mistrovství

PHP a MySQL

Luke Welling
Laura Thomson

Kompletní průvodce vývojáře

- Návrh a tvorba webové databáze
- Internacionalizace a lokalizace
- Propojení PHP a JavaScriptu
- Autentizace a personalizace uživatelů
- Zabezpečení webové aplikace
- Zpracování chyb a výjimek
- Ladění a logování

computer press

Mistrovství PHP a MySQL

Vyšlo také v tištěné verzi

Objednat můžete na
www. computerpress.cz
www.albatrosmedia.cz

Luke Welling a Laura Thomson

Mistrovství PHP a MySQL – e-kniha

ALBATROS MEDIA a.s.

Luke Welling
Laura Thomson

Mistrovství PHP a MySQL

Computer Press
Brno
2017

Mistrovství PHP a MySQL

Luke Welling, Laura Thomson

Překlad: Ondřej Baše
Obálka: IMIDEA
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek

Authorized translation from the English language edition, entitled PHP AND MYSQL WEB DEVELOPMENT, 5th Edition by LUKE WELLING; LAURA THOMSON, published by Pearson Education, Inc, publishing as Addison-Wesley Professional, Copyright © 2017 by Pearson Education, Inc.

CZECH language edition published by Computer Press, an imprint of ALBATROS MEDIA A.S, Copyright © 2017

Translation © Ondřej Baše, 2017

Objednávky knih:
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN tištěné verze 978-80-251-4892-1
ISBN e-knihy 978-80-251-4901-0 (1. zveřejnění, 2017)

Cena uvedená výrobcem představuje nezávaznou doporučenou spotřebitelskou cenu.

Vydalo nakladatelství Computer Press v Brně roku 2017 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 31 420.

1. vydání

ALBATROS MEDIA a.s.

Stručný obsah

Obsah

Kapitola 4

Autoři

Laura Thomson je technickou ředitelkou ve společnosti Mozilla Corporation. V minulosti bývala ředitelkou ve společnostech OmniTI a Tangled Web Design a pracovala pro univerzitu RMIT University a společnost Boston Consulting Group. Je držitelkou bakalářského titulu aplikované informatiky a informačních technologií, přičemž absolvovala s vyznamenáním. Ve svém volném čase ráda jezdí na koni, diskutuje o softwaru s otevřeným zdrojovým kódem a spí.

Luke Welling pracuje jako softwarový analytik. Pravidelně přednáší o tématech otevřeného zdrojového kódu a webového vývoje na konferencích, jako jsou například OSCON, ZendCon, MySQLUC, PHPCon, OSDC a LinuxTag. Pracoval také ve společnostech OmniTI, Hitwise.com, MySQL AB a jako nezávislý konzultant ve společnosti Tangled Web Design. Přednášel o informatice na univerzitě RMIT University ve městě Melbourne v Austrálii a je držitelem bakalářského titulu aplikované informatiky. Ve svém volném čase se snaží zdokonalit svou nespavost.

Přispěvatelé

Julie C. Meloni je produktovou manažerkou a technickou konzultantkou. Žije ve městě Washington, D.C. Napsala několik knih a článků o webových programovacích jazycích a databázích, a to včetně bestselleru *Sams Teach Yourself PHP, MySQL and Apache All in One*.

John Coggeshall vlastní společnost Internet Technology Solutions, LLC, celosvětovou konzultantskou společnost v oblastech internetu a jazyka PHP, a také společnost CoogleNet, která poskytuje síť WiFi. V minulosti býval členem týmu Zend Technologies' Global Services. S jazykem PHP se seznámil v roce 1997 a napsal čtyři knihy a více než 100 článků o PHP technologiích.

Jennifer Kyrnin pracuje jako webová designérka od roku 1995. Napsala kupříkladu knihy *Sams Teach Yourself Bootstrap in 24 Hours*, *Sams Teach Yourself Responsive Web Design in 24 Hours* a Sams Teach Yourself HTML5 Mobile Application Development in 24 Hours.

Úvod

Vítejte, v této knize najdete souhrn našich znalostí, které jsme nabyli používáním jazyka PHP a databázového systému MySQL – jedná se o dva nedůležitější a nejpoužívanější nástroje pro webový vývoj.

Proč byste si měli přečíst tuto knihu?

V této knize se naučíte vytvářet interaktivní webové aplikace od nejjednoduššího objednávkového formuláře až po složité zabezpečené webové aplikace. A co víc – dozvíte se, jak toho dosáhnout s pomocí technologií s otevřeným zdrojovým kódem.

Tato kniha je určena čtenářům, kteří už znají základy jazyka HTML a programovali v některém moderním programovacím jazyku, přičemž nemuseli nutně programovat pro web nebo používat relační databáze. Jestliže začínáte programovat, bude pro vás tato kniha stále užitečná, ale proniknout do ní vám může trvat déle. Snažili jsme se nevynechat žádnou základní koncepci, ale i tak je pouze stručně shrnujeme. Typickými čtenáři této knihy jsou programátoři, kteří chtějí ovládnout jazyk HTML a databázový systém MySQL za účelem tvorby rozsáhlých komerčních webových stránek. Pokud již programujete v některém jiném webovém programovacím jazyku, nemělo by vám v rychlém přečtení této knihy nic překážet.

První vydání této knihy jsme napsali, protože nás unavovaly knihy o jazyce PHP, které byly pouhými referencemi základních funkcí. Ačkoliv jsou tyto knihy užitečné, nijak vám nepomůžou, když za vámi přijde šéf a řekne: „Vytvořte mi nákupní košík.“ Snažili jsme se co nejvíce zjednodušit všechny příklady v této knize. Spoustu ukázkových zdrojových kódu můžete přímo použít ve svých webových stránkách, řadu jiných můžete použít jen s drobnými úpravami.

Co se naučíte v této knize?

Po přečtení této knihy budete umět vytvářet skutečné dynamické webové aplikace. Jestliže jste vytvořili nějaké statické webové stránky v jazyce HTML, znáte omezení tohoto přístupu. Statický obsah z čistých HTML stránek je prostě statický. Zůstává stejný, dokud ho fyzicky nezměníte. Uživatelé nemůžou interagovat s takovými webovými stránkami žádným smysluplným způsobem.

S pomocí skriptovacího jazyka (například PHP) a databázového systému (například MySQL) můžete udělat své webové stránky dynamické tak, aby byly přizpůsobitelné a obsahovaly aktuální informace.

Tato kniha se zaměřuje na aplikace z reálného světa, a to dokonce i v úvodních kapitolách. Nejprve se podíváte na jednoduché systémy a potom se budete propracovávat různými částmi jazyka PHP a databázového systému MySQL.

Posléze se dozvíte o jednotlivých aspektech bezpečnosti a autentizace, jelikož jsou nedílnou součástí skutečných webových stránek, a také zjistíte, jak je implementovat v jazyce PHP a databázi MySQL. Kromě toho se seznámíte s integrací frontendových a backendových technologií – objevíte jazyk JavaScript a roli, jakou může hrát při vývoji vašich aplikací.

V poslední části této knihy se naučíte, jak přistupovat ke skutečným projektům – projdete si fázemi návrhu, plánování a budování níže uvedených projektů:

- Uživatelská autentizace a personalizace
- Webový e-mail
- Integrace se sociálními sítěmi

Všechny tyto projekty můžete zužitkovat tak, jak jsou. Případně je můžete přizpůsobit svým požadavkům. Vybrali jsme je, protože se domníváme, že se jedná o nejběžnější webové aplikace. Pokud potřebujete něco jiného, tato kniha by vám měla pomoct v dosažení vašich cílů.

Co je PHP?

PHP je skriptovací jazyk na straně serveru, který byl navržený speciálně pro web. Do stránky HTML můžete vkládat kód jazyka PHP, jenž se provede při každé návštěvě této stránky. Váš kód jazyka PHP je interpretován webovým serverem, přičemž generuje dokument HTML nebo jiný výstup, který uvidí návštěvník.

Jazyk PHP vznikl v roce 1994. Původně se jednalo o výsledek práce jediného muže – Rasmuse Lerdorfa. Postupně se přidávali k jeho vývoji další talentovaní lidé a z velké části ho přepsali, až se z něj stal rozšířený vyspělý produkt, jakým je dnes. Podle Grega Michillieho ze společnosti Google poháněl tento jazyk v květnu 2013 více než tři čtvrtiny všech webových stránek, přičemž toto číslo vzrostlo na více než 82 % v červenci 2016.

PHP je projekt s otevřeným zdrojovým kódem, což znamená, že máte přístup k jeho zdrojovému kódu, můžete ho používat, měnit a distribuovat ho.

PHP bylo původně zkratkou pro Personal Home Page, ale její význam se změnil po vzoru rekurzivní konvence pojmenování GNU (GNU = Gnu's Not Unix) – nyní znamená PHP Hypertext Preprocessor.

Aktuální hlavní verzí jazyka PHP je verze 7. V této verzi se jazyk dočkal kompletního přepsání jádra Zend a několika podstatných vylepšení. Veškerý zdrojový kód pro tuto knihu je otestovaný v poslední verzi PHP 7 a také v poslední verzi z rodiny verzí PHP 5.6, které jsou v současnosti stále oficiálně podporované.

Domovská stránka jazyka PHP je k dispozici na internetové adrese *http://php.net/*.

Domovská stránka společnosti Zend Technologies se nachází na internetové adrese *http://www.zend.com/*.

Co je MySQL?

MySQL je velmi rychlý a robustní **systém pro správu relačních databází** (RDBMS). Databáze umožňuje efektivně ukládat, vyhledávat, řadit a načítat data. MySQL server řídí přístup k vašim datům. Zajišťuje, aby k datům mohlo přistupovat více uživatelů současně, nabízí rychlý přístup a dohlíží na to, aby přístup získali pouze oprávnění uživatelé. MySQL je tudíž server s podporou více uživatelů a více vláken. Používá jazyk SQL (Structured Query Language) – standardní databázový dotazovací jazyk. MySQL je veřejně dostupný od roku 1996, ale počátek jeho vývoje sahá až do roku 1979. Jedná se o celosvětově nejoblíbenější databázi s otevřeným zdrojovým kódem, která mnohokrát vyhrála cenu Volba čtenářů časopisu Linux Journal.

MySQL je k dispozici pod dvěma licencemi. Můžete si zvolit volně dostupnou licenci s otevřeným zdrojovým kódem (GPL), jestliže splníte její podmínky. Pokud chcete vydat aplikaci s jinou licencí než GPL, která bude zahrnovat MySQL, můžete si koupit komerční licenci.

Proč používat PHP a MySQL?

Když se rozhodnete, že budete vytvářet webové stránky, můžete si vybírat z mnoha různých produktů.

Musíte si vybrat:

- kde budete chtít provozovat svůj webový server – v cloudu, na soukromých virtuálních serverech nebo na skutečném hardwaru,
- operační systém,
- webový server,
- systém pro správu databází nebo jiné datové úložiště,
- programovací nebo skriptovací jazyk.

Můžete skončit s hybridní architekturou s více datovými úložišti. Některé volby závisejí na jiných. Například ne všechny operační systémy běží na všech typech hardwaru, ne všechny webové servery podporují všechny programovací jazyky atd.

Tato kniha nevěnuje příliš pozornosti hardwaru, operačním systémům ani webovým serverům. Není to ani nutné. Na jazyku PHP a systému MySQL je krásné, že fungují na všech nejrozšířenějších operačních systémech, a dokonce na řadě méně známých.

Většinu kódu v jazyce PHP lze napsat tak, aby byl přenositelný mezi operačními systémy a webovými servery. Existuje několik funkcí tohoto jazyka, které jsou závislé na systému souborů (jenž závisí na operačním systému), ale ty jsou v dokumentaci i v této knize řádně označené.

Ať už si vyberete jakýkoliv hardware, operační systém nebo webový server, měli byste vážně uvažovat, že si zvolíte jazyk PHP a systém MySQL.

Několik předností jazyka PHP

Mezi hlavní konkurenty jazyka PHP patří jazyk Python, Ruby, Perl, Java, prostředí Node.js a Microsoft .NET.

V porovnání s těmito produkty má jazyk PHP několik předností:

- výkon,
- škálovatelnost,
- rozhraní pro spoustu různých databázových systémů,
- vestavěné knihovny pro většinu běžných webových úloh,
- nízká cena,
- snadnost učení a používání,
- silná podpora objektově orientovaného programování,
- přenositelnost,
- flexibilita vývojového přístupu,
- dostupnost zdrojového kódu,
- dostupnost podpory a dokumentace.

Následuje podrobnější informace o jeho přednostech.

Výkon

PHP je velmi rychlý jazyk. S pomocí jediného levného serveru můžete zpracovat miliony návštěv denně. Umí obsluhovat vše – od malého kontaktního formuláře až po webové stránky typu Facebook nebo Etsy.

Škálovatelnost

PHP má něco, co Rasmus Lerdorf často označuje jako architektura SN (sdílené nic). To znamená, že ho můžete efektivně a levně implementovat do velkého množství webových serverů.

Integrace s databází

PHP nativně podporuje spoustu databázových systémů. Kromě databáze MySQL se můžete připojovat kupříkladu k databázím PostgreSQL, Oracle, MongoDB a MSSQL. Verze PHP 5 a PHP 7 rovněž nabízejí SQL rozhraní pro běžné soubory, které se nazývá SQLite.

Prostřednictvím standardu ODBC (Open Database Connectivity) se můžete připojovat k libovolné databázi přes ovladač ODBC. Řadí se k nim především produkty společnosti Microsoft, ale také spousta jiných.

Kromě nativních knihoven poskytuje jazyk PHP rovněž abstraktní vrstvu databázových objektů (PDO), jež přináší konzistentní přístup a prosazuje postupy bezpečného programování.

Vestavěné knihovny

Protože jazyk PHP byl navržený pro použití na webu, obsahuje spoustu vestavěných funkcí, které vykonávají mnoho užitečných webových úloh. Můžete dynamicky generovat obrázky, připojovat se k webovým službám a jiným síťovým službám, parsovat kód jazyka XML, odesílat e-maily, pracovat s cookies a generovat dokumenty PDF, a to vše jen s pomocí několika řádků kódu.

Cena

PHP je k dispozici zdarma. Jeho poslední verzi si můžete kdykoliv stáhnout na internetové adrese *http://www.php.net* bez poplatků.

Snadnost učení jazyka PHP

Syntaxe jazyka PHP vychází z jiných programovacích jazyků – zejména z jazyků C a Perl. Pokud už znáte jazyk C nebo Perl nebo jiný programovací jazyk podobný jazyku C, například jazyk C++ nebo Java, budete produktivní s jazykem PHP téměř okamžitě.

Podpora objektově orientovaného programování

Jazyk PHP ve verzi 5 přinesl skvěle navržené objektově orientované prvky, které byly dále vylepšeny ve verzi 7. Jestliže umíte programovat v jazyce Java nebo C++, najdete zde prvky, které už znáte – kupříkladu dědičnost, soukromé a chráněné vlastnosti a metody, abstraktní třídy a metody, rozhraní, konstruktory a destruktory. Objevíte v něm dokonce i méně obvyklé prvky, jako jsou například iterátory nebo traity.

Přenositelnost

PHP je k dispozici pro spoustu různých operačních systémů. Psát kód v jazyce PHP je možné ve volně dostupných unixových operačních systémech (například Linuxu nebo FreeBSD), v komerčních unixových systémech, v systému OS X a v systému Windows.

Flexibilita vývojového přístupu

V jazyce PHP můžete vyřešit jednoduché úkoly jednoduše, a stejně tak jednoduše můžete vytvářet obrovské aplikace s pomocí některého frameworku založeného na architektuře MVC (Model-View-Controller).

Zdrojový kód

Máte přístup ke zdrojovému kódu jazyka PHP. Pokud chcete něco upravit nebo přidat do tohoto jazyka, můžete to udělat.

Nemusíte čekat na to, až jeho výrobce vydá novou verzi. Nemusíte se obávat, že by výrobce přestal podnikat nebo se rozhodl přestat podporovat svůj produkt.

Dostupnost podpory a dokumentace

Společnost Zend Technologies (*http://www.zend.com/*), která stojí za vývojem jádra jazyka PHP, financuje svůj vývoj tak, že nabízí placenou podporu a související software.

Dokumentace a komunita okolo jazyka PHP jsou vyspělé a bohaté zdroje spousty informací.

Klíčové funkce jazyka PHP 7

Dlouho očekávaná verze PHP 7 vyšla veřejně v prosinci roku 2015. Jak už víte z úvodu, tato kniha se zabývá verzemi PHP 5.6 a PHP 7, což vás pravděpodobně přivede k otázce: „Kam se poděla verze PHP 6?“ Krátká odpověď zní: „Neexistuje žádná verze PHP 6 a nikdy ani veřejně nevyšla.“ Existovala sice snaha o vývoj kódu, která byla označovaná jako „PHP 6“, ale samotný vývoj se nikdy neuskutečnil. Vývojový tým měl spoustu ambiciózních plánů, přičemž následně narazil na spoustu komplikací, které nakonec znemožnily věnovat se dále vývoji. PHP 7 není PHP 6 – neobsahuje funkce, které byly navrhované pro PHP 6. Místo toho se jedná o zcela novou verzi se svým vlastním cílem, a tím je lepší výkon.

Pod kapotou jazyka PHP 7 se skrývá kompletně přepsané jádro Zend Engine, které přináší výrazné vylepšení výkonu pro mnoho webových aplikací – někdy až o 100 %. Cílem verze PHP 7 nebylo pouze zvýšit efektivitu a snížit paměťové nároky, ale také zachovat zpětnou kompatibilitu. A skutečně bylo zavedeno jen velmi málo nekompatibilních změn. Tyto změny budou popsané v jednotlivých kapitolách této knihy tak, aby příklady fungovaly ve verzi PHP 5.6 i ve verzi PHP 7. Někteří poskytovatelé webhostingu totiž zatím nepřešli na verzi PHP 7.

Několik předností systému MySQL

Hlavními konkurenty MySQL v oblasti relačních databází jsou PostgreSQL, Microsoft SQL Server a Oracle. Na webu se však začíná objevovat rostoucí trend používání NoSQL/nerelačních databází, jejichž hlavním zástupcem je databáze MongoDB. Existuje však stále celá řada důvodů, proč pohlížet na databázový systém MySQL jako na dobrou volbu.

MySQL má mnoho předností:

- vysoký výkon,
- nízká cena,
- snadnost konfigurace a učení,
- přenositelnost,
- dostupnost zdrojového kódu,
- dostupnost podpory.

Následuje podrobnější popis jeho předností.

Výkon

Databázový systém MySQL je nepopiratelně rychlý. Stránku s banchmarky vývojářů najdete na internetové adrese *http://www.mysql.com/why-mysql/benchmarks/*.

Nízká cena

MySQL je k dispozici zdarma pod licencí otevřeného softwaru a za mírný poplatek pod komerční licencí. To znamená, že pokud distribuujete MySQL jako součást aplikace, potřebujete komerční licenci, ale nepotřebujete ji pro aplikace s otevřeným zdrojovým kódem. Pokud nehodláte distribuovat svou aplikaci (což je typické pro většinu webových aplikací), pracujete na volně dostupné aplikaci nebo aplikaci s otevřeným zdrojovým kódem, nemusíte si kupovat licenci.

Snadnost použití

Většina moderních databází používá jazyk SQL. V případě, že jste někdy používali jiný systém pro správu relačních databází, neměli byste mít žádné potíže se systémem MySQL. Ten lze totiž snadněji nastavit než převážnou část podobných produktů.

Přenositelnost

MySQL můžete používat ve spoustě unixových operačních systémů a také v systému Windows.

Zdrojový kód

Podobně jako u jazyka PHP můžete stáhnout a upravit zdrojový kód systému MySQL. To samozřejmě není pro většinu vývojářů příliš důležitý faktor, ale alespoň by vás měl přesvědčit, že vývoj tohoto systému bude pokračovat i v budoucnu a že v případě krize byste mohli tuto možnost využít.

Ve skutečnosti existuje několik forků a náhrad za systém MySQL, které byste mohli používat, a to včetně systému MariaDB, jejž vyvíjí původní autoři systému MySQL – včetně Michaela „Montyho" Wideniuse (*https://mariadb.org/*).

Dostupnost podpory

Ne všechny produkty s otevřeným zdrojovým kódem mají svou rodičovskou společnost, která by poskytovala podporu, školení, konzultace a certifikace – všechny tyto výhody můžete získat od společnosti Oracle (jež obdržela systém MySQL spolu s akvizicí společnosti Sun Microsystems, která už předtím převzala zakládající společnost MySQL AB).

Co je nového v MySQL 5.x?

V době psaní této knihy byla aktuální verzí systému MySQL verze 5.7.

Mezi nově přidané funkce v posledních několika verzích patří:

- celá řada bezpečnostních vylepšení,
- podpora indexů typu `FULLTEXT` pro InnoDB tabulky,
- NoSQL styl rozhraní API pro InnoDB,
- podpora partitioningu,
- vylepšení replikace (včetně řádkové replikace a replikace GTID),
- fond vláken,
- systém PAM (Pluggable Authentication Module),
- škálovatelnost pro více jader,
- lepší diagnostické nástroje,
- InnoDB jako výchozí formát úložiště dat,
- podpora IPv6,
- rozhraní Plugin API,
- plánování událostí,
- automatické aktualizace.

K dalším změnám se řadí kompatibilita se standardem ANSI a lepší efektivita.

Pokud stále používáte staré verze 4.x nebo 3.x, měli byste vědět, že v různých verzích od verze 4.0 přibyly tyto funkce:

- pohledy,
- uložené procedury,
- triggery a kurzory,
- podpora poddotazů,
- typy GIS pro ukládání geografických dat,
- lepší podpora internacionalizace,
- transakčně bezpečný formát úložiště dat InnoDB,
- mezipaměť pro dotazy, jež výrazně vylepšuje rychlost zpracování opakujících se dotazů, které jsou často spouštěné webovými aplikacemi.

Jak je uspořádaná tato kniha?

