Basta executar a página HTML (`localhost/funcoes.html`) e testar a aplicação. Vale registrar que abordamos a aplicação de funções em sua sintaxe mais completa – que é por passagem de parâmetros. Porém não é a única forma de empregarmos funções, visto que podemos fazê-la sem o retorno de resultados. A partir do exemplo anterior, vamos supor que desejamos apenas obter a potenciação e o fatorial. O script PHP conterá os seguintes comandos:

Quadro 39 – Script PHP para funções sem retorno

```
01. <?php
02. $numero1 = $_GET["numero1"];
03. $numero2 = $_GET["numero2"];

04. fatorial($numero1);
05. potencia($numero1, $numero2);

06. function fatorial($numero)
07. {
08.     $resultado = 1;
09.     for($contador = 1; $contador <= $numero; $contador++)
10.        $resultado = $resultado * $contador;
11.     echo "O fatorial de $numero é ".$resultado."<BR>";
12. }

13. function potencia($numero1, $numero2)
14. {
14.     $resultado = 1;
16.     for($contador = 1; $contador <= $numero2; $contador++)
17.         $resultado = $resultado * $numero1;
18.     echo "$numero1 elevado a $numero2 é ".$resultado;
19. }
20. ?>
```

Percebam que ambas as funções são chamadas pelos próprios nomes (linha 04 e 05), sem uma instrução que as antecedem. Da mesma forma, nenhuma das funções apresenta o comando `return`, o que indica que valores não serão retornados. Como funciona?

- Na linha 04, a função `fatorial` é chamada, passando o conteúdo da variável `$numero1` (obtido na linha 02);
- O controle é desviado para a linha 06 e executado até a linha 12 (cujo resultado é mostrado na própria função), quando então volta à linha subsequente à chamada (linha 05);
- Semelhante situação ocorre na linha 05.

Existe ainda outra forma de trabalharmos com funções: sem passagem de parâmetros e sem retornos, conforme mostrado no Quadro 40:

Quadro 40 – Script PHP para funções sem passagem de parâmetros

```
01. <?php
02. fatorial();
03. potencia();

04. function fatorial()
05. {
06.     $numero1 = $_GET["numero1"];
07.     $resultado = 1;
08.     for($contador = 1; $contador <= $numero1; $contador++)
09.         $resultado = $resultado * $contador;
10.     echo "O fatorial de $numero1 é ".$resultado."<BR>";
11. }

12. function potencia()
13. {
14.     $numero1 = $_GET["numero1"];
15.     $numero2 = $_GET["numero2"];
16.     $resultado = 1;
17.     for($contador = 1; $contador <= $numero2; $contador++)
18.        $resultado = $resultado * $numero1;
19.    echo "$numero1 elevado a $numero2 é ".$resultado;
20. }
21. ?>
```

Percebam que, neste caso, apenas chamamos as funções (linhas 02 e 03) sem passar nenhum parâmetro – as próprias funções são encarregadas de receber os valores que necessitam para desempenhar as tarefas (linha 06 na função `fatorial` e, linhas 14 e 15 na função `potencia`).

A ideia geral ao empregar funções, em processamento de dados, é utilizar o bom senso: se tivermos que repetir trechos de instruções, então fica muito mais fácil implementa-los separadamente e, sempre que forem necessários, fazer a chamada. Outro detalhe a favor do uso de funções é que se for necessário que sejam alteradas, fazemos isto somente uma única vez na própria função e esta alteração é refletida no programa todo – sempre que for chamada.