Tato kniha je rozdělená na pět hlavních částí:

- Část 1, „Jazyk PHP“, obsahuje přehled hlavních částí jazyka PHP s příklady. Každý příklad řeší nějaký úkol z praxe. Tuto část zahajuje kapitola 1, „Rychlokurz jazyka PHP“. Pokud jste již používali jazyk PHP, můžete ji prolítnout. Jestliže začínáte s jazykem PHP, nebo dokonce začínáte programovat, pravděpodobně zde strávíte o něco více času.
- Část 2, „Systém MySQL“, popisuje koncepce a návrh systémů pro správu relačních databází, jazyk SQL, spojení databáze MySQL s aplikací PHP a také pokročilá témata, jako jsou bezpečnost a optimalizace.
- Část 3, „Bezpečnost webových aplikací“, se zabývá běžnými problémy vývoje webových aplikací v jakémkoliv jazyce. Dozvíte se v ní, jak používat jazyk PHP a systém MySQL pro přihlašování uživatelů a zabezpečený sběr, přenos a ukládání dat.
- Část 4, „Pokročilé techniky v jazyce PHP“, detailně popisuje nejdůležitější vestavěné funkce jazyka PHP. Vybrali jsme pro vás skupinu funkcí, kterou jsou užitečné pro stavbu webové aplikace. Naučíte se o interakci se serverem a sítí, o generování obrázků, o práci s časem a o zpracování relací.
- Část 5, „Praktické projekty s jazykem PHP a systémem MySQL“, řeší problémy z praxe – například správu velkých projektů a ladění. Obsahuje ukázkové projekty, které demonstrují sílu a všestrannost jazyka PHP a systému MySQL.

Závěrem

Doufáme, že si užijete tuto knihu a učení se jazyku PHP a systému MySQL tak jako my, když jsme poprvé začali používat tyto produkty. Opravdu je radost s nimi pracovat. Brzy se připojíte k mnoha tisícům webových vývojářů, kteří používají tyto robustní nástroje pro snadné budování dynamických realtime webových aplikací.

Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu přeložilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

Computer Press
Albatros Media a.s., pobočka Brno
IBC
Příkop 4
602 00 Brno

nebo

sefredaktor.pc@albatrosmedia.cz

Computer Press neposkytuje rady ani jakýkoli servis pro aplikace třetích stran. Pokud budete mít dotaz k programu, obraťte se prosím na jeho tvůrce.

Zdrojové kódy ke knize

Ze stránky knihy na *www.albatrosmedia.cz* si po klepnutí na odkaz Soubory ke stažení můžete přímo stáhnout archiv s ukázkovými kódy.

Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nelze. Pokud v některé z našich knih nějakou najdete, ať už v textu nebo v kódu, budeme rádi, pokud nám ji oznámíte.

Veškerá existující errata najdete na stránce knihy na *www.albatrosmedia.cz* po klepnutí na odkaz Soubory ke stažení. (Nejsou-li žádná errata zatím k dispozici, není odkaz Soubory ke stažení dostupný.)

KAPITOLA

1

Rychlokurz jazyka PHP

V této kapitole najdete stručný přehled syntaxe a jazykových konstrukcí jazyka PHP. Pokud už jste programátorem PHP, možná doplní vaše znalosti. Jestliže jste pracovali s jazykem C, Perl, Python nebo jiným programovacím jazykem, určitě ji pochopíte rychle.

V této knize se dozvíte, jak pracovat s jazykem PHP, a to na praktických příkladech, které pocházejí z naší vlastní zkušenosti se stavbou reálných webových stránek. Knihy o programování někdy učí pouze základní syntaxi s velmi jednoduchými příklady. My jsme se rozhodli, že se od nich odlišíme. Pravděpodobně budete chtít něco vybudovat, spustit a pochopit, jak používat jazyk PHP, a ne se prodírat další referencí syntaxe a funkcí, která není o nic lepší než online dokumentace.

Vyzkoušejte si tyto příklady. Napište je do editoru nebo stáhněte z webových stránek, změňte je, rozbijte je a zkuste je znovu opravit.

Tato kapitola začíná objednávkovým formulářem online produktu, jenž ukazuje, jak používat proměnné, operátory a výrazy v jazyce PHP. Navíc se zabývá typy proměnných a prioritou operátorů. Dozvíte se, jak přistupovat k proměnným formuláře a měnit je – pomocí výpočtu celkové ceny a hodnoty daně objednávky zákazníka.

Potom vytvoříte ukázkový online objednávkový formulář, přičemž v PHP skriptu budete validovat vstupní data. Objevíte koncepci pravdivostních hodnot, podmíněné příkazy `if`, `else`, podmíněný výraz `?:` a příkaz `switch`. Nakonec prozkoumáte cykly, s nimiž vygenerujete opakující se tabulky HTML.

Ke klíčovým tématům z této kapitoly patří:

- **vkládání kódu jazyka PHP do dokumentu HTML,**
- **přidávání dynamického obsahu,**
- **přistupování k formulářovým proměnným,**
- **pochopení identifikátorů,**
- **vytváření uživatelských proměnných,**
- **prozkoumání typů proměnných,**
- **přiřazování hodnot proměnným,**
- **definování a používání konstant,**
- **pochopení oblastí platnosti proměnných,**
- **pochopení operátorů a priorit,**
- **vyhodnocování výrazů,**
- **používání proměnných funkcí,**
- **dělat rozhodnutí s pomocí příkazů `if`, `else` a `switch`,**
- **využívat výhod iterace prostřednictvím cyklů `while`, `do` a `for`.**

Než začnete – nainstalovaný jazyk PHP

Abyste mohli pracovat s příklady v této kapitole (a ve zbytku knihy), musíte mít k dispozici webový server s nainstalovaným PHP. Pokud chcete vytěžit z příkladů maximum, měli byste si je spustit a poté se je pokusit změnit. K tomu potřebujete vývojový počítač, na kterém můžete experimentovat.

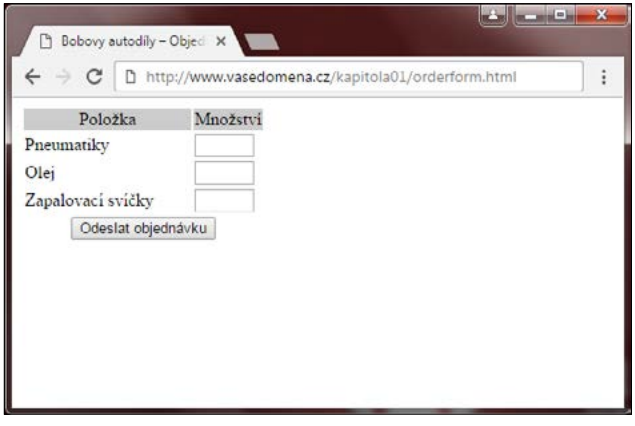
Pokud nemáte nainstalovaný jazyk PHP ve svém počítači, nainstalujte si ho nebo požádejte svého systémového administrátora, ať vám ho nainstaluje. Instalační návod najdete v příloze A.

Tvorba ukázkové aplikace – Bobovy autodíly

Aplikace na straně serveru nejčastěji zpracovávají formuláře jazyka HTML. Jazyk PHP se začnete učit tak, že vytvoříte objednávkový formulář pro Bobovy autodíly, fiktivní společnost s náhradními díly. Veškerý kód příkladů z této knihy najdete v archivu se zdrojovými kódy.

Tvorba objednávkového formuláře

Bobův HTML programátor připravil objednávkový formulář pro díly, které Bob prodává. Tento relativně jednoduchý objednávkový formulář, zobrazený na obrázku 1.1, se podobá spoustě jiných formulářů, které jste již pravděpodobně na webu viděli. Bob by rád zjistil, co si jeho zákazníci objednali, spočítal celkové ceny jejich objednávek a určil hodnotu daně pro tyto objednávky.



Obrázek 1.1. Bobův počáteční objednávkový formulář zaznamenává pouze množství jednotlivých produktů

Část kódu jazyka HTML pro tento formulář je k dispozici ve výpise 1.1.

Výpis 1.1. orderform.html – Kód HTML pro Bobův základní objednávkový formulář

```
<form action="processorder.php" method="post">
  <table style="border: 0px;">
    <tr style="background: #cccccc;">
      <td style="width: 150px; text-align: center;">Položka</td>
      <td style="width: 15px; text-align: center;">Množství</td>
    </tr>
    <tr>
      <td>Pneumatiky</td>
      <td><input type="text" name="tireqty" size="3" maxlength="3" /></td>
    </tr>
    <tr>
      <td>Olej</td>
      <td><input type="text" name="oilqty" size="3" maxlength="3" /></td>
    </tr>
    <tr>
      <td>Zapalovací svíčky</td>
      <td><input type="text" name="sparkqty" size="3" maxlength="3" /></td>
    </tr>
    <tr>
      <td colspan="2" style="text-align: center;">
        <input type="submit" value="Odeslat objednávku" />
      </td>
    </tr>
  </table>
</form>
```

Povšimněte si, že jako akci formuláře nastavujete název PHP skriptu, jenž bude zpracovávat zákazníkovu objednávku (ten napíšete za chvíli). Obecně řečeno – hodnotou atributu `action` je URL adresa, jež se načte, když uživatel klepne na odesílací tlačítko. Data, která uživatel zadal do formuláře, prohlížeč odešle na tuto adresu URL pomocí metody protokolu HTTP definované atributem `method` – buď `get` (data odesílaná na konci adresy URL), nebo `post` (data odesílaná jako samostatná zpráva).

Také si zapamatujte názvy formulářových polí – `tireqty`, `oilqty` a `sparkqty`. Ty budete znovu používat v PHP skriptu. Proto je důležité pojmenovávat pole smysluplně, abyste si je lehce zapamatovali a mohli ji použít při psaní PHP skriptu. Některé editory zdrojového kódu jazyka HTML generují automaticky názvy polí jako `field23`. Ty si lze jen obtížně zapamatovat. Váš život PHP programátora bude mnohem jednodušší, když budou názvy polí odpovídat datům, která do nich vkládáte.

Zkuste si zvolit nějakou konvenci pojmenování tak, aby všechny názvy polí na celých vašich webových stránkách měly stejný formát. Díky tomu se budete moct rychleji rozhodovat, zda kupříkladu zkrátíte slovo v poli nebo použijete podtržítka místo mezer.

Zpracování formuláře

Abyste mohli zpracovat formulářová data, musíte vytvořit skript, na nějž se odkazujete v atributu `action` elementu `form` – v tomto případě *processorder.php*. Otevřete si tedy textový editor a vytvořte tento soubor. Potom do něho vložte níže uvedený zdrojový kód:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Bobovy autodíly – Výsledek objednávky</title>
  </head>
  <body>
    <h1>Bobovy autodíly</h1>
    <h2>Výsledek objednávky</h2>
  </body>
</html>
```

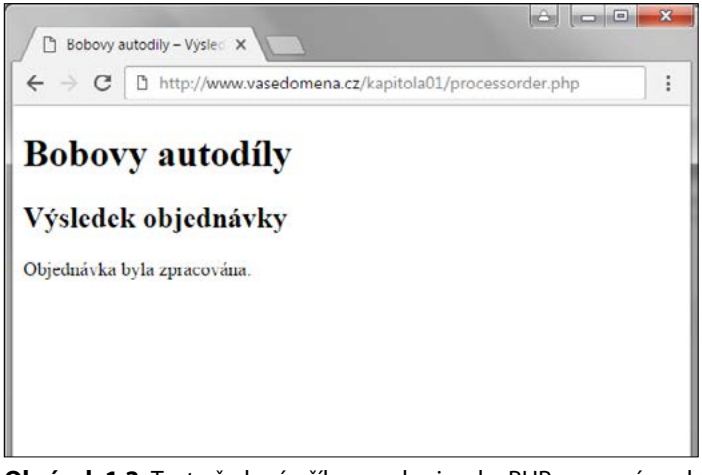
Všimněte si, že jste dosud napsali jen kód v jazyce HTML. Nyní do něj vložíte jednoduchý kód v jazyce PHP.

Vkládání kódu PHP do dokumentu HTML

Pod nadpis `h2` přidejte následující řádky:

```
<?php
  echo '<p>Objednávka byla zpracována.</p>';
?>
```

Uložte soubor a načtete ho ve svém webovém prohlížeči tak, že vyplníte Bobův formulář a klepnete na tlačítko **Odeslat objednávku**. Měli byste vidět podobný výstup jako na obrázku 1.2.



Obrázek 1.2. Text předaný příkazu echo jazyka PHP vypsaný prohlížečem

Kód jazyka PHP jste vložili do obyčejného dokumentu HTML. Zkuste si prohlédnout zdrojový kód v prohlížeči. Měl by vypadat takto:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Bobovy autodíly – Výsledek objednávky</title>
  </head>
  <body>
    <h1>Bobovy autodíly</h1>
    <h2>Výsledek objednávky</h2>
    <p>Objednávka byla zpracována.</p>
  </body>
</html>
```

Žádný kód jazyka PHP zde není. Je tomu tak z toho důvodu, že interpret jazyka PHP prošel celý skript a nahradil kód jazyka PHP výstupem daného skriptu. To znamená, že pomocí jazyka PHP můžete generovat kód HTML pro prohlížeč – tj. prohlížeč nemusí rozumět jazyku PHP.

Tento příklad demonstruje koncepci skriptování na straně serveru. Webový server interpretoval a spustil kód jazyka PHP. To se liší od jazyka JavaScript a jiných klientských technologií, které interpretuje a spouští webový prohlížeč na počítači uživatele.

Váš soubor nyní obsahuje čtyři typy textů:

- kód HTML,
- značky jazyka PHP,
- příkazy jazyka PHP,
- bílé znaky.

Můžete také přidávat komentáře.

Většinu řádků výše uvedeného příkladu tvoří však čistý kód jazyka HTML.

Značky jazyka PHP

Předchozí příklad začínal výrazem `<?php` a končil výrazem `?>`. To se podobá značkám jazyka HTML, které rovněž začínají symbolem menší než (<) a končí symbolem větší než (>). Tyto výrazy (`<?php` a `?>`) nazýváme **značky jazyka PHP**. Těmi sdělujeme webovému serveru, kde začíná a končí kód jazyka PHP. Text mezi nimi tudíž webový server interpretuje jako jazyk PHP. S textem mimo ně pak zachází jako s běžným jazykem HTML. Značky jazyka PHP umožňují „uniknout“ z kódu jazyka HTML.

Existují však dva styly značek jazyka PHP:

- **Styl jazyka XML**

  ```
  <?php echo '<p>Objednávka byla zpracována.</p>'; ?>
  ```

  S tímto stylem se budete setkávat v této knize, jelikož je obvyklejší. Administrátoři serverů ho nemůžou zakázat, takže vám bude fungovat na všech serverech. To je důležité, pokud se chystáte psát aplikace, které budou instalovány na různých místech. Tento styl můžete používat v dokumentech XML (Extensible Markup Language).

- **Stručný styl**

  ```
  <? echo '<p>Objednávka byla zpracována.</p>'; ?>
  ```

  Tento styl značek je nejjednodušší, přičemž se řídí standardem SGML (Standardized Generalized Markup Language). Abyste ho mohli používat, musíte povolit nastavení `short_open_tag` v konfiguračním souboru nebo zkompilovat jazyk PHP tak, aby podporoval krátké značky. Více informací o tom, jak použít tento styl, najdete v příloze A. Tento styl není vhodný pro kód, který se chystáte distribuovat. Nebude fungovat v mnoha prostředích, protože už není standardně povolený.

Příkazy jazyka PHP

Interpretu jazyka PHP sdělíte, co chcete udělat, pomocí příkazů, které vkládáte mezi otevírací a uzavírací značky. Ve výše uvedeném příkladu jste se setkali s jedním typem příkazu:

```
echo '<p>Objednávka byla zpracována.</p>';
```

Pravděpodobně jste správně uhádli, že příkaz `echo` má velmi jasný výsledek – vypisuje jemu předaný textový řetězec do webového prohlížeče. Na obrázku 1.2 jste viděli, že v okně webového prohlížeče se objevil text „Objednávka byla zpracována“.

Všimněte si středníku na konci příkazu `echo`. Středníky oddělují příkazy v jazyce PHP, podobně jako tečky oddělují věty v češtině. Pokud jste někdy programovali v jazyce C nebo Java, určitě váš použití středníku tímto způsobem nepřekvapí.

Opomenutí středníku je častou syntaktickou chybou. Lze se jí snadno dopustit, ale stejně tak ji je možné jednoduše najít a opravit.

Bílé znaky

Znak konce řádku, mezeru a tabulátor označujeme termínem **bílé znaky**. Nejspíše víte, že prohlížeče ignorují bílé znaky v kódu jazyka HTML. Stejně se chová interpret jazyka PHP. Prohlédněte si tento fragment kódu:

```
<h1>Vítejte na Bobových autodílech!</h1><p>Co byste si chtěli objednat
dnes?</p>
```

a tento kód:

```
<h1>Vítejte         na Bobových
autodílech!<h1>
<p>Co byste si chtěli
objednat dnes?</p>
```

Oba produkují stejný výstup, jelikož prohlížeči se jeví totožné. Bílé znaky byste měli používat s rozvahou tak, abyste pomáhali lidem – vylepšovali čitelnost zdrojového kódu. To samé platí pro jazyk PHP. Mezi příkazy jazyka PHP nemusíte vkládat žádné bílé znaky, ale když umístíte jednotlivé příkazy na samostatné řádky, výrazně tím usnadníte čtení kódu. Například:

```
echo 'ahoj ';
echo 'světe';
```

a

```
echo 'ahoj ';echo 'světe';
```

se shodují, ale první verze je snadněji čitelná.

Komentáře

Komentáře fungují jako poznámky pro lidi, kteří budou číst váš zdrojový kód. Pomocí komentářů můžete ozřejmit účel skriptu, sdělit, kdo ho napsal, proč byl napsán daným způsobem, kdy byl naposledy upraven atd. S komentáři se většinou setkáte v téměř všech skriptech jazyka PHP (s výjimkou těch nejjednodušších).

Interpret jazyka PHP ignoruje všechny komentáře. Zachází s nimi tedy podobně jako s bílými znaky.

Jazyk PHP podporuje komentáře ve stylu jazyka C, C++ a shellového skriptu.

Následuje víceřádkový komentář ve stylu jazyka C, jenž se může objevit na začátku PHP skriptu:

```
/* Autor: Bob Smith
   Poslední úprava: 10. dubna 2016
   Tento skript zpracovává objednávky zákazníků.
*/
```

Víceřádkové komentáře by měly začínat posloupností `/*` a končit posloupností `*/`. Stejně jako v jazyce C není možné vnořovat víceřádkové komentáře do sebe.

Můžete také používat jednořádkové komentáře. Buď ve stylu jazyka C++:

```
echo '<p>Objednávka byla zpracována.</p>'; // Vypisuje stav objednávky
```

nebo ve stylu shellového skriptu:

```
echo '<p>Objednávka byla zpracována.</p>'; # Vypisuje stav objednávky
```

Všechno, co následuje za symbolem komentáře (`//` nebo `#`), interpret považuje za komentář, dokud nedosáhne konce řádku nebo koncové značky PHP (podle toho, co se nachází dříve).

Na následujícím řádku kódu je text před uzavírací značkou (`toto je komentář`) součástí komentáře. Text za uzavírací značkou (`toto není`) se považuje za kód jazyka HTML:

```
// toto je komentář ?> toto není
```

Přidání dynamického obsahu

Dosud jste nepoužili jazyk PHP k ničemu jinému, než čeho byste dosáhli pomocí jazyka HTML.

Hlavním důvodem, proč vývojáři používají skriptovací jazyk na straně serveru, je jeho schopnost poskytovat dynamický obsah uživatelům. To je velmi podstatný faktor, protože obsah, který se mění, je důvodem, proč se návštěvníci na webové stránky vracejí. S pomocí jazyka PHP toho dosáhnete jednoduše.

Pro začátek následuje jednoduchý příklad. Nahraďte kód PHP v souboru *processorder.php* následujícím kódem:

```
<?php
  echo "<p>Objednávka byla zpracována ";
  echo date('j. n. Y H:i');
  echo "</p>";
?>
```

Tento kód byste mohli přepsat na jeden řádek prostřednictvím operátoru řetězení (`.`):

```
<?php
  echo "<p>Objednávka byla zpracována ".date('j. n. Y H:i')."</p>";
?>
```

Obrázek 1.3. Funkce date() vrací naformátovaný časový údaj

Vestavěná funkce `date()` ve výše uvedeném kódu sděluje zákazníkovi, kterého dne a v kolik hodin byla jeho objednávka zpracována. Tato informace se mění s každým spuštěním tohoto skriptu. Výstup lze vidět na obrázku 1.3.

Volání funkce

Prohlédněte si volání funkce `date()`. Jedná se o obecnou formu volání funkcí. Jazyk PHP nabízí rozsáhlou knihovnu funkcí, které můžete používat při vývoji webových aplikací. Většině těchto funkcí předáváte nějaká data, přičemž ony vám vrací také nějaké data.

Nyní se znovu podívejte na volání této funkce:

```
date('j. n. Y H:i');
```

Všimněte si, že jí předáváte textový řetězec (textová data) mezi dvojicí závorek. Prvek uvnitř závorek se označuje jako **argument** nebo **parametr** funkce. Tyto argumenty slouží jako vstup, jejž daná funkce zpracovává, aby na výstup poslala určité výsledky.

Funkce date()

Funkce `date()` očekává, že jí předáte jako argument textový řetězec, jenž reprezentuje formát výstupního času. Každé písmeno v tomto textovém řetězci představuje komponentu data nebo času:

- `H` označuje hodinu ve 24hodinovém formátu s vedoucí nulou tam, kde je nutná,
- `i` představuje minutu s případnou vedoucí nulou,
- `j` je den v měsíci bez vedoucích nul,
- `n` reprezentuje měsíc bez vedoucích nul,
- `Y` označuje rok.

POZNÁMKA

Pokud vás funkce **`date()`** varuje, že jste nenastavili časovou zónu, měli byste doplnit nastavení **`date.timezone`** do souboru *php.ini*. Více informací najdete v ukázkovém souboru *php.ini* v příloze A.

Úplný přehled formátů podporovaných funkcí `date()` se nachází v kapitole 19.

Přístup k proměnným formuláře

Účelem objednávkového formuláře je sbírat objednávky zákazníků. Zjistit, co zákazník napsal, je s jazykem PHP snadné. Přesná metoda závisí na verzi jazyka PHP a na tom, jakou konfiguraci máte v souboru *php.ini*.

Proměnné formuláře

Ve svém PHP skriptu můžete přistupovat k jednotlivým formulářovým polím jako k proměnným, které se jmenují stejně jako příslušná formulářová pole. V jazyce PHP poznáte proměnné tak, že začínají symbolem dolaru ($). Obvyklou syntaktickou chybou, které se programátoři PHP dopouštějí, je, že zapomínají na symbol dolaru.

V závislosti na verzi PHP a konfiguraci můžete přistupovat k proměnným různými způsoby. Nedávno byly všechny tyto způsoby s výjimkou jednoho označeny za zastaralé. Pokud jste tedy používali jazyk PHP v minulosti, dávejte si na tuto změnu pozor.

K obsahu pole `tireqty` můžete přistupovat následujícím způsobem:

```
$_POST['tireqty']
```

`$_POST` je pole obsahující data odeslaná požadavkem HTTP POST – tj. metoda formuláře měla hodnotu `POST`. Existují tři pole, která mohou obsahovat formulářová data: `$_POST`, `$_GET` a `$_REQUEST`. Některé z polí `$_GET` nebo `$_POST` obsahuje všechny proměnné formuláře. Které z nich to bude, závisí na tom, jestli jste zvolili metodu formuláře `GET` nebo `POST`. Kromě toho je k dispozici kombinace všech dat odeslaných metodami `GET` a `POST`, a to v poli `$_REQUEST`.

Jestliže jste odeslali formulářová data metodou `POST`, hodnota pole `tireqty` je uložená v prvku `$_POST['tireqty']`. Kdybyste odeslali data metodou GET, hodnotu byste našli skrze `$_GET['tireqty']`. V obou případech bude tato hodnota dostupná přes `$_REQUEST['tireqty']`.

Jedná se o tzv. **superglobální** pole. Na superglobální proměnné ještě narazíte později, až bude řeč o oblasti platnosti proměnných.

Ukažme si, jak vytvořit snadněji použitelné kopie těchto proměnných.

Ke zkopírování hodnoty jedné proměnné do jiné použijte operátor přiřazení, což je v jazyce PHP rovnítko (=). Níže uvedeným příkazem vytvoříte novou proměnnou `$tireqty` a zkopírujete do ní hodnotu proměnné `$_POST['tireqty']`:

```
$tireqty = $_POST['tireqty'];
```

Vložte níže uvedený blok kódu na začátek skriptu. Ostatní skripty z této knihy, které budou zpracovávat data, budou mít podobný blok kódu na začátku. Protože neprodukuje žádný výstup, mohli byste ho vložit i nad nebo pod značku `<html>` a jiné značky jazyka HTML. My vkládáme tento blok na začátek skriptu, kde ho lze jednoduše najít:

```
<?php
  // vytváříme zkrácené názvy proměnných
  $tireqty = $_POST['tireqty'];
  $oilqty = $_POST['oilqty'];
  $sparkqty = $_POST['sparkqty'];
?>
```

Právě jste vytvořili tři nové proměnné – `$tireqty`, `$oilqty` a `$sparkqty` – a přidělili jste jim data odeslaná metodou POST z formuláře.

Mohli byste vypsat jejich hodnoty do prohlížeče kupříkladu takto:

```
echo $tireqty.' pneumatik<br />';
```

Tento přístup však nelze doporučit.

V této fázi jste si nezkontrolovali, zda uživatel zadal do všech formulářových polí smysluplná data. Zkuste vkládat záměrně špatná data a sledujte, co se stane. Po přečtení zbytku kapitoly možná budete chtít doplnit validaci dat do tohoto skriptu.

Načíst uživatelská data a vzápětí je vypsat do prohlížeče je extrémně riskantní z hlediska bezpečnosti. Rozhodně tento přístup nedoporučujeme. Vstupní data byste měli filtrovat. O filtrování dat bude pojednávat kapitola 4 a o bezpečnosti se dozvíte více v kapitole 14.

Prozatím bude stačit, když budete vypisovat uživatelská data do webového prohlížeče až poté, co je předáte funkci `htmlspecialchars()`. Například v tomto případě napište tento řádek kódu:

```
echo 'pneumatik: '.htmlspecialchars($tireqty).'<br />';
```

Aby váš skript začal dělat něco viditelného, přidejte následující řádky na jeho konec:

```
echo '<p>Vaše objednávka:</p>';
echo 'pneumatik: '.htmlspecialchars($tireqty).'<br />';
echo 'lahví oleje: '.htmlspecialchars($oilqty).'<br />';
echo 'zapalovacích svíček: '.htmlspecialchars($sparkqty).'<br />';
```

Když načtete tento skript ve webovém prohlížeči, jeho výstup by měl vypadat podobně jako na obrázku 1.4. Samotné hodnoty samozřejmě závisejí na tom, co napíšete do formuláře.

Obrázek 1.4. Proměnné, které uživatel zadal do formuláře, jsou dostupné ve skriptu processorder.php

V následující části si popíšeme několik zajímavých prvků tohoto příkladu.

Řetězení textových řetězců

V ukázkovém skriptu vypisujete pomocí příkazů `echo` hodnoty, které uživatel zadal do jednotlivých formulářových polí, a k tomu doplňujete vysvětlující text. Pokud si pozorněji prohlédnete tyto příkazy, zjistíte, že název proměnné je od textu oddělený tečkou (`.`):

```
echo 'pneumatik: '.htmlspecialchars($tireqty).'<br />';
```

Tečka reprezentuje operátor řetězení textových řetězců, jenž skládá textové řetězce (kusy textu) dohromady. Budete ho často používat při odesílání výstupu webovému prohlížeči příkazem `echo`. Díky tomu nemusíte psát více příkazů `echo`.

Jestliže chcete vypsat obsah proměnné, můžete ji vložit do uvozovek (s poli je situace složitější, a proto o kombinaci pole a textových řetězců pojednává kapitola 4). Uvažte tento příklad:

```
$tireqty = htmlspecialchars($tireqty);
echo "pneumatik: $tireqty<br />";
```

Tento blok kódu se shoduje s prvním příkazem z této části. Oba formáty jsou platné – můžete si vybrat, jaký chcete. Tomuto procesu nahrazení proměnné jejím obsahem v textovém řetězci se říká **interpolace**.

Zapamatujte si, že interpolace probíhá pouze uvnitř textových řetězců uzavřených do uvozovek. Nemůžete vkládat názvy proměnných mezi apostrofy. Například tento řádek kódu:

```
echo 'pneumatik: $tireqty<br />';
```

by odeslal do prohlížeče text `pneumatik: $tireqty<br />`. S uvozovkami interpret nahrazuje názvy proměnných jejich hodnotami. S apostrofy interpret ponechá názvy proměnných jako běžný text.

Proměnné a literály

Proměnné a textové řetězce, které řetězíme dohromady ve výše uvedených příkazech `echo`, jsou dvě naprosto odlišné věci. Proměnné jsou značky pro data. Textové řetězce jsou samotná data. Když použijeme kus surových dat v kódu, označujeme ho jako **literál**, abychom ho odlišili od **proměnné**.

`$tireqty` je proměnná – značka, jež reprezentuje data, která zákazník zadal. Na druhou stranu – `'pneumatik: '` je literál. Můžete ho považovat za konečnou hodnotu – tedy skoro. Vzpomínáte si na druhý příklad z předcházející části? Interpret PHP nahradil název proměnné `$tireqty` v textovém řetězci hodnotou uloženou v této proměnné.

Jistě si vybavujete, že existují dva druhy textových řetězců – jedny s uvozovkami a druhé s apostrofy. PHP vyhodnocuje textové řetězce uzavřené mezi uvozovky, a proto v nich

nahrazuje názvy proměnných hodnotami. S textovými řetězci v apostrofech zachází jako se skutečnými literály.

K dispozici je rovněž třetí způsob definování textových řetězců, a to pomocí syntaxe heredoc, kterou budou jistě znát všichni uživatelé jazyka Perl. Ta umožňuje specifikovat dlouhé textové řetězce přehledně pomocí koncové značky pro uzavření textového řetězce. V následujícím příkladu vypisujeme třířádkový textový řetězec:

```
echo <<<toJeKonec
  řádek 1
  řádek 2
  řádek 3
toJeKonec
```

Token `toJeKonec` si můžete zvolit jakýkoliv. Musíte si být však jistí, že se neobjevuje v samotném textu. Textový řetězec heredoc uzavřete tak, že umístíte uzavírací token na začátek řádku.

Textové řetězce heredoc jsou interpolovány – stejně jako textové řetězce uzavřené do uvozovek.

Identifikátory

Identifikátory jsou názvy proměnných. (Názvy funkcí a tříd jsou také identifikátory; na funkce a třídy se zaměřují kapitoly 5 a 6.) Musíte znát jednoduchá pravidla pro definici platných identifikátorů:

- Identifikátory můžou mít libovolnou délku a můžou se skládat z písmen, číslic a podtržítek.
- Identifikátory nesmí začínat číslicí.
- Interpret jazyka PHP rozlišuje velikost písmen v identifikátorech. To znamená, že proměnná `$tireqty` není stejná jako proměnná `$TireQty`. Jejich záměna je běžnou programátorskou chybou. Názvy funkcí mají výjimku na toto pravidlo – jejich názvy lze používat s jakoukoliv velikostí písmen.
- Proměnná může mít stejný název jako funkce. To je ale velmi matoucí, a proto byste se takové situaci měli vyhýbat. Také nemůžete vytvořit funkci, která nese stejný název jako jiná funkce.

Kromě proměnných předaných z formuláře HTML můžete definovat své vlastní proměnné.

Jazyk PHP má tu vlastnost, že v něm nemusíte deklarovat proměnné předtím, než je použijete. Proměnná vznikne automaticky tak, že jí poprvé přiřadíte hodnotu. Podrobnější informace najdete v následující části.

Hodnoty proměnným přiřazujete pomocí operátoru přiřazení (=) stejně, jako když jste kopírovali hodnotu z jedné proměnné do druhé. Na Bobových stránkách budete chtít

zjistit celkový počet položek objednávky a její celkovou cenu. Proto si můžete vytvořit dvě proměnné pro uložení těchto čísel. Na začátku jim přiřaďte nulové hodnoty:

```
$totalqty = 0;
$totalamount = 0.00;
```

Na obou řádcích vytváříme proměnnou a přiřazujeme jí hodnotu (zapsanou jako literál). Je také možné přiřazovat hodnoty proměnných jiným proměnným, jak lze vidět na tomto příkladu:

```
$totalqty = 0;
$totalamount = $totalqty
```

Typy proměnných

Typ proměnné označuje typ dat, která lze do ní ukládat. Jazyk PHP nabízí skupinu datových typů. Různá data je možné ukládat v různých datových typech.

Datové typy jazyka PHP

PHP podporuje tyto základní datové typy:

- **Integer** – pro celá čísla
- **Float** (nebo také **double**) – pro desetinná čísla
- **String** – pro textové řetězce a znaky
- **Boolean** – pro pravdivostní hodnoty `true` a `false`
- **Array** – pro ukládání více datových položek (více o polích najdete v kapitole 3)
- **Object** – pro ukládání instancí tříd (viz kapitola 6)

Třemi speciálními typy jsou **NULL**, **resource** a **callable**.

Proměnné, kterým nikdo nepřiřadil žádnou hodnotu, nenastavil je nebo jim explicitně přidělil hodnotu `NULL`, jsou typu NULL.

Některé vestavěné funkce (například databázové funkce) vracejí proměnné datového typu resource. Jedná se o externí prostředky (například o databázová spojení). S datovým typem resource nebudete muset téměř nikdy pracovat přímo, ale data tohoto typu vracejí funkce a vy je potom předáváte jako argumenty jiným funkcím.

Datový typ callable mají funkce, které předáváte jiným funkcím.

Síla typování

PHP bývá označovaný jako slabě typovaný (nebo také dynamicky typovaný) jazyk. Ve většině programovacích jazyků lze do proměnných ukládat jen data jednoho konkrétního typu a přitom tento typ musíte deklarovat předem (například v jazyce C). V jazyce PHP získává proměnná datový typ na základě hodnoty, kterou jí přidělíte.

Když jsme vytvořili proměnné `$totalyqty` a `$totalamount`:

```
$totalqty = 0;
$totalamount = 0;
```

rozhodli jsme o jejich datových typech. Jelikož jsme proměnné `$totalqty` přidělili `0`, bude datového typu `integer`. Podobně proměnná `$totalamount` je typu `float`.

Přestože se to může zdát divné, mohli byste nyní do svého skriptu doplnit následující řádek:

```
$totalamount = 'Ahoj';
```

Proměnná `$totalamount` by nově získala datový typ `string`. Interpret jazyka PHP mění typ proměnné podle toho, jakou hodnotu jste do ní uložili naposled.

Schopnost měnit typy transparentně za běhu může být neobyčejně užitečná. PHP ví, jaký datový typ jste umístili do své proměnné. Až načtete hodnotu této proměnné, vrátí data se stejným datovým typem.

Přetypování

Můžete předstírat, že proměnná nebo hodnota je jiného datového typu, prostřednictvím přetypování. To funguje stejným způsobem jako v jazyce C. Dočasný typ můžete uvést do závorek před název proměnné, kterou chcete přetypovat.

Jak vidíte, jazyk PHP poskytuje spoustu svobody v této oblasti. Všechny jazyky vám umožňují měnit hodnotu proměnné, ale jen málo z nich vám dovolí měnit její datový typ, a dokonce mnohem méně vám umožní měnit název proměnné.

Proměnná proměnných používá jako hodnotu název jiné proměnné. Například když přidělíte proměnné hodnotu:

```
$nazevpromenne = 'tireqty';
```

můžete používat výraz `$$nazevpromenne` místo `$tireqty`. Hodnotu proměnné `$tireqty` můžete přidělit kupříkladu takto:

```
$$nazevpromenne = 5;
```

To je srovnatelné se zápisem:

```
$tireqty = 5;
```

Tento přístup se může zdát poněkud divný, ale později se k němu ještě vrátíme. Nebudeme muset vypisovat všechny formulářové proměnné zvlášť, ale místo toho použijeme cyklus a proměnnou proměnných, abychom je zpracovali všechny automaticky. Příklad si uvedeme v části s cyklem `for` později v této kapitole.

Definujeme a používáme konstanty

Už jste viděli, že můžete měnit hodnotu uloženou v proměnné. Navíc můžete také definovat konstanty. Konstanta ukládá hodnotu podobně jako proměnná, ale hodnotu jí předělíte pouze jednou a potom ji nesmíte nikde měnit.

V naší ukázkové aplikaci jsme si mohli uložit ceny jednotlivých položek jako konstanty. Ty definujeme pomocí funkce `define`:

```
define('TIREPRICE', 100);
define('OILPRICE', 10);
define('SPARKPRICE', 4);
```

Pokud přidáte tyto řádky do svého skriptu, budete mít tři konstanty, které můžete použít pro výpočet celkové ceny objednávky.

Povšimněte si, že názvy konstant mají velká písmena. Tato konvence pochází z jazyka C a jejím účelem je usnadnit rozlišení konstant a proměnných na první pohled. Nemusíte se jí řídit, ale díky ní bude váš kód čitelnější a lépe udržovatelný.

Dalším rozdílem mezi konstantami a proměnnými je, že když se odkazujete na konstantu, nemusíte před ní psát symbol dolaru. Jestliže chcete získat hodnotu konstanty, použijte pouze její název. Kdybyste chtěli použít některou z právě vytvořených konstant, mohli byste napsat například:

```
echo TIREPRICE;
```

PHP nabízí také spoustu vestavěných konstant. Jejich přehled obdržíte po zavolání funkce `phpinfo()`:

```
phpinfo();
```

Tato funkce vypisuje (kromě mnoha dalších užitečných informací) seznam předdefinovaných proměnných a konstant. Některé z nich si popíšeme v této knize.

Další rozdíl mezi proměnnou a konstantou je, že do konstanty lze ukládat pouze celá čísla, desetinná čísla a textové řetězce. Tyto datové typy bývají souhrnně označovány jako skalární hodnoty.

Oblast platnosti proměnných

Oblastí platnosti rozumíme místo ve skriptu, kde je určitá proměnná viditelná. V jazyce PHP máme šest základní pravidel pro oblast platnosti:

- Vestavěné superglobální proměnné jsou viditelné kdekoliv ve skriptu.
- Definované konstanty jsou viditelné globálně – je možné je používat uvnitř i vně funkcí.

- Globální proměnné deklarované ve skriptu jsou viditelné v tomto skriptu, ale ne uvnitř funkcí.
- Proměnné uvnitř funkce, které deklarujeme jako globální, odpovídají stejně pojmenovaným globálním proměnným.
- Proměnné vytvořené uvnitř funkce a deklarované jako statické nejsou mimo tuto funkci viditelné, ale zachovávají si hodnotu mezi jednotlivými voláními této funkce. (Tuto myšlenku rozvineme podrobněji v kapitole 5.)
- Proměnné vytvořené uvnitř funkce jsou lokální v této funkci a přestávají existovat, když tato funkce skončí.

Pole `$_GET` a `$_POST` a několik dalších speciálních proměnných mají svá vlastní pravidla pro oblast platnosti. Nazýváme je **superglobální proměnné** a jsou viditelné všude – vně i uvnitř funkcí.

Kompletní seznam superglobálních proměnných si uvedeme zde:

- `$GLOBALS` – pole všech globálních proměnných (podobně jako s klíčovým slovem `global` získáme přístup ke globálním proměnným uvnitř funkce – například přes `$GLOBALS['mapromenna']`)
- `$_SERVER` – pole serverových proměnných prostředí
- `$_GET` – pole proměnných předaných skriptu metodou GET
- `$_POST` – pole proměnných předaných skriptu metodou POST
- `$_COOKIE` – pole proměnných získaných z cookies
- `$_FILES` – pole proměnných vztahujících se k uploadovaným souborům
- `$_ENV` – pole proměnných prostředí
- $_REQUEST – pole všech proměnných souvisejících s uživatelským vstupem, které obsahuje dohromady vše, co obsahují pole `$_GET`, `$_POST` a `$_COOKIE` (mimo pole `$_FILES`)
- `$_SESSION` – pole proměnných relace

Ke všem těmto superglobálním proměnným se budeme postupně vracet v průběhu této knihy.

Oblasti platnosti se budeme věnovat podrobněji, až se budeme bavit o funkcích a třídách. Prozatím pracujeme jen s globálními proměnnými.

Operátory

Operátory jsou symboly, s nimiž můžeme upravovat hodnoty a proměnné tak, že na nich provádíme různé operace. Bez některých z těchto operátorů se neobejdeme při výpočtu celkové ceny a hodnoty daně objednávky.

Již jste se setkali se dvěma operátory: operátorem přiřazení (=) a operátorem řetězení (.). Za chvíli se seznámíte s ostatními.

Obecně řečeno – operátory přijímají jednu, dvě nebo tři hodnoty, přičemž převažují operátory se dvěma hodnotami. Například přiřazovací operátor přijímá dvě hodnoty : úložiště na levé straně symbolu = a výraz na pravé straně. Tyto argumenty se nazývají **operandy** – tj. prvky, s nimiž provádíme operace.

Aritmetické operátory

Aritmetické operátory jsou jednoduché – jedná se o běžné matematické operátory. Přehled aritmetických operátorů jazyka PHP najdete v tabulce 1.1.

Tabulka 1.1. Aritmetické operátory jazyka PHP

Operátor	Název	Příklad
+	Sčítání	`$a + $b`
-	Odečítání	`$a - $b`
*	Násobení	`$a * $b`
/	Dělení	`$a / $b`
%	Modulo	`$a % $b`

U všech těchto operátorů platí, že výsledek operace si můžeme ukládat. Například takto:

```
$vysledek = $a + $b;
```

Sčítání a odečítání funguje tak, jak si myslíte. Cílem těchto operátorů je sčítat nebo odečítat hodnoty uložené v proměnných `$a` a `$b`.

Kromě toho lze symbol minus (-) používat jako unární operátor – tj. operátor, který má jediný argument (operand) – a to pro záporná čísla:

```
$a = -1;
```

Násobení a dělení funguje rovněž tak, jak čekáte. Všimněte si, že operátorem násobení je hvězdička (*) místo běžného symbolu násobení a operátorem dělení je lomítko (/) místo běžného symbolu dělení.

Operátor modulo vrací zbytek po celočíselném dělení proměnné `$a` proměnou `$b`. Zde je příklad:

```
$a = 27;
$b = 10;
$vysledek = $a % $b;
```

Hodnotou proměnné `$vysledek` je zbytek po dělení čísla 27 číslem 10 – tj. číslo `7`.

Měli byste si zapamatovat, že aritmetické operátory běžně aplikujete na celá a desetinná čísla. Pokud je aplikujete na textové řetězce, interpret jazyka PHP se pokusí převést tex-

tový řetězec na číslo. Jestliže obsahuje písmeno `e` nebo `E`, bude ho považovat za vědecký zápis a převede na desetinné číslo, v opačném případě ho převede na celé číslo. Standardně hledá číslice na začátku textového řetězce a ty pak použije jako součást hodnoty; pokud je nenajde, převede textový řetězec na nulu.

Operátory řetězení

Už jste měli příležitost použít jediný operátor řetězení. Pomocí něj můžete spojovat dva textové řetězce a výsledek si uložit podobně, jako kdybyste sčítali dvě čísla:

```
$a = "Bobovy ";
$b = "autodíly";
$vysledek = $a.$b;
```

Proměnná `$vysledek` nyní obsahuje textový řetězec `"Bobovy autodíly"`.

Operátor přiřazení

Již jste viděli základní operátor přiřazení (`=`). Vždy o něm můžete mluvit jako o operátoru přiřazení a číst ho jako „má hodnotu". Například:

```
$totalqty = 0;
```

Tento řádek kódu můžete přečíst jako „`$totalqty` má hodnotu nula". Důvod bude jasnější, až narazíte na porovnávací operátory později v této kapitole, protože kdybyste tento operátor nazývali „rovná se", snadno byste se mohli splést.

Hodnoty vrácené přiřazením

Operátor přiřazení vrací výslednou hodnotu podobně jako jiné operátory. Když napíšete:

```
$a + $b
```

hodnotou tohoto výrazu bude výsledek sečtení proměnných `$a` a `$b`. Podobně můžete napsat:

```
$a = 0;
```

Hodnotou tohoto celého výrazu je nula.

Tato technika nám umožňuje zapisovat výrazy typu:

```
$b = 6 + ($a = 5);
```

Tímto řádkem nastavujeme proměnné `$b` hodnotu 11. Toto chování platí pro všechna přiřazení – hodnotou celého přiřazení je hodnota, které je přiřazena levému operandu.

Když vyhodnocujete hodnotu výrazu, můžete používat závorky, abyste zvýšili prioritu dílčího výrazu. Tato technika funguje stejně jako v matematice.