Exercícios resolvidos

1. Elabore um script PHP para mostrar a soma de uma quantidade de termos, informada pelo usuário, da seguinte série:

   $$Soma = \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + \frac{x^{termos}}{termos!}$$

   O valor de *x* também é informado pelo usuário.

```
<?php
$termos = $_GET["termos"];
$x      = $_GET["x"];
$soma   = 0;

for($contador = 2; $contador <= $termos; $contador++)
{
```

```
  $divisao = potencia($x, $contador)/fatorial($contador);
  echo potencia($x, $contador)."/".fatorial($contador)." = "
  .$divisao."<BR>";
  $soma = $soma + $divisao;
}
echo "A soma dos termos é $soma";

function potencia($base, $expoente)
{
  $resultado = 1;
  for($contador = 1; $contador <= $expoente; $contador++)
    $resultado = $resultado * $base;
  return $resultado;
}

function fatorial($numero)
{
  $resultado = 1;
  for($contador = 1; $contador <= $numero; $contador++)
    $resultado = $resultado * $contador;
  return $resultado;
}
?>
```

2. Você conseguiria fazer o exercício anterior, sem utilizar comandos de repetição nas funções?

```
<?php
$termos = $_GET["termos"];
$x      = $_GET["x"];
$soma   = 0;

for($contador = 2; $contador <= $termos; $contador++)
{
  $divisao = potencia($x, $contador)/fatorial($contador);
  echo potencia($x, $contador)."/".fatorial($contador)." = "
  .$divisao."<BR>";
  $soma = $soma + $divisao;
}
echo "A soma dos termos é $soma";

function potencia($base, $expoente)
{
  if($expoente == 0)
    return 1;
  return potencia($base, $expoente - 1) * $base;
}

function fatorial($numero)
{
  if($numero == 1)
    return $numero;
  return fatorial($numero - 1) * $numero;
}
?>
```

É o que chamamos de função recursiva, ou seja, a própria função passa parâmetros para ela mesma (`return fatorial($numero - 1) * $numero;`), encerrando esta

operação até que determinada condição seja verdadeira (`if($numero == 1) return $numero;`).

3. Qual a diferença de funções com e sem passagem de parâmetros?

   As funções com passagem de parâmetros recebem, do fluxo principal, os valores a partir do quais a função será processada e, naquelas sem passagem de parâmetros, estes valores são obtidos pela própria função.

4. Qual a diferença de funções com e sem retorno?

   A função com retorno devolve um valor ao fluxo principal que a chamou; já as sem retorno, mantém os valores processados na própria função.

5. Qual a relação existente entre os valores passados às funções (pelo fluxo principal) e os valores recebidos pelas funções?

   Existe uma relação posicional, ou seja, o primeiro parâmetro passado será armazenado no primeiro parâmetro recebido, independentemente dos nomes utilizados nos parâmetros.

10 CONSIDERAÇÕES FINAIS

De modo geral, este capítulo apresentou as estruturas básicas empregadas em processamento de dados. Ressalta-se que a grande maioria das linguagens de programação apresentam estas estruturas tais como aqui demonstradas. Outras apresentam estruturas próprias, enquanto uma parte disponibiliza várias funções que oferecem uma gama maior de possibilidades de processamento. Porém, com as estruturas aprendidas, temos condições de implementar grande parte de soluções computacionais.

Para saber mais, indicamos alguns livros. Caso sua curiosidade seja de:

Aprofundamento em algoritmos:

PIVA JUNIOR, Dilermando; ENGELBRECHT, Angela de Mendonça; NAKAMITI, Gilberto Shigueo; BIANCHI, Francisco. **Algoritmos e programação de computadores**. Rio de Janeiro: Elsevier, 2012.

Praticar mais exercícios:

LOPES, Anita; GARCIA, Guto. **Introdução à programação – 500 algoritmos resolvidos**. Rio de Janeiro: Campus, 2002.

Aprofundamento em PHP:

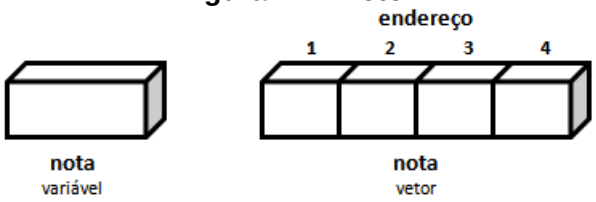
NIEDERAUER, Juliano. **PHP para quem conhece PHP**. 3 ed. São Paulo: Novatec, 2008.