Kombinované operátory přiřazení

Kromě základního přiřazení existuje navíc skupina kombinovaných operátorů přiřazení. Každý z nich umožňuje zkráceným zápisem provést další operaci s proměnnou a výsledek přiřadit zpět do této proměnné. Například:

```
$a += 5;
```

je ekvivalentní s:

```
$a = $a + 5;
```

Kombinované operátory jsou k dispozici pro všechny aritmetické operátory a pro operátor řetězení. Přehled všech kombinovaných operátorů přiřazení je k dispozici v tabulce 1.2.

Tabulka 1.2. Kombinované operátory přiřazení

Operátor	Použití	Ekvivalentní s
+=	$a += $b;	$a = $a + $b
-=	$a -= $b;	$a = $a - $b
*=	$a *= $b;	$a = $a * $b
/=	$a /= $b;	$a = $a / $b
%=	$a %= $b;	$a = $a % $b
.=	$a .= $b;	$a = $a . $b

Operátory inkrementace a dekrementace

Operátory inkrementace (++) a dekrementace (--) se podobají kombinovaným operátorům přiřazení += a -=, až na několik rozdílů.

Všechny operátory inkrementace mají dva účely – navyšují a přiřazují hodnotu. Uvažte tento kód:

```
$a = 4;
echo ++$a;
```

Na druhém řádku se nachází operátor pre-inkrementace. Nazývá se tak proto, že je umístěný před názvem proměnné `$a`. Tento operátor nejprve navyšuje hodnotu proměnné `$a` o `1` a potom teprve vrací navýšenou hodnotu. V tomto případě zvýší hodnotu na 5 a následně ji vrátí, aby ji mohl příkaz `echo` vypsat. Celý výraz má hodnotu `5`. Nezapomínejte však, že tento operátor pouze nevrací hodnotu `$a + 1`, ale také ji ukládá do proměnné `$a`.

V případě, že by se operátor ++ nacházel za proměnnou `$a`, jednalo by se o operátor postinkrementace. Ten se chová trochu jinak. Podívejte se na následující blok kódu:

```
$a = 4;
echo $a++;
```

V tomto případě se pořadí operací obrátí. Operátor nejprve vrátí hodnotu proměnné `$a` a příkaz `echo` ji vypíše a potom se její hodnota zvýší. Celý výraz má hodnotu `4`. Tuto hodnotu také vypíše příkaz `echo`. Následně dojde ke zvýšení hodnoty proměnné `$a` na `5`.

Jak jste pravděpodobně uhádli, operátor dekrementace (`--`) se chová podobně, ale hodnotu proměnné `$a` bude snižovat, a ne zvyšovat.

Operátor reference

Operátor reference (`&`, ampersand) lze používat ve spojení s přiřazením. Když běžně přiřadíte jednu proměnnou do druhé, interpret PHP zkopíruje její hodnotu někam jinam do paměti. Například:

```
$a = 5;
$b = $a;
```

Těmito řádky jsme vytvořili kopii hodnoty proměnné `$a` a uložili jsme ji do proměnné `$b`. Když nyní změníme hodnotu proměnné `$a`, hodnota proměnné `$b` se nezmění:

```
$a = 7; // $b má stále hodnotu 5
```

Tvorbě kopie se můžete vyhnout operátorem reference. Například:

```
$a = 5;
$b = &$a;
$a = 7; // $a a $b mají nyní hodnotu 7
```

Reference můžou být ošidné. Zapamatujte si, že reference se chová spíše jako alias než jako ukazatel. Proměnné `$a` a `$b` ukazují na stejné místo paměti. Můžete to změnit tak, že jednu z nich zrušíte:

```
unset($a);
```

Zrušením nezměníte hodnotu proměnné `$b` (`7`), ale narušíte vazbu mezi proměnnou `$a` a touto hodnotu `7` uloženou v paměti.

Porovnávací operátory

Porovnávací operátory porovnávají dvě hodnoty. Tyto operátory vracejí některou z pravdivostních hodnot – buď `true`, nebo `false`, a to podle výsledku srovnání.

Operátor rovnosti

Porovnávací operátor rovnosti (`==`) umožňuje ověřit, jestli jsou dvě hodnoty shodné. Například bychom mohli použít tento výraz:

```
$a == $b
```

Tak otestujeme, zda se hodnoty uložené v proměnných `$a` a `$b` shodují. Výsledek tohoto porovnání bude `true`, pokud se shodují, jinak bude `false`.

Operátor == si můžete snadno splést s operátorem =, operátorem přiřazení. Použití špatného operátoru nezobrazí žádnou chybovou zprávu, ale určitě vygeneruje špatné výsledky. Nenulové hodnoty budou vyhodnocené jako `true`, zatímco nulové hodnoty jako `false`. Kupříkladu kdybyste inicializovali dvě proměnné následovně:

```
$a = 5;
$b = 7;
```

a potom byste použili `$a = $b`, výsledek by byl `true`. Proč? Hodnotou výrazu `$a = $b` je hodnota přiřazená operandu na levé straně, což je v tomto případě číslo `7`. Protože `7` je nenulová hodnota, výraz bude vyhodnocený jako `true`. Pokud jste chtěli testovat pomocí výrazu `$a == $b`, který by byl vyhodnocený jako `false`, právě jste do svého kódu zavedli logickou chybu, kterou bude nesmírně těžké odhalit. Proto vždy pečlivě kontrolujete, který z těchto dvou operátorů používáte, abyste měli jistotu, že jste zvolili ten správný.

Použití operátoru přiřazení namísto porovnávacího operátoru rovnosti je častou chybou, kterou pravděpodobně uděláte za svou programátorskou kariéru mnohokrát.

Další porovnávací operátory

Jazyk PHP nabízí několik dalších porovnávacích operátorů. Všechny porovnávací operátory jsou uvedené v tabulce 1.3. Zdůrazníme především operátor identity (===), jenž vrací `true` jen tehdy, když se oba operandy shodují a jsou stejného datového typu. Například výraz `0=='0'` bude vyhodnocený jako `true`, ale výraz `0==='0'` nikoliv, jelikož prvním operandem je celé číslo, kdežto druhým je textový řetězec obsahující znak `0`.

Tabulka 1.3. Porovnávací operátory jazyka PHP

Operátor	Název	Použití
==	Rovná se	`$a == $b`
===	Identický	`$a === $b`
!=	Nerovná se	`$a != $b`
!==	Neidentický	`$a !== $b`
<>	Nerovná se	`$a <> $b`
<	Menší než	`$a < $b`
>	Větší než	`$a > $b`
<=	Menší nebo rovno	`$a <= $b`
>=	Větší nebo rovno	`$a >= $b`

Logické operátory

Logické operátory kombinují výsledky logických podmínek. Například kdybyste chtěli zjistit, zda se hodnota proměnné `$a` nachází v rozmezí `0` až `100`, použili byste podmínky `$a >= 0` a `$a <= 100` a logický operátor AND:

```
$a >= 0 && $a <= 100
```

Jazyk PHP podporuje logické operátory AND, OR, XOR (exclusive or) a NOT.

Přehled logických operátorů a jejich způsobů použití najdete v tabulce 1.4.

Tabulka 1.4. Logické operátory jazyka PHP

Operátor	Název	Použití	Výsledek
!	NOT	!$b	Vrací true, pokud $b je false a opačně.
&&	AND	$a && $b	Vrací true, jestliže $a i $b jsou true; jinak false.
||	OR	$a || $b	Vrací true, pokud $a, $b nebo obě podmínky jsou true; jinak false.
and	AND	$a and $b	Stejný jako operátor &&, ale má nižší prioritu.
or	OR	$a or $b	Stejný jako operátor ||, ale má nižší prioritu.
xor	XOR	$a xor $b	Vrací true, pokud je $a nebo $b true, a false v případě, že jsou obě podmínky true nebo obě false.

Operátory and a or mají nižší prioritu než operátory && a ||. O prioritách se dozvíte více později v této kapitole.

Bitové operátory

Bitové operátory umožňují zacházet s celým číslem jako se sérií bitů, které ho tvoří. Pravděpodobně je nebudete potřebovat příliš často, ale pro kompletnost je najdete v tabulce 1.5.

Tabulka 1.5. Bitové operátory jazyka PHP

Operátor	Název	Použití	Výsledek
&	Bitový AND	$a & $b	Bity nastavené (tj. bity s hodnotou 1) v $a a $b budou nastavené ve výsledku.
|	Bitový OR	$a | $b	Bity nastavené v $a nebo $b budou nastavené ve výsledku.
~	Bitový NOT	~$a	Bity nastavené v $a nebudou nastavené ve výsledku a naopak.
^	Bitový XOR	$a ^ $b	Bity nastavené v $a nebo $b, ale ne v obou budou nastavené ve výsledku.
<<	Posun doleva	$a << $b	Posouvá $a o $b bitů doleva.
>>	Posun doprava	$a >> $b	Posouvá $a o $b bitů doprava.

Další operátory

Kromě dosud popsaných operátorů máte k dispozici také několik dalších.

Operátor čárka (`,`) odděluje parametry funkce a jiné seznamy položek. Běžně ho vývojáři používají zcela automaticky, bez přemýšlení.

Pomocí speciálních operátorů `new` a `->` lze vytvářet instance tříd a přistupovat k metodám a vlastnostem třídy. Podrobněji budou vysvětlené v kapitole 6.

Existuje rovněž několik dalších operátorů, které si zde stručně popíšeme.

Ternární operátor

Ternární operátor (`?:`) zapisujeme takto:

```
podmínka ? hodnota pro true : hodnota pro false
```

Tento operátor zkracuje příkaz `if-else`, o němž si povíme více později v této kapitole.

Následuje jednoduchý příklad:

```
($znamka >= 50 ? 'Prospěl' : 'Nedostačující')
```

Tímto výrazem ověřujeme, jestli student prospěl, nebo byly jeho znalosti nedostačující.

Operátor potlačení chyby

Operátor potlačení chyby (`@`) je možné použít před libovolným výrazem – před čímkoliv, co generuje nebo má nějakou hodnotu. Například:

```
$a = @(57/0);
```

Bez operátoru `@` by tento řádek způsobil varování o dělení nulou. S operátorem `@` je však tato chyba potlačená.

Pokud potlačujete varování tímto způsobem, musíte napsat kód, ve kterém budete tyto chyby detekovat a ošetřovat. Jestliže jste v konfiguračním souboru *php.ini* povolili nastavení `track_errors`, chybovou zprávu najdete v globální proměnné `$php_errormsg`.

Operátor spuštění

Operátor spuštění je ve skutečnosti dvojicí operátorů – jedná se o dvojici zpětných apostrofů (`''`). Zpětná uvozovka se liší od běžného apostrofu – obvykle se nachází na stejné klávese jako tilda (~).

Interpret PHP zkouší spustit vše, co se nachází mezi zpětnými apostrofy, jako příkaz v příkazovém řádku na serveru. Hodnotou tohoto výrazu je výstup z tohoto příkazu.

Kupříkladu v unixových operačních systémech můžete použít:

```
$vystup = 'ls -la';
echo '<pre>'.$vystup.'</pre>';
```

Nebo na serveru s operačním systémem Windows můžete použít:

```
$vystup = 'dir c:';
echo '<pre>'.$vystup.'</pre>';
```

V obou blocích kódu načítáme obsah adresáře a ukládáme ho do proměnné `$vystup`. Potom ho vypisujeme do webového prohlížeče.

Existují také jiné způsoby spouštění příkazů na serveru. Popíšeme si je v kapitole 17.

Operátory pro práci s poli

K dispozici je celá řada operátorů pro práci s poli. Operátor prvku pole (`[]`) umožňuje přistupovat k prvkům pole. V některých případech lze používat rovněž operátor `=>`. Na tyto operátory se zaměříme v kapitole 3.

Navíc existuje spousta dalších operátorů, které si podrobněji popíšeme v kapitole 3, ale zde si je ve stručnosti uvedeme v tabulce 1.6.

Tabulka 1.6. Operátory pole jazyka PHP

Operátor	Název	Použití	Výsledek
+	Sjednocení	`$a + $b`	Vrací pole obsahující vše z polí $a a $b.
==	Rovnost	`$a == $b`	Vrací `true`, jestliže $a a $b obsahují stejné dvojice klíčů a hodnot.
===	Identita	`$a === $b`	Vrací `true`, jestliže $a a $b mají stejné dvojice klíčů a hodnot ve stejném pořadí a stejného typu.
!=	Nerovnost	`$a != $b`	Vrací `true`, když se $a a $b nerovnají.
<>	Nerovnost	`$a <> $b`	Vrací `true`, když se $a a $b nerovnají.
!==	Neidentita	`$a !== $b`	Vrací `true`, jestliže $a a $b nejsou identické.

Pravděpodobně jste si všimli, že operátory pole z tabulky 16.1 mají své ekvivalentní operátory pro práci se skalárními hodnotami. Například pomocí operátoru + můžete sečíst skalární hodnoty nebo sjednotit pole, což je smysluplné chování. Nemůžete však mezi sebou porovnávat pole a skalární typy.

Typový operátor

Existuje jeden typový operátor – `instanceof`. Tento operátor je určený pro objektově orientované programování, ale zde ho uvádíme pro kompletnost. (O objektově orientovaném programování se dozvíte více v kapitole 6.)

Prostřednictvím operátoru `instanceof` můžete ověřit, jestli je objekt instancí konkrétní třídy, jako je tomu v tomto příkladu:

```
class UkazkovaTrida{};
$mujObjekt = new UkazkovaTrida();
if ($mujObjekt instanceof UkazkovaTrida)
  echo "mujObjekt je instancí UkazkovaTrida";
```

Výpočet součtů formuláře

Když už víte, jak používat operátory jazyka PHP, můžete spočítat celkovou cenu a hodnotu daně Bobova objednávkového formuláře. Za tímto účelem vložte níže uvedený zdrojový kód na konec svého skriptu PHP:

```
$totalqty = 0;
$totalqty = $tireqty + $oilqty + $sparkqty;
echo "<p>Objednáno položek: ".$totalqty."<br />";
$totalamount = 0.00;

define('TIREPRICE', 2500);
define('OILPRICE', 250);
define('SPARKPRICE', 100);

$totalamount = $tireqty * TIREPRICE +
               $oilqty * OILPRICE +
               $sparkqty * SPARKPRICE;

echo "Cena: ".number_format($totalamount, 2)." Kč<br />";

$taxrate = 0.20;  // výše DPH je 20%
$totalamount = $totalamount * (1 + $taxrate);
echo "Celková cena s DPH: ".number_format($totalamount,2)." Kč</p>";
```

Když obnovíte vaši stránku ve webovém prohlížeči, měli byste vidět něco podobného jako na obrázku 1.5.

Jak je patrné, v tomto kódu používáte několik operátorů. Cenu počítáte pomocí operátoru sčítání (+) a násobení (*) a k tomu zapojujete také operátor řetězení textových řetězců (.), abyste vypsali výsledky do webového prohlížeče.

Obrázek 1.5. Celkové součty objednávky jsme spočítali, naformátovali a zobrazili

Funkcí `number_format()` formátujeme součty do textových řetězců se dvěma desetinnými místy. Ta pochází z matematické knihovny jazyka PHP.

Jestliže si prohlédnete výpočty, nejspíš vás napadne, proč jsou výpočty uspořádané tak, jak jsou. Například uvažte tento příkaz:

```
$totalamount = $tireqty * TIREPRICE +
               $oilqty * OILPRICE +
               $sparkqty * SPARKPRICE;
```

Celková cena se zdá být správná, ale proč interpret jazyka PHP násobí dříve, než provádí sčítání? Odpověď spočívá v prioritě operátorů – tj. v pořadí jejich vyhodnocování.

Pochopení priorit a asociací

Operátory mají své priority – tj. pořadí, v němž jsou zpracovány. Navíc mají také asociace – pořadí, ve kterém jsou zpracovány operátory se stejnou prioritou. Uspořádání může být zleva doprava (zkráceně označované **zleva**), zprava doleva (označované jako **zprava**) nebo **irelevantní**.

V tabulce 1.7 najdete priority a asociace operátorů jazyka PHP. V této tabulce se operátory s nejnižší prioritou nacházejí nahoře a priorita se zvyšuje, jak postupujete tabulkou dolů.

Tabulka 1.7. Priorita operátorů jazyka PHP

Asociace	Operátory
zleva	,
zleva	or
zleva	xor
zleva	and
zprava	print
zleva	= += -= *= /= .= %= &= |= ^= ~= <<= >>=
zleva	? :
zleva	||
zleva	&&
zleva	|
zleva	^
zleva	&
irelevantní	== != === !=
irelevantní	< <= > >=
zleva	<< >>

Asociace	Operátory
zleva	`+ - .`
zleva	`* / %`
zprava	`!`
irelevantní	`instanceof`
zprava	`~ (int) (float) (string) (array) (object) (bool) @`
irelevantní	`++ --`
zprava	`[]`
irelevantní	`clone new`
irelevantní	`()`

Prozatím jsme se nezabývali operátorem s nejvyšší prioritou – starými dobrými závorkami. Cílem závorek je zvýšit prioritu čehokoliv, co se nachází uvnitř nich. Tímto způsobem je možné měnit pravidla priorit.

Připomeňte si tuto část výše uvedeného příkladu:

```
$totalamount = $totalamount * (1 + $taxrate);
```

Kdybyste napsali:

```
$totalamount = $totalamount * 1 + $taxrate;
```

násobení by mělo přednost před sčítáním, takže byste dostali špatný výsledek. Pomocí závorek můžete vynutit, aby interpret jazyka PHP nejprve sečetl výraz `1 + $taxrate`.

Závorek můžete použít, kolik chcete. Nejvnitřnější pár závorek bude vyhodnocený jako první.

Povšimněte si rovněž v tabulce 1.7 operátoru, který ještě neznáte – konstrukce `print` se shoduje s konstrukcí `echo`. Obě generují výstup.

V této knize se setkáte spíše s příkazem `echo`, ale pokud vám přijde čitelnější příkaz `print`, můžete ho používat. Přestože `echo` a `print` nejsou funkce, lze je volat jako funkce s argumenty mezi závorkami. S oběma je možné zacházet jako s operátory – jednoduše zapíšete textový řetězec za klíčové slovo `echo` nebo `print`.

Pokud zavoláte `print` jako funkci, vrátí hodnotu `(1)`. Toto chování může být užitečné, jestliže chcete generovat výstup uvnitř složitějších výrazů, ale také to znamená, že příkaz `print` je pomalejší než příkaz `echo`.

Funkce pro zpracování proměnných

Než opustíme svět proměnných a operátorů, popíšeme si funkce pro zpracování proměnných. Jazyk PHP nabízí knihovnu, která umožňuje upravovat a testovat proměnné různými způsoby.

Testování a nastavení typů proměnných

Většina funkcí pro práci s proměnnými slouží pro testování typu. Nejběžnější jsou funkce `gettype()` a `settype()`. Mají následující funkční prototypy – určují, co funkce očekávají jako argumenty a co vracejí:

```
string gettype(mixed promenna);
bool settype(mixed promenna, string typ);
```

Funkci `gettype()` předáváte proměnnou. Tato funkce určí její typ a vrátí textový řetězec s názvem typu – `bool`, `int`, `double` (pro desetinná čísla, z historických důvodů), `string`, `array`, `object`, `resource` nebo `NULL`. Kromě toho vrací `unknown type`, pokud daná proměnná nemá žádný standardní typ.

Funkci `settype()` předáváte proměnnou, jejíž typ chcete změnit, a textový řetězec obsahující nový typ proměnné.

POZNÁMKA

V této knize a dokumentaci php.net se setkáte s datovým typem **`mixed`**. Tento typ neexistuje, ale jelikož jazyk PHP zpracovává typy velmi flexibilně, spousta funkcí přijímá argument, který může mít více datových typů. Argumenty, u nichž lze použít více datových typů, se obvykle zobrazují s pseudotypem **`mixed`**.

Tyto funkce je možné volat následovně:

```
$a = 56;
echo gettype($a).'<br />';
settype($a, 'float');
echo gettype($a).'<br />';
```

Při prvním volání funkce `gettype()` je proměnná `$a` typu `integer`. Zavolání funkce `settype()` změní typ na `float`, který se však hlásí jako `double` (nezapomeňte na tento rozdíl).

Jazyk PHP poskytuje také několik specifických funkcí pro testování typu. Všechny přijímají proměnnou jako argument a vracejí `true` nebo `false`. Patří k nim:

- `is_array()` – ověřuje, jestli je proměnná polem
- `is_double`, `is_float()` a `is_real()` (ekvivalentní funkce) – ověřují, zda je proměnná desetinným číslem
- `is_long()`, `is_int()` a `is_integer()` (ekvivalentní funkce) – ověřují, jestli je proměnná celým číslem
- `is_string()` – ověřuje, zda je proměnná textovým řetězcem
- `is_bool()` – ověřuje, zda je proměnná pravdivostní hodnotou
- `is_object()` – ověřuje, jestli je proměnná objektem

- `is_resource()` – ověřuje, zda je proměnná prostředkem
- `is_null()` – ověřuje, jestli je proměnná `null`
- `is_scalar()` – ověřuje, zda je proměnná skalární hodnotou – celé číslo, pravdivostní hodnota, textový řetězec nebo desetinné číslo
- `is_numeric()` – ověřuje, jestli je proměnná číslem libovolného typu nebo číselným textovým řetězcem
- `is_callable()` – ověřuje, zda je proměnná názvem platné funkce

Testování stavu proměnné

PHP obsahuje několik funkcí pro testování stavu proměnné. První z nich je funkce `isset()`, jež má následující prototyp:

```
bool isset(mixed promenna[, mixed promenna [,...]])
```

Tato funkce přijímá název proměnné jako argument, přičemž vrací `true` v případě, že existuje; v opačném případě vrací `false`. Můžete jí rovněž předat seznam proměnných oddělených čárkami – vrátí `true`, jestliže jsou všechny tyto proměnné nastavené.

Proměnnou můžete zcela zrušit pomocí doprovodné funkce `unset()`, která má následující prototyp:

```
void unset(mixed promenna[, mixed promenna[,...]])
```

Tato funkce odstraní proměnnou, kterou jí předáte.

Funkce `empty()` ověřuje, zda proměnná neexistuje nebo má prázdnou, případně nulovou hodnotu. Vrací `true` nebo `false` a má tento prototyp:

```
bool empty(mixed promenna)
```

Ukažme si příklad s těmito třemi funkcemi.

Zkuste dočasně vložit níže uvedený kód do svého skriptu:

```
echo 'isset($tireqty): '.isset($tireqty).'<br />';
echo 'isset($neexistujici): '.isset($neexistujici).'<br />';
echo 'empty($tireqty): '.empty($tireqty).'<br />';
echo 'empty($neexistujici): '.empty($neexistujici).'<br />';
```

Obnovte stránku, abyste viděli výsledek.

Funkce `isset()` by měla vrátit hodnotu `1` (`true`) pro proměnnou `$tireqty`, a to bez ohledu na to, jakou hodnotu zadáte do formuláře (nebo jestli ji vůbec zadáte). Návratová hodnota `empty()` už ale závisí na tom, co jste zadali do formuláře.

Proměnná `$neexistujici` neexistuje, a proto generuje prázdný výsledek (`false`) ve funkci `isset()` a hodnotu `1` (`true`) ve funkci `empty()`.

Tyto funkce se hodí zejména tehdy, když potřebujete ověřit, jestli uživatel vyplnil povinná pole ve formuláři.

Reinterpretace proměnných

Přetypovat proměnnou lze i zavoláním funkce. Následující tři funkce můžou být užitečné pro tento úkol:

```
int intval(mixed promenna[, int zaklad=10])
float floatval(mixed promenna)
string strval(mixed promenna)
```

Všechny mají proměnnou jako parametr a vracejí její hodnotu převedenou do příslušného datového typu. Funkce `intval()` umožňuje také definovat základ číselné soustavy pro převod v případě, že bychom převáděli textový řetězec na číslo (tak je možné převádět například textové řetězce obsahující čísla v šestnáctkové soustavě na celá čísla).

Rozhodování podle podmínek

Řídicí struktury jsou struktury jazyka, které umožňují řídit tok programu. Můžeme je rozdělit na podmíněné příkazy a cykly.

Pokud máme smysluplně reagovat na vstupní data, musíme se v našem kódu rozhodovat. Konstrukce, pomocí nichž činí náš program rozhodnutí, se nazývají **podmíněné příkazy**.

Příkaz if

Můžete se rozhodovat pomocí příkazu `if`. Příkazu `if` byste měli dát podmínku. Pokud bude podmínka pravdivá, interpret jazyka PHP provede za ní následující blok. Podmínky v příkazech `if` musíte uzavřít do závorek.

V případě, že si návštěvník neobjedná žádné pneumatiky, žádné lahve oleje ani žádné zapalovací svíčky, pravděpodobně klepnul na tlačítko **Odeslat objednávku** omylem dříve, než stihl vyplnit formulář. V takovém případě by se na stránce se zprávou „Objednávka byla zpracována“ měla objevit smysluplnější zpráva.

Pokud si zákazník nic neobjednal, měli bychom mu zobrazit kupříkladu zprávu: „Nic jste si neobjednal(a) na předchozí stránce.“ Toho dosáhneme jednoduše prostřednictvím následujícího příkazu `if`:

```
if ($totalqty == 0)
  echo 'Nic jste si neobjednal(a) na předchozí stránce.<br />';
```

Zde se nachází podmínka `$totalqty == 0`. Vzpomeňte si, že operátor rovnosti (`==`) se chová jinak než operátor přiřazení (`=`).

Podmínka `$totalqty == 0` bude pravdivá, když se proměnná `$totalqty` bude rovnat nule. Pokud se proměnná `$totalqty` nebude rovnat nule, podmínka bude vyhodnocena jako `false`. Jestliže bude podmínka pravdivá, provede se příkaz `echo`.

Bloky kódu

Často budete chtít spustit více než jeden příkaz po vyhodnocení podmínky podmíněného příkazu, jakým je příkaz `if`. Více příkazů můžete seskupovat do **bloku**. Blok definujete tak, že uzavřete příkazy mezi složené závorky:

```
if ($totalqty == 0) {
  echo '<p style="color:red">';
  echo 'Nic jste si neobjednal(a) na předchozí stránce.<br />';
  echo '</p>';
}
```

Tři řádky uzavřené mezi složené závorky tvoří nyní blok kódu. V případě, že interpret PHP vyhodnotí podmínku jako pravdivou, provede všechny tři řádky. Jestliže ji vyhodnotí jako nepravdivou, všechny tři řádky přeskočí.

POZNÁMKA

Jak už víte, jazyk PHP se nestará o to, jak rozvrhnete zdrojový kód. Měli byste ho však správně odsazovat, abyste vylepšili jeho čitelnost. Odsazení vám umožní, abyste zahlédli na první pohled řádky, které se provedou, pokud bude podmínka splněna; také odlišuje příkazy seskupené do bloků a příkazy, které jsou součástí cyklů a funkcí. V předchozích příkladech můžete vidět, že příkaz a příkazy závisející na příkazu `if` jsou odsazené.

Příkaz else

Často se budete chtít rozhodnout nejen o tom, jaké chcete provést akce po splnění podmínky, ale také o tom které záložní akce provést při jejím nesplnění.

Příkaz `else` umožnuje definovat alternativní akce, jestliže interpret PHP vyhodnotí podmínku příkazu `if` jako `false`. Například si představte, že byste chtěli Bobovy zákazníky varovat, když si nic neobjednají, ale na druhou stranu – pokud si něco objednali byste jim vypsali, co si objednali.

Když přeuspořádáte kód a přidáte do něj příkaz `else`, můžete zobrazit buď varování, nebo shrnutí:

```
if ($totalqty == 0) {
  echo 'Nic jste si neobjednal(a) na předchozí stránce.<br />';
} else {
  echo 'pneumatik: '.htmlspecialchars($tireqty).'<br />';
  echo 'lahví oleje: '.htmlspecialchars($oilqty).'<br />';
  echo 'zapalovacích svíček: '.htmlspecialchars($sparkqty).'<br />';
}
```

Složitější logické procesy můžete podchytit vnořováním příkazů `if` do sebe. V níže uvedeném kódu zobrazíte varování jen tehdy, když podmínka `$totalqty == 0` bude `true`. Jednotlivé řádky shrnutí zobrazíte jen za předpokladu, že budou splněny jejich vlastní podmínky:

```
if ($totalqty == 0) {
  echo 'Nic jste si neobjednal(a) na předchozí stránce.<br />';
} else {
  if ($tireqty > 0)
    echo 'pneumatik: '.htmlspecialchars($tireqty).'<br />';
  if ($oilqty > 0)
    echo 'lahví oleje: '.htmlspecialchars($oilqty).'<br />';
  if ($sparkqty > 0)
    echo 'zapalovacích svíček: '.htmlspecialchars($sparkqty).'<br />';
}
```

Příkaz elseif

U spousty rozhodovacích procesů budete potřebovat více než dvě možnosti. Sekvenci více podmínek můžete vytvořit příkazem `elseif`, jenž je kombinací příkazů `else` a `if`. Interpret PHP prochází sekvenci podmínek, dokud nenarazí na první pravdivou.

Bob poskytuje slevu pro velké objednávky pneumatik. Schéma slev vypadá takto:

- Méně než 10 pneumatik – bez slevy
- 10–49 pneumatik – 5% sleva
- 50–99 pneumatik – 10% sleva
- 100 a více pneumatik – 15% sleva

Kód pro výpočet slevy můžete vytvořit s použitím podmínek a příkazů `if` a `elseif`. V tomto případě budete muset rovněž spojit dvě podmínky do jedné pomocí operátoru AND (`&&`):

```
if ($tireqty < 10) {
  $discount = 0;
} elseif (($tireqty >= 10) && ($tireqty <= 49)) {
  $discount = 5;
} elseif (($tireqty >= 50) && ($tireqty <= 99)) {
  $discount = 10;
} elseif ($tireqty >= 100) {
  $discount = 15;
}
```

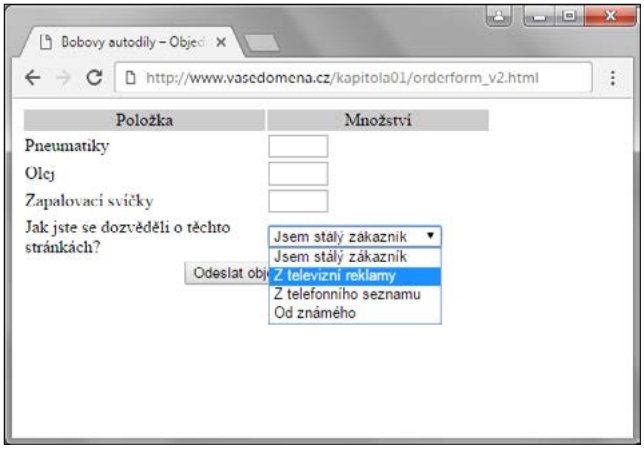
Příkaz `elseif` můžete psát i s mezerou jako `else if`. Když napíšete sérii příkazů `elseif`, nesmíte zapomínat, že se provede pouze jediný blok příkazů. V tomto případě to nevadilo, protože jednotlivé podmínky se vzájemně vylučovaly – mohla být pravdivá jen jediná z nich. Jestliže napíšete podmínky tak, že bude moct být pravdivých více podmínek současně, interpret PHP provede pouze blok příkazů následující za první pravdivou podmínkou.

Příkaz switch

Příkaz `switch` se podobá příkazu `if`, ale umožňuje, aby podmínka nabyla více než dvou hodnot. U příkazu `if` může být podmínka buď `true`, nebo `false`. V příkazu `switch` může podmínka nabývat různých hodnot za předpokladu, že budou jednoduchého datového typu (celé číslo, textový řetězec nebo desetinné číslo). Musíte použít příkazy `case` pro zpracování jednotlivých hodnot a případně výchozí větev `default` pro zpracování čehokoliv, pro co neuvedete příkaz `case`.

Bob by se chtěl dozvědět, jaký typ reklamy pro něj funguje. Za tímto účelem můžete přidat otázku do objednávkového formuláře. Když vložíte tento kód HTML do objednávkového formuláře, bude vypadat podobně jako na obrázku 1.6:

```
<tr>
  <td>Jak jste se dozvěděli o těchto stránkách?</td>
  <td><select name="find">
    <option value = "a">Jsem stálý zákazník</option>
    <option value = "b">Z televizní reklamy</option>
    <option value = "c">Z telefonního seznamu</option>
    <option value = "d">Od známého</option>
  </select></td>
</tr>
```



Obrázek 1.6. Objednávkový formulář se nyní dotazuje zákazníků, jak našli stránky Bobovy autodíly

Tento kód HTML přidává novou formulářovou proměnnou (`find`), jež může nabývat hodnoty `'a'`, `'b'`, `'c'` nebo `'d'`. Tu byste mohli zpracovat pomocí skupiny příkazů `if` a `elseif` následovně:

```
if ($find == "a") {
  echo "<p>Stálý zákazník.</p>";
} elseif ($find == "b") {
  echo "<p>Zákazník oslovený televizní reklamou.</p>";
```

```
} elseif ($find == "c") {
  echo "<p>Zákazník, který našel odkaz v telefonním seznamu.</p>";
} elseif ($find == "d") {
  echo "<p>Zákazník, kterého odkázal známý.</p>";
} else {
  echo "<p>Není jasné, jak nás tento zákazník našel.</p>";
}
```

Alternativně byste mohli napsat příkaz `switch`:

```
switch ($find) {
  case "a":
    echo "<p>Stálý zákazník.</p>";
    break;
  case "b":
    echo "<p>Zákazník oslovený televizní reklamou.</p>";
    break;
  case "c":
    echo "<p>Zákazník, který našel odkaz v telefonním seznamu.</p>";
    break;
  case "d":
    echo "<p>Zákazník, kterého odkázal známý.</p>";
    break;
  default:
    echo "<p>Není jasné, jak nás tento zákazník našel.</p>";
    break;
}
```

Povšimněte si, že oba příklady počítají s tím, že jste extrahovali hodnotu proměnné `$find` z pole `$_POST`.

Příkaz `switch` se chová poněkud jinak než příkazy `if` a `elseif`. Příkaz `if` má vliv na jediný příkaz, dokud nevytvoříte blok příkazů pomocí složených závorek. Příkaz `switch` se chová opačně. Když interpret PHP aktivuje příkaz `case` uvnitř příkazu `switch`, provádí příkazy, dokud nenarazí na příkaz `break`. Bez příkazů `break` by příkaz `switch` spustil veškerý kód, který následuje za prvním příkazem `case` s pravdivou podmínkou. Jakmile narazí na příkaz `break`, přeskočí na první řádek kódu za příkazem `switch`.

Srovnání různých podmíněných příkazů

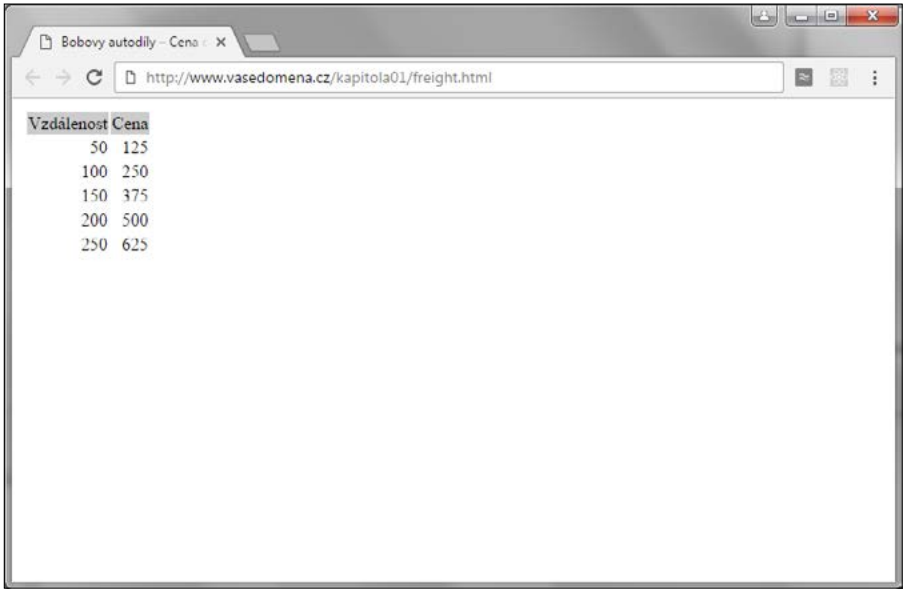
Pokud jste dosud neznali příkazy popsané v předchozích částech, možná si říkáte: „Který z nich je nejlepší?“

Na tuto otázku však nelze odpovědět. Neexistuje nic, co můžete dělat s příkazy `else`, `elseif` nebo `switch`, ale čeho byste nedosáhli se skupinou příkazů `if`. Měli byste si vybrat příkaz, který bude pro danou situaci nejlépe čitelný. Cit pro volbu vhodného příkazu pro různé situace získáte časem, jak budete získávat více zkušeností.

Opakujeme akce v iteracích

Počítače vždy vynikaly v automatizaci opakujících se úkolů. Jestliže chcete něco udělat stejným způsobem vícekrát, můžete použít cyklus, v němž zopakujete některé části svého programu.

Bob chce zobrazit tabulku s cenami dopravy u objednávky. S dopravcem, kterého si vybral, závisí cena dopravy na vzdálenosti. Cenu lze spočítat jednoduchým výpočtem. Tabulka s dopravou by se měla podobat tabulce na obrázku 1.7.



Obrázek 1.7. Tato tabulka ukazuje cenu dopravy se vzrůstající vzdáleností

Výpis 1.2 obsahuje kód HTML pro tuto tabulku. Jak můžete vidět, kód je dlouhý a často se opakuje.

Výpis 1.2. freight.html – Kód HTML pro Bobovu tabulku dopravy

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Bobovy autodíly – Cena dopravy</title>
  </head>
  <body>
    <table style="border: 0px; padding: 3px">
    <tr>
      <td style="background: #cccccc; text-align: center;">Vzdálenost</td>
      <td style="background: #cccccc; text-align: center;">Cena</td>
    </tr>
```

```
      <tr>
        <td style="text-align: right;">50</td>
        <td style="text-align: right;">125</td>
      </tr>
      <tr>
        <td style="text-align: right;">100</td>
        <td style="text-align: right;">250</td>
      </tr>
      <tr>
        <td style="text-align: right;">150</td>
        <td style="text-align: right;">375</td>
      </tr>
      <tr>
        <td style="text-align: right;">200</td>
        <td style="text-align: right;">500</td>
      </tr>
      <tr>
        <td style="text-align: right;">250</td>
        <td style="text-align: right;">625</td>
      </tr>
      </table>
    </body>
  </html>
```

Není nutné najmout znuděného člověka, který by psal podobný kód a kterého by bylo nutné platit za odpracované hodiny, když lze použít levný a neúnavný počítač.

Cykly umožňují spouštět příkazy a bloky příkazů jazyka PHP opakovaně.

Cyklus while

Nejjednodušším typem cyklu v jazyce PHP je cyklus `while`. Spoléhá se na podmínku, podobně jako příkaz `if`. Rozdíl mezi cyklem `while` a příkazem `if` spočívá v tom, že u příkazu `if` se následný blok příkazů spustí jen jednou, pokud je podmínka pravdivá. Cyklus `while` provádí tento blok příkazů neustále dokola, dokud je podmínka pravdivá.

Cyklus `while` obvykle používáte, když dopředu nevíte, kolik iterací se provede, než přestane podmínka platit. Kdybyste potřebovali pevný počet iterací, měli byste zvolit spíše cyklus `for`.

Základní struktura cyklu `while` vypadá takto:

```
while (podminka) vyraz;
```

Následující cyklus `while` zobrazuje čísla od 1 do 5:

```
$num = 1;
while ($num <= 5) {
  echo $num."<br />";
  $num++;
}
```

Na začátku každé iterace ověřuje interpret PHP podmínku. Pokud ji vyhodnotí jako `false`, nespustí následný blok a ukončí cyklus. Potom provede příkaz, jenž následuje za celým cyklem.

Prostřednictvím cyklu `while` ale můžeme udělat něco užitečnějšího – například můžeme zobrazit tabulku dopravy z obrázku 1.7. Ve výpise 1.3 používáme cyklus `while` pro generování této tabulky.

Výpis 1.3. freight.php – Generování Bobovy tabulky dopravy v jazyce PHP

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Bobovy autodíly – Cena dopravy</title>
  </head>
  <body>
    <table style="border: 0px; padding: 3px">
    <tr>
      <td style="background: #cccccc; text-align: center;">Vzdálenost</td>
      <td style="background: #cccccc; text-align: center;">Cena</td>
    </tr>

    <?php
    $distance = 50;
    while ($distance <= 250) {
      echo "<tr>
            <td style=\"text-align: right;\">".$distance."</td>
            <td style=\"text-align: right;\">".($distance * 2.5)."</td>
            </tr>\n";
      $distance += 50;
    }
    ?>

    </table>
  </body>
</html>
```

Aby byl kód vygenerovaný skriptem dobře čitelný, musíte do něj vkládat zalomení a mezery. I když prohlížeče ignorují tyto bílé znaky, jsou důležité pro lidi, kteří budou číst kód. Často se budete dívat do vygenerovaného kódu HTML, pokud výstup nebude takový, jaký očekáváte.

Ve výpise 1.3 můžete vidět posloupnost znaků `\n` uvnitř textového řetězce. Pokud se nachází mezi uvozovkami, reprezentuje znak nového řádku.

Cykly for a foreach

V předchozí části jste použili cyklus `for` velmi rozšířeným způsobem. Nastavili jste počítadlo, přičemž před každou iterací ověřujete jeho hodnotu podmínkou. Na konci každé iterace upravíte jeho hodnotu.

Cyklus tohoto typu můžete zapsat kompaktněji pomocí cyklu `for`. Základní struktura cyklus `for` vypadá takto:

```
for (vyraz1; podminka; vyraz2)
  vyraz3;
```

- Interpret PHP vyhodnocuje výraz `vyraz1` na začátku. V něm obvykle nastavujete výchozí hodnotu počítadla.
- Podmínku ověřuje před každou iterací. Pokud podmínka vrátí `false`, cyklus končí. Zde většinou porovnáváte počítadlo s limitem.
- Výraz `vyraz3` spouští na konci každé iterace. V něm obvykle upravujete hodnotu počítadla.
- Výraz `výraz3` spouští pro každou iteraci. Jedná se většinou o blok příkazů, které tvoří tělo cyklu.

Cyklus `while` z výpisu 1.3 můžete přepsat na cyklus `for`. Kód PHP bude vypadat takto:

```
<?php
for ($distance = 50; $distance <= 250; $distance += 50) {
  echo "<tr>
        <td style=\"text-align: right;\">".$distance."</td>
        <td style=\"text-align: right;\">".($distance * 2.5)."</td>
        </tr>\n";
}
?>
```

Obě varianty, s cyklem `while` a `for`, fungují stejně. Zápis cyklu `for` je o něco kompaktnější – ušetříte dva řádky.

Oba cykly jsou ekvivalentní – žádný není lepší ani horší než ten druhý. Pro danou situaci můžete použít ten, který se vám bude zdát intuitivnější.

Můžete také spojit proměnnou proměnných s cyklem `for`, abyste zpracovali sérii opakujících se formulářových polí. Kdybyste kupříkladu měli formulářová pole s názvy `nazev1`, `nazev2`, `nazev3` atd., mohli byste je zpracovat takto:

```
for ($i = 1; $i <= $ciselnenazvy; $i++) {
  $temp = "nazev$i";
  echo htmlspecialchars($$temp).'<br />';
}
```

Díky tomu, že dynamicky vytváříte názvy proměnných, můžete přistupovat k jednotlivým formulářovým polím.

Kromě cyklu `for` existuje rovněž cyklus `foreach`, jenž byl navržený speciálně pro pole. O tomto cyklu bude pojednávat kapitola 3.

Cyklus do...while

Poslední typ cyklu, který si popíšeme, se chová trochu odlišně. Obecná struktura cyklu `do...while` vypadá takto:

```
do
  vyraz;
while (podminka);
```

Cyklus `do...while` se liší od cyklu `while` tím, že interpret PHP ověřuje podmínku až na konci iterace. To znamená, že příkaz nebo blok příkazů v tomto cyklu se provede vždy alespoň jedenkrát.

Dokonce i tehdy, když bude podmínka na začátku vyhodnocena jako `false`, a tudíž nikdy nenabude hodnoty `true`, tělo cyklu se provede jedenkrát před ověřením takové podmínky:

```
$cislo = 100;
do {
  echo $cislo."<br />";
} while ($num < 1);
```

Jak ukončit řídicí strukturu nebo skript

Pokud chcete ukončit provádění kusu kódu, můžete si vybrat ze tří přístupů, a to v závislosti na tom, čeho chcete dosáhnout.

Jestliže chcete ukončit provádění cyklu, můžete použít příkaz `break`, který už znáte z části o příkazu `switch`. Když použijete příkaz `break` v cyklu, provádění skriptu bude pokračovat na řádku, jenž následuje za tímto cyklem.