Aprofundamento em outras linguagens de programação:

DEITEL, Paul; DEITEL, Harvey. **Java – Como programar**. 8 ed. São Paulo: Pearson, 2010.

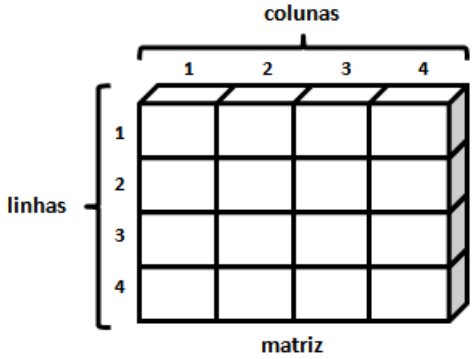
De qualquer forma, seu próximo passo em processamento de dados é estudar as estruturas de dados. Vocês devem ter percebido que, neste capítulo, somente empregamos variáveis que armazenavam apenas um único valor a cada execução das instruções de entrada e atribuição. Caso fosse necessário armazenar novos valores (e manter os valores originais) teríamos que declarar nova variável. Dependendo o número de dados a serem armazenados, esta tarefa se tornaria muito volumosa – imaginem quantas variáveis seriam necessárias para guardar todos os salários dos funcionários de uma multinacional! Esta questão é resolvida com uma estrutura particular de dados chamada de vetores. Trata-se, nada mais, do que uma variável dividida em tantos espaços quantos necessários e identificados por um endereço (ou índice), conforme mostrado na Figura 22:

Figura 22 – Vetor



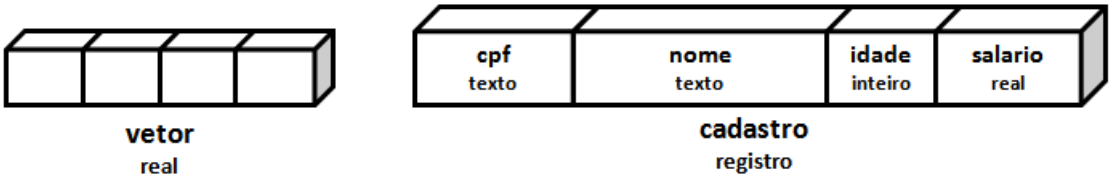
A Figura 22 mostra um vetor de uma dimensão – visualmente, os valores de notas estão guardados horizontalmente. Porém, é possível que esta estrutura tenha duas (ou mais dimensões). Neste caso, são comumente chamadas de matrizes, sendo que cada espaço destinado a armazenar um valor, é identificado pela coordenada formada pela linha e coluna, de acordo com a Figura 23:

Figura 23 – Matriz



Apesar da possibilidade de armazenarmos (e recuperarmos) vários valores nestas estruturas, as mesmas apresentam a limitação de que estes valores devem ser sempre da mesma natureza: ou somente Real, ou somente Inteiro... Porém, existe uma estrutura, chamada de registro, que possibilita que variáveis de naturezas distintas possam ser agrupadas e tratadas conjuntamente. A Figura 24 mostra um registro, comparado a um vetor:

Figura 24 – Registro



Dominando estas estruturas, já estaremos preparados para o estudo dos bancos de dados...

Seja como for, gerenciar informações requer conhecimentos, no mínimo, básicos em processamento de dados, visto que é a forma mais rápida de obtermos as informações a partir de dados. Além da rapidez, outro fator a ser considerado é o próprio registro da forma como as informações foram obtidas (algoritmo) – desde que adequadamente projetados. Neste sentido, a tecnologia alinhada ao fator humano é indispensável.