V případě, že byste chtěli přeskočit na další iteraci cyklu, mohli byste místo něj použít příkaz `continue`.

Kdybyste chtěli ukončit provádění celého skriptu PHP, mohli byste použít příkaz `exit`. Tento přístup je užitečný, když zpracováváte chyby. Mohli byste například upravit dříve uvedený příklad následovně:

```
if ($totalqty == 0) {
  echo 'Nic jste si neobjednal(a) na předchozí stránce.<br />';
  exit;
}
```

Zavolání příkazu `exit` přeruší provádění zbytku skriptu.

Alternativní syntaxe řídicích struktur

Pro všechny dosud popsané řídicí struktury existuje také alternativní syntaxe. Stačí nahradit otevírací závorku ({) dvojtečkou (:) a koncovou závorkou jedním z těchto klíčových slov: `endif`, `endswitch`, `endwhile`, `endfor` nebo `endforeach` (podle aktuální řídicí struktury). Pro cyklus `do…while` ale žádná alternativní syntaxe není.

Například tento kód:

```
if ($totalqty == 0) {
  echo 'Nic jste si neobjednal(a) na předchozí stránce.<br />';
  exit;
}
```

byste mohli přepsat do alternativní syntaxe s klíčovými slovy `if` a `endif`:

```
if ($totalqty == 0):
  echo 'Nic jste si neobjednal(a) na předchozí stránce.<br />';
  exit;
endif;
```

Řídicí struktura declare

Další řídicí struktura jazyka PHP, struktura `declare`, není tak často využívána jako ostatní konstrukce. Obecný tvar této řídicí struktury vypadá následovně:

```
declare (direktiva) {
  // blok
}
```

Tato struktura nastavuje **prováděcí direktivy** pro blok kódu – tj. pravidla, jak ho spustit. V současnosti existují jen dvě direktivy: `ticks` a `encoding`.

Například direktiva `ticks=`*n* vám umožňuje volat konkrétní funkci pro každých *n* řádků uvnitř bloku kódu, což může být užitečné při profilování a ladění.

Direktivou `encoding` nastavujete znakovou sadu pro daný skript:

```
declare(encoding='UTF-8');
```

V tomto případě nemusí za příkazem `declare` následovat blok kódu, pokud používáte jmenné prostory. O jmenných prostorech se dozvíte více později.

Řídicí struktura `declare` je zde uvedena pouze pro kompletnost. Příklady, jak používat funkce `tick`, najdete v kapitolách 25 a 26.

Co bude dál?

Nyní víte, jak získat objednávku zákazníka a jak s ní pracovat. V příští kapitole se naučíte, jak ukládat objednávku, abyste ji mohli načíst a vyřídit později.

KAPITOLA

2

Ukládání a načítání dat

Mezi hlavní témata této kapitoly patří:

- **ukládání dat,**
- **otevírání souboru,**
- **vytváření souboru a zápis do souboru,**
- **uzavírání souboru,**
- **čtení ze souboru,**
- **uzamykání souborů,**
- **mazání souborů,**
- **další užitečné souborové funkce,**
- **lepší řešení – systémy pro správu databází.**

Když už víte, jak pracovat s daty, která uživatel zadal do formuláře HTML, můžete se zaměřit na to, jak je uložit na později. Ve většině případů, včetně příkladu z předchozí kapitoly, budete chtít uložit tato data a načíst je později. V tomto případě musíte zapsat objednávku zákazníka do trvalého úložiště, aby mohla být vyřízena později.

V této kapitole se naučíte ukládat objednávku zákazníka z minulé kapitoly do souboru a načítat ji zpět. Zjistíte rovněž, proč to nemusí být vždy dobré řešení. Jestliže máte větší množství objednávek, měli byste raději používat systém pro správu databází, jakým je kupříkladu systém MySQL.

Ukládání dat

Data lze ukládat dvěma způsoby – do prostého souboru nebo do databáze.

Prostý soubor může mít různé formáty, ale obvykle se tímto termínem označuje běžný textový soubor. V příkladu z této kapitoly budete zapisovat objednávky zákazníků do textového souboru tak, že na každém řádku se bude nacházet jedna objednávka.

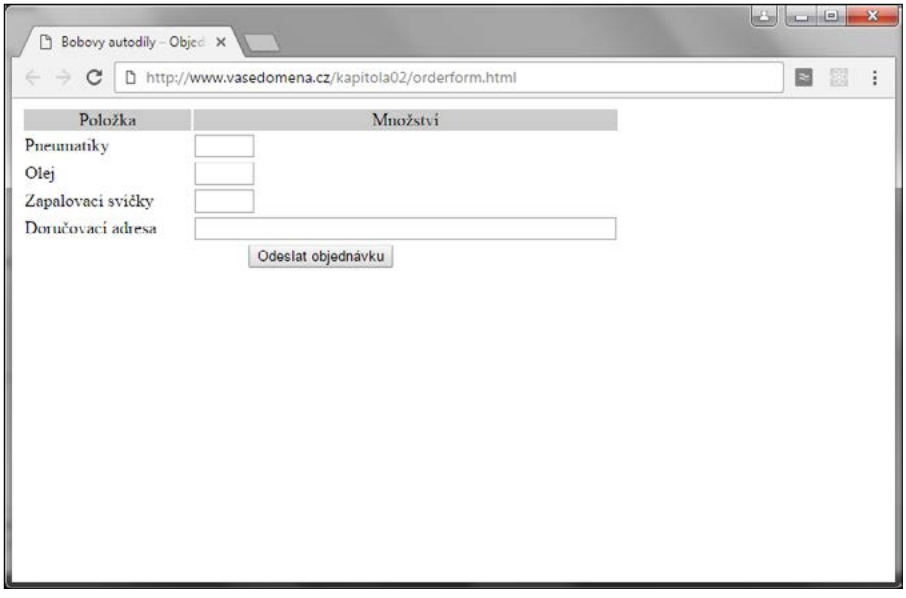
Zapisovat objednávky tímto způsobem je snadné, ale také omezující, jak zjistíte později v této kapitole. Pokud pracujete s větším množstvím dat, použijete pravděpodobně místo souboru databázi. Prosté soubory mají však také své místo – v některých situacích se vám jistě bude hodit, když budete vědět, jak s nimi pracovat.

Procesy zápisu do souborů a čtení z nich se podobají ve spoustě programovacích jazyků. Pokud jste někdy psali něco v jazyce C nebo například shellový skript, budou vám jistě známé.

Ukládání a načítání Bobových objednávek

V této kapitole použijete mírně upravenou verzi objednávkového formuláře z minulé kapitoly. Začněte tímto formulářem a kódem, který jste napsali v jazyce PHP pro zpracování dat objednávky.

Do daného formuláře jsme doplnili pole pro získání doručovací adresy zákazníka. Upravený formulář je možné vidět na obrázku 2.1.



Obrázek 2.1. Tato verze objednávkového formuláře zjišťuje doručovací adresu zákazníka

Formulářové pole pro doručovací adresu se nazývá `address`. Na serveru pak vznikne proměnná, k níž můžete přistupovat – `$_REQUEST['address']`, `$_POST['address']` nebo `$_POST['address']` (podle hodnoty atributu `method` u vašeho formuláře).

V této kapitole zapíšete všechny objednávky do jediného souboru. Potom vytvoříte webové rozhraní pro Bobovy zaměstnance, aby si v něm mohli prohlížet obdržené objednávky.

Zpracování souborů

Zápis dat do souborů vyžaduje tři kroky:

- Otevřete soubor. Pokud neexistuje, musíte ho vytvořit.
- Zapíšete data do souboru.
- Zavřete soubor.

Obdobně načítání dat ze souboru lze provést ve třech krocích:

- Otevřete soubor. Jestliže nemůžete otevřít soubor (například když neexistuje), musíte to detekovat a uhlazeně ukončit proces.
- Načtěte data ze souboru.
- Zavřete soubor.

Při načítání dat ze souboru máme na výběr z mnoha možností, které se liší v množství načtených dat najednou. Některé z nich si popíšeme podrobněji. Prozatím začneme otevřením souboru.

Otevírání souboru

Soubor otevřete pomocí funkce `fopen()`. Při jeho otevírání musíte specifikovat, jak ho hodláte používat – tzv. **režim souboru**.

Výběr režimu souboru

Operační systém na serveru musí vědět, co budete dělat se souborem, jejž otevíráte. Musí také vědět, zda jiný skript může otevřít tento soubor, zatímco ho máte otevřený, a jestli máte (nebo vlastník skriptu) oprávnění používat ho požadovaným způsobem. Režim souboru umožňuje operačnímu systému určit, jak přistupovat k požadavkům jiných lidí nebo skriptů, a umožňuje ověřit, zda máte oprávnění k příslušnému souboru.

Když otevíráte soubor, musíte učinit tři rozhodnutí:

- Můžete otevřít soubor pouze pro čtení, pouze pro zápis nebo pro čtení i zápis.
- Pokud zapisujete do souboru, můžete přepisovat současný obsah souboru nebo přidávat nová data na jeho konec. Případně můžete chtít ukončit provádění programu místo toho, abyste přepsali soubor, pokud tento soubor už existuje.
- Jestliže se pokoušíte zapisovat do souboru v systému, který rozlišuje mezi binárními a textovými soubory, nejspíše budete muset specifikovat tuto skutečnost.

Funkce `fopen()` podporuje kombinace všech tří možností.

Otevírání souboru funkcí fopen()

Předpokládejme, že budeme chtít zapsat objednávku do souboru. Můžeme tento soubor otevřít pro zápis takto:

```
$fp = fopen("$document_root/../orders/orders.txt", 'w');
```

Funkce `fopen()` očekává dva až čtyři argumenty. Obvykle postačí dva jako na výše uvedeném řádku kódu.

Prvním z nich je název souboru, který chceme otevřít. Můžeme definovat cestu k tomuto souboru – například v předchozím kódu se soubor `orders.txt` nachází v adresáři `orders`.

Použili jsme vestavěnou proměnnou `$_SERVER['DOCUMENT_ROOT']`, ale zkrátili jsme její název (podobně jako u formulářových proměnných).

Tato proměnná ukazuje na kořenový adresář webového serveru. Na tomto řádku používáme posloupnost `..`, což znamená, že přistupujeme k rodičovskému adresáři kořenového adresáře pro dokumenty. Tento adresář se nachází z bezpečnostních důvodů mimo strom dokumentů. V tomto případě nechceme, aby byl tento soubor dostupný přes web, ale pouze přes rozhraní, které vytvoříme. Takovou cestu nazýváme **relativní cesta**, jelikož definuje pozici v systému souborů relativně k jinému adresáři.

Podobně jako u zkrácených názvů formulářových proměnných musíte přidat následující řádek na začátek svého skriptu:

```
$document_root = $_SERVER['DOCUMENT_ROOT'];
```

Kopírujeme obsah dlouhé proměnné do proměnné s kratším názvem.

Je možné také definovat **absolutní cestu** k souboru. Jedná se o cestu z kořenového adresáře (`/` v systému Unix nebo *C:* v systému Windows). Na unixovém serveru by mohla tato cesta vypadat například takto: */data/orders*. Pokud neuvedeme žádnou cestu, předpokládáme, že soubor se nachází ve stejném adresáři jako samotný skript. Umístění cílového adresáře samozřejmě závisí na konfiguraci serveru.

V systému Unix používáme lomítka (`/`) v cestách. V systému Windows jsou běžnější zpětná lomítka (`\`). V případě zpětných lomítek je musíme **escapovat** (označit jako speciální znaky), aby je funkce `fopen()` přijala správně. Znak escapujeme tak, že před něj vložíme zpětné lomítko:

```
$fp = fopen("$document_root\\..\\orders\\orders.txt", 'w');
```

Jen málo vývojářů vkládá zpětná lomítka do cest, jelikož s lomítky bude váš kód fungovat pouze v operačním systému Windows. Jestliže použijete obyčejná lomítka, můžete většinou přesouvat kód mezi systémy Windows a Unix, aniž byste museli cokoliv měnit.

Druhým parametrem funkce `fopen()` je režim souboru, přičemž by se mělo jednat o textový řetězec. Tento textový řetězec definuje, co chcete dělat s daným souborem. V tomto případě předáváte režim souboru `'w'` funkci `fopen()` – to znamená, že otevíráte soubor pro zápis. Přehled režimů souboru najdete v tabulce 2.1.

Tabulka 2.1. Přehled režimů souboru pro funkci `fopen()`

Režim	Název režimu	Význam
`r`	Čtení	Otevírá soubor pro čtení, přičemž začíná od počátku souboru.
`r+`	Čtení	Otevírá soubor pro čtení a zápis, přičemž začíná od počátku souboru.
`w`	Zápis	Otevírá soubor pro zápis, přičemž začíná na začátku souboru. Pokud tento soubor už existuje, smaže jeho obsah. Jestliže neexistuje, pokusí se ho vytvořit.

Režim	Název režimu	Význam
w+	Zápis	Otevírá soubor pro zápis a čtení, přičemž začíná od počátku souboru.
x	Opatrný zápis	Otevírá soubor pro zápis, přičemž začíná od počátku souboru. Pokud tento soubor již existuje, neotevře ho, ale místo toho funkce fopen() vrátí hodnotu false a PHP vygeneruje varování.
x+	Opatrný zápis	Otevírá soubor pro zápis a čtení, přičemž začíná od počátku souboru. Jestliže soubor existuje, neotevře ho, ale místo toho funkce fopen() vrátí hodnotu false a PHP vygeneruje varování.
a	Doplnění	Otevírá soubor pouze pro doplnění (zápis), přičemž začíná na konci obsahu souboru, pokud existuje. V případě, že neexistuje, pokusí se ho vytvořit.
a+	Doplnění	Otevírá soubor pro doplnění (zápis) a čtení, přičemž začíná na konci obsahu souboru, pokud existuje. V případě, že neexistuje, pokusí se ho vytvořit.
b	Binární	Používá se v kombinaci s ostatními režimy. Jestliže váš systém souborů rozlišuje binární a textové soubory, pravděpodobně ho budete chtít použít. Například systém Window je rozlišuje, ale systém Unix nikoliv. Praxí osvědčeným postupem je vždy uvádět tento režim, abyste dosáhli maximální přenositelnosti zdrojového kódu. Tento režim je výchozí.
t	Textový	Používá se v kombinaci s ostatními režimy. Tento režim je k dispozici pouze pro systém Windows. Je lepší se mu zcela vyhnout – nastavte ho, jen dokud nepřepracujete svůj kód tak, aby fungoval s režimem b.

Volba režimu souboru závisí na způsobu práce se souborem. Ve výše uvedeném příkladu jsme použili režim 'w', díky čemuž můžeme uložit pouze jednu objednávku do daného souboru. Kdykoliv se pokusíme uložit další objednávku, přepíšeme tu předchozí. Toto chování není příliš logické, proto bychom měli definovat raději režim doplnění (ve spojení s binárním režimem):

```
$fp = fopen("$document_root\\..\\orders\\orders.txt", 'ab');
```

Třetí parametr funkce fopen() je volitelný. Můžeme ho použít v případě, že budeme chtít hledat soubor také v include_path (konfigurační možnost jazyka PHP, popsaná v příloze A). V takovém případě bychom přidělili tomuto parametru hodnotu true. Pokud řekneme skriptu PHP, aby hledal soubor v include_path, nemusíme uvádět název adresáře ani cestu:

```
$fp = fopen('orders.txt', 'ab', true);
```

Čtvrtý parametr je taktéž volitelný. Funkce fopen() umožňuje doplňovat před názvy souborů protokol (například http://) a otevírat vzdálená umístění. Některé protokoly přidávají ještě další parametr. Na tento případ užití funkce fopen() se zaměříme v příští části.

Jestliže funkce fopen() otevře soubor úspěšně, vrátí prostředek, což je jistý ukazatel na soubor. Ten bychom si měli uložit do proměnné – v tomto příkladu jej ukládáme do proměnné $fp. Pomocí ní budeme přistupovat k souboru, když z něj budeme chtít číst nebo do něj zapisovat.

Otevírání souborů přes FTP nebo HTTP

Funkcí `fopen()` lze otevírat nejen lokální soubory, ale také vzdálené soubory přes protokoly FTP, HTTP a jiné. Tuto možnost můžete zakázat tak, že vypnete direktivu `allow_url_fopen` v souboru *php.ini*. V případě, že narazíte na problémy při otevírání vzdálených souborů, zkontrolujte svůj soubor *php.ini*.

Jestliže název souboru začíná `ftp://`, vznikne pasivní spojení s FTP serverem a funkce `fopen()` vrátí ukazatel na začátek tohoto souboru.

Pokud název souboru začíná `http://`, vznikne HTTP spojení s webovým serverem a funkce `fopen()` vrátí ukazatel na odpověď.

Nezapomínejte, že u doménových jmen nezáleží na velikost písmen, ale u cest a názvů souborů už může být velikost písmen důležitá.

Řešení problémů s otevíráním souborů

Chybou může být snažit se otevřít soubor, k němuž nemáte práva čtení nebo zápisu. Tato chyba obvykle nastává v unixových operačních systémech, ale občas se s ní setkáte i v systému Windows. V takovém případě interpret PHP zobrazí varování (viz obrázek 2.2).

Obrázek 2.2. Interpret varuje, že soubor nelze otevřít

V případě, že obdržíte tuto chybovou zprávu, musíte ověřit, že uživatelský účet, pod kterým běží váš skript, má oprávnění pro přístup k příslušnému souboru. V závislosti na konfiguraci vašeho serveru může být váš skript spuštěn pod uživatelským účtem webového serveru nebo pod vlastníkem adresáře, v němž se tento skript nachází.

Na většině operačních systémů běží skripty pod uživatelským účtem webového serveru. Kdyby váš skript běžel na unixovém systému v adresáři *~/public_html/kapitola02/*, mohli byste kupříkladu vytvořit adresář s právem zápisu pro skupinu, do které byste ukládali objednávky:

```
mkdir cesta/k/objednavkam
chgrp apache cesta/k/objednavkam
chmod 775 cesta/k/objednavkam
```


Mohli byste rovněž změnit vlastníka souboru na uživatelský účet webového serveru. Někteří lidé si pravděpodobně vytvoří veřejně zapisovatelný soubor, aby si ulehčili práci, ale pamatujte si, že adresáře a soubory, do nichž může zapisovat kdokoliv, jsou nebezpečné. Adresáře přístupné z webu by nikdy neměly být zapisovatelné. Přesně z tohoto důvodu se váš adresář *orders* nachází mimo strom dokumentů. O bezpečnosti se dozvíte více v kapitole 15.

Špatně nastavená oprávnění jsou pravděpodobně nejčastějším zdrojem potíží při otevírání souborů, ale nejsou jediným. Pokud není možné otevřít soubor, musíte se to dozvědět, abyste se zbytečně nepokoušeli číst nebo zapisovat data.

Jestliže otevírání souboru selže, funkce `fopen()` vrátí hodnotu `false`. Navíc interpret PHP vypustí chybu s úrovní varování (`E_WARNING`). S touto chybou se můžete vypořádat lépe, když potlačíte tuto chybovou zprávu a zobrazíte svou vlastní:

```
@$fp = fopen("$document_root/../orders/orders.txt", 'ab');

if (!$fp) {
  echo "<p><strong>Vaše objednávka nemohla být zpracována. Zkuste to
       prosím později.</strong></p>";
  exit;
}
```

Operátorem `@` před voláním funkce `fopen()` sdělujeme jazyku PHP, aby potlačil všechny případné chyby, které vzniknou při jejím zpracování. Většinou bychom uvítali, kdybychom se dozvěděli, že se něco pokazilo, ale v tomto případě se vypořádáme s tímto problémem jinak.

Mohli bychom přepsat tento řádek následovně:

```
$fp = @fopen("$document_root/../orders/orders.txt", 'ab');
```

U tohoto zápisu je méně nápadné, že používáme operátor potlačení chyby, proto bude složitější náš kód odladit.

POZNÁMKA

Použití operátoru potlačení chyby bývá považované za špatný styl, proto na něj pohlížejte jako na zjednodušení. Zde popsaný způsob je nejjednodušší verzí zpracování chyb. Elegantnější řešení zpracování chyb najdete v kapitole 7.

Příkazem `if` ověřujeme proměnnou `$fp`, abychom zjistili, jestli funkce `fopen()` vrátila platný ukazatel na soubor. Pokud ho nevrátila, vypisujeme chybovou zprávu a ukončujeme provádění skriptu.

Výstup lze vidět na obrázku 2.3.

Obrázek 2.3. Použití vlastních chybových zpráv namísto standardních zpráv jazyka PHP je uživatelsky přívětivější

Zapisování do souboru

Zapisování do souboru je v jazyce PHP relativně lehká úloha. Můžeme použít buď funkci `fwrite()` (zkratka pro **file write**), nebo funkci `fputs()` (zkratka pro **file put string**), přičemž `fputs()` je alias pro `fwrite()`. Funkci `fwrite()` voláme takto:

```
fwrite($fp, $outputstring);
```

Touto funkcí zapisujeme textový řetězec uložený v proměnné `$outputstring` do souboru, na nějž ukazuje proměnná `$fp`.

Alternativu k funkci `fwrite()` představuje funkce `file_put_contents()`, která má následující prototyp:

```
int file_put_contents(string nazevsouboru,
                      mixed data
                      [, int priznaky[
                      [, resource kontext]])
```

Tato funkce zapisuje *data* do souboru *nazevsouboru*, aniž by bylo nutné volat funkci `fopen()` nebo funkci `fclose()`. Kromě toho má také své dvojče – funkci `file_get_contents()`. Volitelné parametry *priznaky* a *kontext* obvykle zužitkujete, když budete zapisovat do vzdálených souborů pomocí protokolu HTTP nebo FTP (s těmito funkcemi se ještě setkáte v kapitole 18).

Parametry funkce fwrite()

Funkce `fwrite()` má tři parametry, ale třetí z nich je volitelný. Prototyp funkce `fwrite()` vypadá následovně:

```
int fwrite( resource ukazatel, string retezec[, int delka])
```

Třetí parametr, *delka*, reprezentuje maximální počet bajtů pro zápis. Pokud ho definujeme, funkce `fwrite()` bude zapisovat *retezec* do souboru, na nějž se odkazuje *ukazatel*, dokud nedosáhne konce tohoto textového řetězce nebo nezapíše *delka* bajtů (podle toho, které omezení přijde na řadu jako první).

Délku zapisovaného textového řetězce je možné zjistit prostřednictvím vestavěné funkce `strlen()`:

```
fwrite($fp, $outputstring, strlen($outputstring));
```

Třetí parametr pravděpodobně použijete při zápisu v binárním režimu, jelikož pomáhá zamezit problémům s kompatibilitou mezi platformami.

Formáty souborů

Když vytváříte datový soubor jako v předchozím příkladu, záleží jen na vás, jaký formát zvolíte. (Kdybyste se však chystali použít tento datový soubor v jiné aplikaci, museli byste se řídit formátem této aplikace.)

Nyní sestavíme textový řetězec, jenž bude představovat jediný záznam v tomto datovém souboru. Uděláme to kupříkladu takto:

```
$outputstring = $date."\t".$tireqty." pneumatik\t".
                $oilqty." lahví oleje\t".
                $sparkqty." zapalovacích svíček\t".$totalamount."\t".
                $address."\n";
```

V tomto jednoduchém příkladu ukládáme každý záznam na samostatný řádek v souboru. Uložením záznamu na samostatný řádek získáme prostý oddělovač záznamů a tím bude znak nového řádku. Protože znak nového řádku je neviditelný, zapisujeme ho pomocí řídicí sekvence `\n`.

V této knize zapisujeme datová pole ve stejném pořadí, přičemž je oddělujeme znakem tabulátoru. A jelikož znak tabulátoru je rovněž neviditelný, zapisujeme ho prostřednictvím řídicí sekvence `\t`.

Můžete si ale vybrat jakýkoliv smysluplný oddělovač, který bude snadno čitelný. Oddělovačem by mělo být něco, co se jistě nevyskytne ve vstupních datech, případně byste měli odstranit nebo escapovat všechny výskyty oddělovače. Pokud si prohlédnete kompletní zdrojový kód tohoto příkladu, zjistíte, že odstraňuje problematické znaky pomocí funkce `preg_replace` (pracující s regulárním výrazem). Více se naučíte o zpracování vstupních dat v kapitole 4.

Speciální oddělovač polí nám pomůže rozdělit data zpět do samostatných proměnných, až budeme v budoucnu tato data načítat. Tomu se budeme věnovat v kapitolách 3 a 4. Prozatím budeme s objednávkou zacházet jako s jediným textovým řetězcem.

Až zpracujeme několik objednávek, obsah tohoto souboru bude vypadat podobně jako v ukázkovém výpise 2.1.

Výpis 2.1. orders.txt – Ukázka, co může obsahovat objednávkový soubor

```
6. 1. 2017 15:51  4 pneumatik  1 lahví oleje  6 zapalovacích svíček
  520.8 Kč  Krátká 22, Malé město
6. 1. 2017 15:51  1 pneumatik  0 lahví oleje  0 zapalovacích svíček
  120 Kč    Hlavní 33, Staré město
6. 1. 2017 15:53  0 pneumatik  1 lahví oleje  4 zapalovacích svíček
  31.2 Kč   Úzká 127, Stonava
```

Zavírání souboru

Až přestanete pracovat se souborem, musíte ho zavřít. K tomu byste měli zavolat funkci `fclose()`:

```
fclose($fp);
```

Tato funkce vrací hodnotu `true`, jestliže byl soubor úspěšně zavřen, nebo `false`, pokud nebyl. Tento proces je mnohem méně náchylný k chybám než otevírání souboru, proto zde netestujeme návratovou hodnotu.

Kompletní kód pro finální verzi skriptu *processorder.php* je k dispozici ve výpise 2.2.

Výpis 2.2. processorder.php – Finální verze skriptu pro zpracování objednávek

```
<?php
  // vytváříme zkrácené názvy proměnných
  $tireqty = (int) $_POST['tireqty'];
  $oilqty = (int) $_POST['oilqty'];
  $sparkqty = (int) $_POST['sparkqty'];
  $address = preg_replace('/\t|\R/',' ',$_POST['address']);
  $document_root = $_SERVER['DOCUMENT_ROOT'];
  $date = date('j. n. Y H:i');
?>
<!DOCTYPE html>
<html>
```

```
<head>
  <meta charset="UTF-8" />
  <title>Bobovy autodíly – Výsledek objednávky</title>
</head>
<body>
  <h1>Bobovy autodíly</h1>
  <h2>Výsledek objednávky</h2>
  <?php
    echo "<p>Objednávka byla zpracována ".$date."</p>";
    echo '<p>Vaše objednávka:</p>';

    $totalqty = 0;
    $totalamount = 0.00;

    define('TIREPRICE', 100);
    define('OILPRICE', 10);
    define('SPARKPRICE', 4);

    $totalqty = $tireqty + $oilqty + $sparkqty;
    echo "<p>Objednáno položek: ".$totalqty."<br />";

    if ($totalqty == 0) {
      echo "Nic jste si neobjednal(a) na předchozí stránce.<br />";
    } else {
      if ($tireqty > 0) {
        echo 'pneumatik: '.htmlspecialchars($tireqty).'<br />';
      }
      if ($oilqty > 0) {
        echo 'lahví oleje: '.htmlspecialchars($oilqty).'<br />';
      }
      if ($sparkqty > 0) {
        echo 'zapalovacích svíček: '.htmlspecialchars($sparkqty).'<br />';
      }
    }


    $totalamount = $tireqty * TIREPRICE
                 + $oilqty * OILPRICE
                 + $sparkqty * SPARKPRICE;

    echo "Cena: ".number_format($totalamount, 2)." Kč<br />";

    $taxrate = 0.20;  // výše DPH je 20%
    $totalamount = $totalamount * (1 + $taxrate);
    echo "Celková cena s DPH: ".number_format($totalamount,2)." Kč</p>";

    echo "<p>Doručovací adresa: ".htmlspecialchars($address)."</p>";

    $outputstring = $date."\t".$tireqty." pneumatik\t".
                    $oilqty." lahví oleje\t".
                    $sparkqty." zapalovacích svíček\t".$totalamount." Kč\t".
```

```
                    $address."\n";

      // otevřeme soubor pro doplnění
      @$fp = fopen("$document_root/../orders/orders.txt", 'ab');

      if (!$fp) {
        echo "<p><strong>Vaše objednávka nemohla být zpracována. Zkuste to
             prosím později.</strong></p>";
        exit;
      }

      flock($fp, LOCK_EX);
      fwrite($fp, $outputstring, strlen($outputstring));
      flock($fp, LOCK_UN);
      fclose($fp);

      echo "<p>Objednávka byla uložena.</p>";
    ?>
  </body>
</html>
```

Čtení ze souboru

Prozatím můžou vidět přes web zpracované objednávky jen Bobovi zákazníci, ale když se na ně budou chtít podívat Bobovi zaměstnanci, musí si otevřít soubor s objednávkami.

Vytvořme tedy webové rozhraní, v němž by si mohli přečíst obsah tohoto souboru Bobovi zaměstnanci. Kód tohoto rozhraní se nachází ve výpise 2.3.

Výpis 2.3. vieworders.php – Rozhraní s objednávkami pro zaměstnance

```
<?php
  // vytváříme zkrácené názvy proměnných
  $document_root = $_SERVER['DOCUMENT_ROOT'];
?>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Bobovy autodíly – Výsledek objednávky</title>
  </head>
  <body>
    <h1>Bobovy autodíly</h1>
    <h2>Objednávky zákazníků</h2>
    <?php
      @$fp = fopen("$document_root/../orders/orders.txt", 'rb');
      flock($fp, LOCK_SH); // uzamkneme soubor pro čtení

      if (!$fp) {
        echo "<p><strong>Žádné nevyřízené objednávky.<br />
             Zkuste to prosím později.</strong></p>";
```

```
        exit;
      }

      while (!feof($fp)) {
        $order = fgets($fp);
        echo htmlspecialchars($order)."<br />";
      }

      flock($fp, LOCK_UN); // uvolníme zámek čtení
      fclose($fp);
    ?>
  </body>
</html>
```


Tento skript se řídí dříve popsanou posloupností akcí:

- otevřít soubor
- načíst data ze souboru
- zavřít soubor

Výstup skriptu z výpisu 2.3 si můžete prohlédnout na obrázku 2.4.

Obrázek 2.4. Skript vieworders.php zobrazuje všechny objednávky ze souboru orders.txt v okně webového prohlížeče

Zaměřme se na funkce tohoto skriptu podrobněji.

Otevření souboru pro čtení – fopen()

Opět otevíráme soubor funkcí `fopen()`. V tomto případě ho otevíráme pro čtení, a proto volíme režim `'rb'`:

```
$fp = fopen("$document_root/../orders/orders.txt", 'rb');
```

Příznak zastavení – feof()

V tomto příkladu čteme data ze souboru v cyklu `while`, dokud se nedostaneme na konec souboru. Konec souboru testujeme v podmínce cyklu `while` prostřednictvím funkce `feof()`:

```
while (!feof($fp))
```

Funkce `feof()` má ukazatel souboru jako svůj jediný parametr. Vrací `true`, jestliže tento ukazatel souboru dosáhne konce souboru. Přestože se může název této funkce zdát zvláštní, lze si ho snadno zapamatovat, protože se jedná o zkratku **File End of File**.

Tento případ užití je poměrně rozšířený – ze souboru čteme data, dokud nenarazíme na konec souboru (`EOF`).

Čtení po řádcích – fgets(), fgetss() a fgetcsv()

V předchozím příkladu čteme ze souboru pomocí funkce `fgets()`:

```
$order = fgets($fp);
```

Ta načítá řádek ze souboru. V tomto případě čte, dokud nenarazí na znak nového řádku (`\n`) nebo `EOF`.

Na výběr máme z mnoha různých funkcí pro čtení souborů. Funkce `fgets()` je kupříkladu užitečná, když pracujeme se soubory s běžným textem, který chceme zpracovat po částech.

Zajímavou variantou je funkce `fgetss()`, jež má následující prototyp:

```
string fgetss(resource fp[, int delka [, string pripustne_znacky]]);
```

Tato funkce se podobá funkci `fgets()` až na to, že ořezává značky HTML a PHP obsažené v textu. Pokud chcete ponechat určité značky, zahrňte je do textového řetězce *pristupne_znacky*. Tuto funkci byste měli používat kvůli bezpečnosti, pokud načítáte soubor napsaný někým jiným nebo obsahující uživatelský vstup. Neomezený kód HTML v souboru by mohl narušit pečlivě zvolené formátování. Neomezený kód PHP nebo JavaScriptu by mohl využít zákeřný uživatel k narušení zabezpečení.

Funkce `fgetcsv()` představuje další variantu funkce `fgets()`. Má následující prototyp:

```
array fgetcsv (resource fp, int delka [, string oddelovac
              [, string obal[, string escapovaci_znak]]])
```

S touto funkcí můžete rozdělovat řádky podle oddělovače, jímž může být kupříkladu tabulátor (jako v předchozím příkladu) nebo čárka (jak bývá běžné v tabulkových procesorech a jiných aplikacích). Pokud chcete získat samostatné proměnné z objednávky namísto celých řádků textu, pomocí funkce `fgetcsv()` toho dosáhnete snadno. Zavoláte ji podobně, jako byste volali funkci `fgets()`, ale předáte jí oddělovač, kterým jste oddělili jednotlivá pole. Například:

```
$order = fgetcsv($fp, 0, "\t");
```

Tímto kódem bychom získali řádek ze souboru a rozdělili ho tabulátory (`\t`). Výsledkem by bylo pole (v tomto příkladu `$order`). Pole si popíšeme podrobněji v kapitole 3.

Parametr *`delka`* by měl být větší než počet znaků nejdelšího řádku, který chcete načítat, nebo `0` v případě, že nechcete omezovat délku řádku.

Parametrem *`obal`* definujete znaky, jimiž jsou obalená jednotlivá pole. Jestliže ho neuvedete, získá výchozí hodnotu `"` (uvozovku).

Čtení celého souboru – readfile(), fpassthru(), file() a file_get_contents()

Nemusíte číst soubor po řádcích, ale můžete ho načíst celý najednou. Máte k dispozici čtyři různé postupy.

První z nich používá funkci `readfile()`. Můžete nahradit téměř celý skript, který jste napsali, jedním řádkem:

```
readfile("$document_root/../orders/orders.txt");
```

Funkcí `readfile()` otevřeme soubor, vypíšeme jeho obsah do standardního výstupu (do webového prohlížeče) a zavřeme soubor. Její prototyp vypadá takto:

```
int readfile(string nazev_souboru, [bool pouzit_include_path
            [, resource kontext]]);
```

Volitelným druhým parametrem je možné určit, jestli by se měl interpret PHP poohlížet po souboru v `include_path` (funguje stejně jako u funkce `fopen()`). Volitelný parametr *`kontext`* slouží pouze pro načítání vzdálených souborů – například přes protokol HTTP. Tento případ užití najdete v kapitole 18. Tato funkce vrací celkový počet načtených bajtů.

Druhý postup spočívá v použití funkce `fpassthru()`. Nejprve však musíte otevřít soubor prostřednictvím funkce `fopen()`. Potom můžete předat ukazatel na soubor jako argument funkci `fpassthru()`, jež vypisuje obsah souboru do standardního výstupu. Jakmile skončí, automaticky zavře daný soubor.

Funkci `fpassthru()` můžete zavolat následovně:

```
$fp = fopen("$document_root/../orders/orders.txt", 'rb');
fpassthru($fp);
```

Funkce `fpassthru()` vrací hodnotu `true`, pokud operace čtení proběhne úspěšně; v opačném případě vrací hodnotu `false`.

Třetí způsob čtení celého souboru představuje funkce `file()`. Tato funkce se podobá funkci `readfile()`, ale nevypisuje obsah souboru do standardního výstupu – převádí ho na pole. O této funkci se dozvíte více v kapitole 3, věnované polím. Jen pro úplnost – zavolali byste ji takto:

```
$souborovepole = file("$document_root/../orders/orders.txt");
```

Na výše uvedeném řádku načítáme obsah celého souboru do pole s názvem `$souborovepole`. Každý řádek tohoto souboru tvoří samostatný prvek tohoto pole. Tato funkce nebyla určena pro načítání binárních souborů ve starších verzích jazyka PHP.

Čtvrtou možností je použít funkci `file_get_contents()`. Tato funkce se podobá funkci `readfile()`, ale vrací obsah souboru jako textový řetězec.

Čtení znaku – fgetc()

Další možností zpracování souboru je čtení po znacích. Toho dosáhnete pomocí funkce `fgetc()`. Ta přijímá ukazatel na soubor jako jediný argument a vrací další znak v tomto souboru. Mohli byste kupříkladu přepsat cyklus `while` v původním skriptu tak, aby používal funkci `fgetc()`:

```
while (!feof($fp)) {
  $char = fgetc($fp);
  if (!feof($fp)) {
    echo ($char == "\n" ? "<br />" : $char);
  }
}
```

V tomto kódu čteme soubor po znacích prostřednictvím funkce `fgetc()`, dokud nedosáhneme konce souboru, přičemž právě načtený znak ukládáme do proměnné `$char`. Všechny výskyty znaku konce řádku nahrazujeme zalomením řádku v jazyce HTML (`<br />`).

To děláme pro lepší formátování výstupu. Kdybychom vypsali soubor se znaky konce řádku, celý soubor by se zobrazil v prohlížeči na jediném řádku. Webové prohlížeče nezobrazují bílé znaky, kterým je i znak konce řádku, proto je musíme nahradit značkami `<br />`. Za tímto účelem můžeme použít ternární operátor.

Mírnou nevýhodou funkce `fgetc()` oproti funkci `fgets()` je, že funkce `fgetc()` vrací znak `EOF`, zatímco funkce `fgets()` nikoliv. Proto musíme znovu testovat podmínku konce souboru funkcí `feof()` poté, co načteme znak, jelikož nechceme vypsat znak `EOF` do prohlížeče.

Čtení souboru po znacích není obvykle efektivní řešení, pokud nemáte dobrý důvod k tomu, abyste zpracovali celý soubor znak po znaku.

Čtení libovolně dlouhých dat – fread()

Poslední možností, jak číst data ze souboru, je načíst libovolný počet bajtů funkcí `fread()`, jež má následující prototyp:

```
string fread(resource fp, int delka);
```

Tato funkce čte *`delka`* bajtů, dokud nenarazí na konec souboru nebo síťového paketu – podle toho, co přijde dřív.

Další souborové funkce

Někdy můžou být užitečné i další souborové funkce. Některé z nich si uvedeme v této části kapitoly.

Ověření existence souboru – file_exists()

Jestliže chcete ověřit, zda soubor existuje, aniž byste ho otevřeli, můžete zavolat funkci `file_exists()`:

```
if (file_exists("$document_root/../orders/orders.txt")) {
  echo 'Máte nevyřízené objednávky.';
} else {
  echo 'Žádné nové objednávky.';
}
```

Určení velikosti souboru – filesize()

Jak je soubor velký, zjistíte pomocí funkce `filesize()`:

```
echo filesize("$document_root/../orders/orders.txt");
```

Vrací velikost souboru v bajtech. Lze ji kombinovat s funkcí `fread()`, která čte celý soubor (nebo jeho část). Můžete dokonce nahradit původní skript následujícím:

```
$fp = fopen("$document_root/../orders/orders.txt", 'rb');
echo nl2br(fread($fp, filesize("$document_root/../orders/orders.txt")));
fclose($fp);
```

Funkce `nl2br()` převádí znaky `\n` na zalomení řádku v jazyce HTML (značky `<br />`).

Smazání souboru – unlink()

V případě, že budete chtít smazat soubor, který už jste zpracovali, můžete použít funkci `unlink()`. (Funkce `delete()` neexistuje.) Například takto:

```
unlink("$document_root/../orders/orders.txt");
```

Tato funkce vrací hodnotu `false`, jestliže není možné smazat vybraný soubor. To obvykle nastává, pokud skript nemá dostatečná oprávnění nebo tento soubor neexistuje.

Navigování v souboru – rewind(), fseek() a ftell()

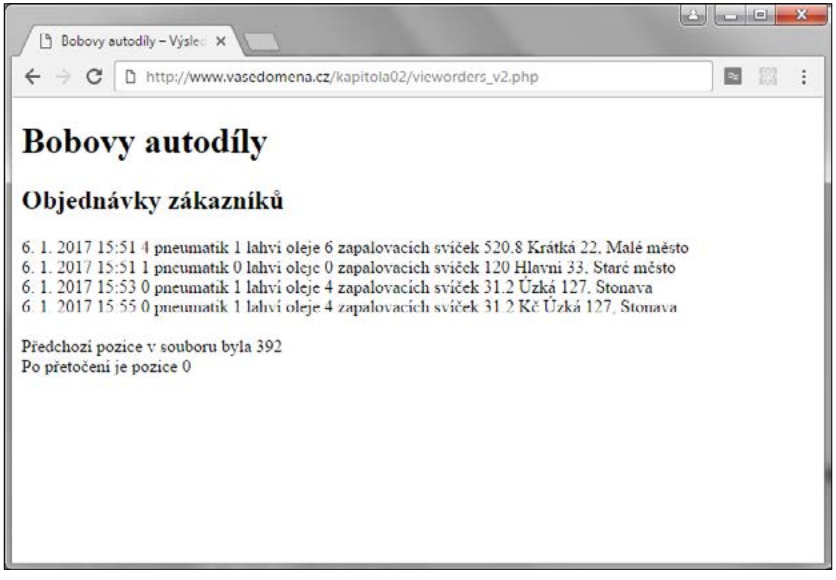
Pozici uvnitř soubor můžete měnit a zjišťovat funkcemi `rewind()`, `fseek()` a `ftell()`.

Funkce `rewind()` resetuje ukazatel souboru na začátek souboru. Funkce `ftell()` reportuje, jak daleko v souboru se ukazatel nachází, a to v bajtech. Kupříkladu můžete přidat níže uvedené řádky na spodek původního skriptu (před příkaz `fclose()`):

```
echo 'Předchozí pozice v souboru byla '.(ftell($fp));
echo '<br />';
```

```
rewind($fp);
echo 'Po přetočení je pozice '.(ftell($fp));
echo '<br />';
```

Výstup prohlížeče by měl vypadat podobně jako na obrázku 2.5.



Obrázek 2.5. Po přečtení objednávek bude ukazatel umístěný na konci souboru – na pozici 392 bajtů. Přetočením na začátek souboru se vrátí na pozici 0.

Pomocí funkce `fseek()` můžete nastavit ukazatel na nějaký bod v souboru. Její prototyp vypadá následovně:

```
int fseek(resource fp, int posun[, int pocatek])
```

Funkcí `fseek()` můžeme nastavit ukazatel souboru *`fp`* na pozici vzdálenou *`posun`* bajtů od bodu *`pocatek`*. Výchozí hodnotou volitelného parametru *`pocatek`* je `SEEK_SET`, která odpovídá začátku souboru. Dalšími přípustnými hodnotami jsou `SEEK_CUR` (aktuální pozice ukazatele souboru) a `SEEK_END` (konec souboru).

Výsledek zavolání funkce `rewind()` je stejný jako výsledek volání funkce `fseek()` s posunem `0`. Prostřednictvím funkce `fseek()` lze kupříkladu vyhledat prostřední záznam v souboru nebo provádět binární vyhledávání. Jestliže někdy dosáhnete takové úrovně složitosti datového souboru, že byste potřebovali takové nástroje, usnadníte si život, pokud použijete databázi (stvořenou pro tento účel).

Zamykání souborů

Představte si situaci, v níž se budou dva zákazníci snažit objednat ve stejnou chvíli. (Taková situaci není nijak neobvyklá, obzvláště tehdy, když vaše webové stránky získají slušnou

návštěvnost.) Co když jeden zákazník zavolá funkci `fopen()` a začne zapisovat a druhý zákazník také zavolá funkci `fopen()` a začne zapisovat? Jak budete vypadat výsledný obsah souboru? Bude se objednávka prvního zákazník nacházet před druhou, nebo naopak? Zapíše se jen první nebo druhá objednávka? Nebo se objednávky nějakým způsobem prolnou do sebe navzájem? Odpověď závisí na mnoha faktorech a mnohdy nelze odpovědět vůbec.

Těmto problémům se vyhnete, pokud začnete soubory zamykat. K tomuto účelu slouží funkce `flock()`. Měli byste ji volat poté, co otevřete soubor, ale předtím, než z něj začnete číst nebo do něj zapisovat.

Prototyp funkce `flock()` vypadá takto:

```
bool flock(resource fp, int operace [, int &blokujici])
```

Musíte jí předat ukazatel na soubor a konstantu reprezentující typ zámku. Tato funkce vrací hodnotu `true`, pokud se podařilo soubor uzamknout; v opačném případě vrací hodnotu `false`. Třetí parametr bude obsahovat hodnotu `true`, jestliže by nastavení zámku způsobilo zablokování aktuálního procesu (musel by čekat).

Přípustné hodnoty parametru *operace* jsou uvedené v tabulce 2.2.

Tabulka 2.2. Možné operace pro funkci `flock()`

Konstanta operace	Význam
LOCK_SH	Zámek čtení. Soubor lze sdílet s dalšími čtenáři.
LOCK_EX	Zámek zápisu. Tato operace je exkluzivní – soubor není možné sdílet.
LOCK_UN	Uvolnění současného zámku.
LOCK_NB	Zamezení blokování při pokusu o získání zámku (není podporované operačním systémem Windows).

Jestliže se rozhodnete pro funkci `flock()`, musíte ji přidat do všech skriptů, které pracují s tímto souborem; jinak bude k ničemu.

Zapamatujte si, že funkce `flock()` nepracuje s protokolem NFS ani jinými síťovými souborovými systémy. Také nefunguje v kombinaci se zastaralými souborovými systémy, které nepodporují zamykání – například se systémem FAT. V některých operačních systémech bývá implementovaná na úrovni procesu, takže nepodporuje rozhraní API s více vlákny.

Můžete ji zapojit i do příkladu s objednávkami, když změníte *processorder.php* následovně:

```
@$fp = fopen("$document_root/../orders/orders.txt", 'ab');

if (!$fp) {
  echo "<p><strong>Vaše objednávka nemohla být zpracována. Zkuste to
       prosím později.</strong></p>";
  exit;
}
```

```
flock($fp, LOCK_EX);
fwrite($fp, $outputstring, strlen($outputstring));
flock($fp, LOCK_UN);
fclose($fp);
```

Měli byste rovněž přidat zámky do skriptu *vieworders.php*:

```
@$fp = fopen("$document_root/../orders/orders.txt", 'rb');
flock($fp, LOCK_SH); // uzamkneme soubor pro čtení
// čtení ze souboru
flock($fp, LOCK_UN); // uvolníme zámek čtení
fclose($fp);
```

Kód je nyní robustnější, ale stále není dokonalý. Co kdyby se dva skripty pokusily získat zámek ve stejnou chvíli? Došlo by k **souběhu** (**race condition**), v němž procesy soupeří o zámek, ale není jisté, který z nich uspěje. To může způsobit řadu problémů. Mnohem lepší by bylo použít databázi.

Lepší řešení – databáze

Doposud jsme v našich příkladech pracovali s prostými textovými soubory. V druhé části této knihy se zaměříme na to, jak používat MySQL – systém pro správu relačních databází (RDBMS). Proč by nás měl vůbec zajímat?

Problémy textových souborů

Textové soubory trpí následujícími problémy:

- Jak narůstá velikost souboru, zpomaluje se rychlost jeho zpracování.
- Hledat konkrétní záznam ve skupině záznamů textového souboru je těžké. Jestliže jsou záznamy nějak uspořádané, můžete použít binární vyhledávání v kombinaci s pevnou velikostí záznamů, abyste mohli hledat dle klíčového pole. Kdybyste hledali vzory (například zákazníky, kteří žijí ve Starém Městě), museli byste načíst a postupně kontrolovat všechny záznamy.
- Vypořádání se se současným přístupem více procesů může být problematické. Už víte, jak zamykat soubory, ale zamykání může způsobit souběh. Může se také stát úzkým místem. S dostatečnou návštěvností webových stránek může velká skupina uživatelů čekat na odemčení souboru, aby mohli odeslat své objednávky. Pokud budou čekat příliš dlouho, pravděpodobně půjdou nakupovat jinam.
- Veškeré dosud popsané zpracování souborů bylo sekvenční – tj. začínáte od začátku souboru a čtete data, až dojdete na jeho konec. Vkládání nebo mazání záznamů z prostředku souboru (náhodný přístup) může být složité, jelikož nakonec načtete celý soubor do paměti, uděláte změny a potom uložíte celý soubor. U velkých datových souborů to představuje příliš velké režijní náklady.

- Kromě oprávnění přístupu k souborům nemáte k dispozici žádný snadný způsob, jak řídit různé úrovně přístupu k datům.

Jak databázové systémy řeší tyto problémy

Systémy pro správu relačních databází řeší všechny tyto problémy:

- Poskytují mnohem rychlejší přístup k datům než textové soubory. Systém MySQL (používaný v této knize) je podle spousty benchmarků rychlejší než jiné databázové systémy.
- Lze se v nich dotazovat na skupinu dat, která splňují určitá kritéria.
- Mají vestavěné mechanizmy pro nakládání se současným přístupem, takže jako programátoři se o to nemusíte starat.
- Umožňují náhodný přístup k datům.
- Mají vestavěný systém oprávnění. Systém MySQL má jisté silné stránky v této oblasti.

Pravděpodobně hlavním důvodem pro použití databázového systém je, že veškerou funkčnost datového úložiště již máte implementovanou. Samozřejmě byste si mohli napsat vlastní knihovnu funkcí, ale proč znovu vynalézat kolo?

V druhé části této knihy zjistíte, jak relační databáze fungují, jak nastavit systém MySQL a jak ho používat pro tvorbu webových stránek opírajících se o databáze.

Jestliže programujete jednoduchý systém a myslíte si, že nepotřebujete složitý databázový systém, ale chcete se vyhnout zamykání a jiným problémům spojeným s používání textového souboru, můžete použít rozšíření SQLite jazyka PHP. Toto rozšíření poskytuje SQL rozhraní pro textové soubory. Tato kniha se zaměřuje na systém MySQL, ale kdybyste chtěli získat více informací o rozšíření SQLite, najdete je na adrese *http://sqlite.org/* a *http://php.net/sqlite*.

Další zdroje

Více informací o práci se systémem souborů najdete v kapitole 17. V ní se dozvíte, jak měnit oprávnění, vlastnictví a názvy souborů, jak pracovat s adresáři a s prostředím systému souborů.

Případně si můžete přečíst sekci online dokumentace jazyka PHP věnovanou systému souborů – na internetové adrese *http://php.net/sqlite*.

Co bude dál

V příští kapitole se dozvíte vše, co potřebujete vědět o polích, a také to, jak s nimi zpracovávat data v PHP skriptech.

KAPITOLA

3

Pole

V této kapitole se dozvíte, jak používat důležité programovací konstrukce – pole. Proměnné v předchozích kapitolách byly **skalární** proměnné, které ukládají jedinou hodnotu. Pole je proměnná, jež ukládá skupinu hodnot. Pole může mít spoustu prvků, přičemž každý z nich může držet jedinou hodnotu – například text, číslo nebo další pole. Pole obsahující jiná pole se nazývají **vícerozměrná pole**.

Jazyk PHP podporuje pole s číselnými a textovými klíči. Pokud jste někdy programovali v jiném programovacím jazyku, pravděpodobně znáte pole s číselnými klíči, ale nejspíš jste nikdy neslyšeli o polích s textovými klíči – možná jste se setkali s podobnými konstrukcemi, jako jsou hašovací tabulky, mapy nebo slovníky. Všem prvkům nemusíte přiřazovat číselné indexové pořadí, ale můžete používat slova nebo jiné smysluplné informace.

V této kapitole budete dále vyvíjet Bobovy stránky a přitom se naučíte, pomocí polí, snadněji zpracovávat opakující se informace – například objednávky zákazníků. Díky tomu budete psát kratší a čistější kód, který bude řešit podobné úkoly, jaké jste dělali v minulé kapitole se soubory.

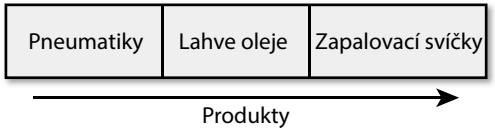
Hlavními tématy této kapitoly jsou:

- **číselně indexovaná pole,**
- **nečíselně indexovaná pole,**
- **operátory pro práci s poli,**
- **vícerozměrná pole,**
- **řazení polí,**
- **funkce pro práci s poli.**

Co je pole?

V kapitole 1 jste se dozvěděli o skalárních proměnných. Skalární proměnná je pojmenovaná lokace, do níž lze ukládat hodnotu. Podobně pole je pojmenované umístění, do kterého je možné ukládat skupinu hodnot – tj. umožňuje seskupovat proměnné.

V této kapitole bude pole sloužit jako seznam Bobových produktů. Na obrázku 3.1 můžete vidět seznam tří produktů uložených v poli. Tyto produkty jsou uložené v proměnné pojmenované `$produkty`. Za okamžik si dozvíte, jak vytvořit takovou proměnnou.



Obrázek 3.1. Bobovy produkty lze ukládat do pole

Až budete mít data v poli, můžete s ním dělat spoustu věcí. V cyklu je můžete ukládat tak, že provedete stejné akce pro jednotlivé hodnoty pole. Můžete přenášet celou skupinu dat jako jedinou jednotku. Na jediném řádku kódu tak můžete předat všechny hodnoty funkci. Případně můžete seřadit produkty abecedně – když předáte celé pole funkci `sort()`.

Hodnoty v poli nazýváme **prvky** pole. Ke každému prvku se váže **klíč** (jinak řečeno **indexové pořadí**), pomocí něhož přistupujeme k tomuto prvku. Pole v mnoha programovacích jazycích mají číselné indexy počínaje `0` (alternativně `1`).

Jazyk PHP umožňuje používat čísla i textové řetězce jako klíče. Můžete tedy zvolit tradiční číselné indexování nebo textové klíče, které budou smysluplnější a užitečnější. Tento přístup nejspíše znáte pod termíny asociativní pole, mapy, hašovací tabulky nebo slovníky z jiných programovacích jazyků. Skutečný přístup se může lišit podle toho, jestli používáte standardní číselné indexování nebo zajímavější klíče.

Začněme poli indexovanými čísly a potom pokračujme poli s vlastními klíči.

Číselně indexovaná pole

Číselně indexovaná pole jsou k dispozici ve většině programovacích jazyků. V jazyce PHP začínají indexy nulou, ale počáteční hodnotu lze změnit.

Inicializace číselně indexovaného pole

Pole z obrázku 3.1 vytvoříte následujícím řádkem kódu:

```
$produkty = array('Pneumatiky', 'Lahve oleje', 'Zapalovací svíčky');
```

Tak vznikne pole s názvem `$produkty`, které obsahuje tři hodnoty: `'Pneumatiky'`, `'Lahve oleje'` a `'Zapalovací svíčky'`. Zapamatujte si, že `array()` je, podobně jako `echo`, jazykovou konstrukcí spíše než funkcí.

Od verze PHP 5.4 můžete používat zkrácenou syntaxi pro tvorbu polí. Ta preferuje znaky `[` a `]` místo operátoru `array()`. Například pole z obrázku 3.1 byste zapsali zkrácenou syntaxí následovně:

```
$produkty = ['Pneumatiky', 'Lahve oleje', 'Zapalovací svíčky'];
```

Pravděpodobně nebudete muset ručně inicializovat pole (závisí to na jeho obsahu). Pokud máte data v jiném poli, můžete zkopírovat jedno pole do druhého pomocí operátoru `=`.

Kdybyste chtěli uložit do pole vzrůstající posloupnost čísel, mohli byste zavolat funkci `range()`, která za vás takové pole vytvoří. Níže uvedeným příkazem vytvoříte pole `$cisla` s prvky v rozmezí `1` až `10`:

```
$cisla = range(1, 10);
```

Funkce `range()` má volitelný třetí parametr, jenž umožňuje nastavit velikost kroku mezi hodnotami. Kdybyste kupříkladu potřebovali vytvořit pole lichých čísel od `1` do `10`, udělali byste to následovně:

```
$licha = range(1, 10, 2);
```

Funkci `range()` můžete volat také se znaky jako v tomto příkladu:

```
$pismena = range('a', 'z');
```

Pokud máte data uložená v souboru na disku, můžete je načítat do pole rovnou ze souboru. Jak postupovat, se dozvíte později v této kapitole v části „Načítání polí ze souborů".

V případě, že máte data pro pole uložená v databázi, můžete je načíst přímo z ní. O tomto procesu se dozvíte více v kapitole 11.

K dispozici jsou rovněž nejrůznější funkce pro extrahování části pole nebo jeho přeuspořádání. Některé z nich objevíte později v této kapitole v části „Další manipulace s poli".

Přístup k obsahu pole

K obsahu pole přistupujete pomocí jeho názvu. Pokud je touto proměnnou pole, přistupujete k jeho obsahu prostřednictvím názvu a klíče (indexového pořadí). Klíč identifikuje hodnotu v poli, k níž přistupujete. Klíč zapisujete do hranatých závorek za název. Jinými slovy – pomocí výrazů `$produkty[0]`, `$produkty[1]` a `$produkty[2]` přistupujete k jednotlivým prvkům pole `$produkty`.

K prvkům pole můžete přistupovat prostřednictvím znaků `{}` místo znaků `[]`. Například pomocí výrazu `$produkty[0]` byste mohli přistupovat k prvnímu prvku pole s produkty.

Standardně bývá prvek s indexovým pořadím `0` prvním prvkem pole. Stejné číslování používají také jazyky C, C++, Java a další programovací jazyky, ale pokud jste se s ním dosud nesetkali, pravděpodobně si na něj budete chvíli zvykat.

Obsah prvku pole změníte operátorem `=` stejně jako u jiných proměnných. Následujícím příkazem nahradíte první prvek v poli (`'Pneumatiky'`) prvkem `'Pojistky'`:

```
$produkty[0] = 'Pojistky';
```

Na následujícím řádku přidáváte nový prvek `'Pojistky'` na konec tohoto pole, takže bude mít celkem čtyři prvky:

```
$produkty[3] = 'Pojistky';
```

Abyste si zobrazili jeho obsah, můžete napsat tento řádek kódu:

```
echo "$produkty[0] $produkty[1] $produkty[2] $produkty[3]";
```

Zapamatujte si, že ačkoliv interpret jazyka PHP parsuje textové řetězce chytře, můžete se nechat snadno zmást. Pokud narazíte na problém s interpretací pole nebo jiných proměnných v textovém řetězci s uvozovkami, můžete je zapsat mimo uvozovky nebo použít komplexní syntaxi, se kterou se setkáte v kapitole 4. Výše uvedený příkaz `echo` funguje správně, ale ve složitějších příkladech později v této kapitole uvidíte proměnné převážně vně uvozovek.

Pole nemusíte inicializovat ani vytvářet předem (podobně jako jiné proměnné jazyka PHP). Vzniknou automaticky, když je poprvé použijete.

V níže uvedeném bloku kódu vytváříme stejné pole `$produkty`, které jsme dříve vytvořili příkazem `array()`:

```
$produkty[0] = 'Pneumatiky';
$produkty[1] = 'Lahve oleje';
$produkty[2] = 'Zapalovací svíčky';
```

Kdyby pole `$produkty` neexistovalo, prvním řádkem bychom vytvořili nové pole s jedním prvkem. Na dalších řádcích přidáváme hodnoty do tohoto pole. Velikost pole narůstá dynamicky, jak do něj přidáváme prvky. Změna velikosti pole není v řadě jiných programovacích jazyků možná.

Přístup k prvkům pole v cyklu

Pokud indexujete pole sekvencí čísel, můžete ho procházet v cyklu `for` a přitom zobrazovat jeho obsah:

```
for ($i = 0; $i < 3; $i++) {
  echo $produkty[$i]." ";
}
```

V tomto cyklu vypisujeme obsah podobně jako v předchozím příkladu, ale vystačíme si s menším množstvím kódu, jelikož nemusíme ručně psát kód pro přístup ke všem prvkům pole. Schopnost zpracovat prvky pole v cyklu je velmi užitečná. K dispozici je také cyklus `foreach`, který byl navržený speciálně pro pole. V tomto případě bychom ho mohli zapsat takto:

```
foreach ($produkty as $aktualni) {
  echo $aktualni." ";
}
```

V tomto cyklu ukládáme jednotlivé prvky pole do proměnné `$aktualni` a vypisujeme ji na výstup.

Pole s jinými klíči

V poli `$produkty` jsme nechali na interpretu jazyka PHP, aby za nás určil výchozí indexová pořadí. To znamená, že první přidaný prvek získal indexové pořadí `0`, druhý prvek má indexové pořadí `1` atd. Jazyk PHP však umožňuje přiřazovat jednotlivým prvkům libovolné skalární klíče.

Inicializace pole

V následujícím kódu vytváříme pole s názvy produktů jako klíči a jejich cenami jako hodnotami:

```
$prices = array('Pneumatiky' => 2500, 'Lahve oleje' => 250,
              'Zapalovací svíčky' => 100);
```

Symbol mezi klíčem a hodnotou (`=>`) je složený z rovnítka, za nímž následuje znak „větší než“.

Přístup k prvkům pole

K obsahu pole můžete opět přistupovat pomocí názvu proměnné a klíče, takže data uložená v poli `$prices` získáte prostřednictvím výrazů `$prices['Pneumatiky']`, `$prices['Lahve oleje']` a `$prices['Zapalovací svíčky']`.

V níže uvedeném kódu vytváříme stejné pole `$prices`. Tentokrát ho ale nevytváříme rovnou se třemi prvky, ale pouze s jedním prvkem a potom do něho přidáváme další dva prvky:

```
$prices = array('Pneumatiky' => 2500);
$prices['Lahve oleje'] = 250;
$prices['Zapalovací svíčky'] = 100;
```

Následuje další mírně odlišný blok kódu, který je však funkčně ekvivalentní. V této verzi nevytváříme pole explicitně vůbec. To vznikne automaticky, když do něj přidáme první prvek:

```
$prices['Pneumatiky'] = 2500;
$prices['Lahve oleje'] = 250;
$prices['Zapalovací svíčky'] = 100;
```

Používáme cykly

Protože klíče v poli nejsou čísla, nemůžete pracovat s poli pomocí cyklu `for` a jednoduchým počítadlem. Můžete ale použít cyklus `foreach()` nebo konstrukce `list()` a `each()`.

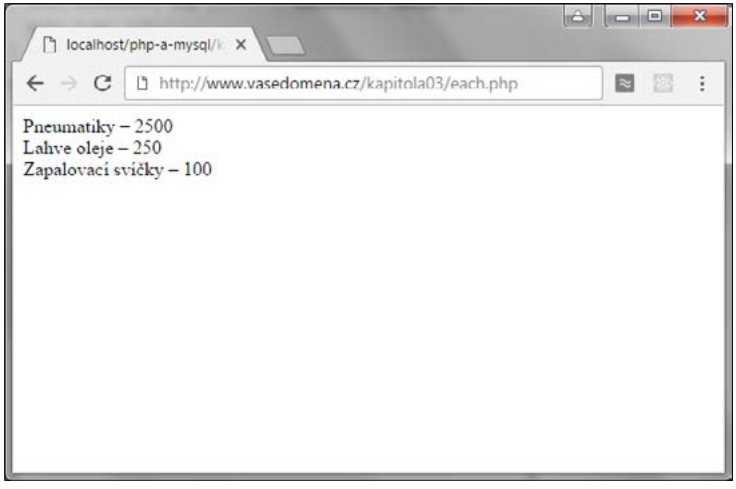
Cyklus `foreach` má trochu jinou strukturu, když neprocházíte číselně indexovaná pole. Můžete jej použít stejně jako v přechozím příkladu, ale navíc můžete začlenit klíče:

```
foreach ($prices as $klic => $hodnota) {
  echo $klic." - ".$hodnota."<br />";
}
```

Takto bychom vypsali obsah pole `$prices` pomocí konstrukce `each()`:

```
while ($element = each($prices)) {
  echo $element['key']." - ".$element['value'];
  echo "<br />";
}
```

Výstup tohoto fragmentu kódu uvidíte na obrázku 3.2.



Obrázek 3.2. Příkazem each() lze procházet pole

V kapitole 1 jsme si popsali cyklus `while` a příkaz `echo`. Ve výše uvedeném kódu jsme použili funkci `each()`, s níž jsme se dosud nesetkali. Ta vrací aktuální prvek v poli a přitom nastavuje následující prvek jako aktuální. Jelikož ji voláme v cyklu `while`, vrátí postupně všechny elementy v poli – provádění cyklu skončí, až dosáhneme konce pole.

V předchozím kódu je proměnná `$element` také pole. Když zavoláte funkci `each()`, vrátí vám pole se čtyřmi hodnotami a čtyřmi klíči. Klíče `key` a `0` obsahují klíč aktuálního prvku a klíče `value` a `1` obsahují hodnotu aktuálního prvku. Třebaže nezáleží na tom, který z nich si vyberete, názvy (které vyhrály v tomto příkladu) jsou popisnější.

Existuje elegantnější řešení, jak vypsat obsah tohoto pole. Konstrukce `list()` umí rozdělovat pole na více hodnot. Můžete tak rozdělit dvojice klíč-hodnota, které získáte funkcí `each()`:

```
while (list($product, $price) = each($prices)) {
  echo $product." - ".$price."<br />";
}
```

V tomto kódu načítáme aktuální prvek pole `$prices` funkcí `each()`, jež nám vrátí pole a nastaví následující prvek jako aktuální. Navíc převádíme prvky s indexovým pořadím `0` a `1` z tohoto vráceného pole na dvě nové proměnné `$product` a `$price`, a to pomocí funkce `each()`.

Pokud používáte funkci `each()`, nezapomínejte na to, že každé pole si udržuje informaci o tom, který prvek je aktuální. Jestliže chcete použít stejné pole dvakrát ve stejném skriptu, musíte vrátit aktuální prvek zpět na začátek pole – pomocí funkce `reset()`. Kdybyste tedy potřebovali znovu vypsat pole `$prices`, napsali byste:

```
reset($prices);
while (list($product, $price) = each($prices)) {
  echo $product." - ".$price."<br />";
}
```

Ve výše uvedeném bloku kódu vracíme aktuální prvek na začátek pole a potom znovu procházíme celé pole.

Operátory pro práci s poli

Jak můžete vidět v tabulce 3.1, většina operátorů pro práci s poli má své analogické alternativy ve skalárních operátorech.

Tabulka 3.1. Operátory pro práci s poli v jazyce PHP

Operátor	Název	Použití	Výsledek
+	Sjednocení	`$a + $b`	Sjednocení polí `$a` a `$b`. Připojuje pole `$b` na konec pole `$a`, ale kolidující klíče vypouští.
==	Rovnost	`$a == $b`	Hodnota `true` v případě, že `$a` a `$b` obsahují stejné prvky.
===	Identita	`$a === $b`	Hodnota `true`, jestliže `$a` a `$b` obsahují stejné prvky, které mají shodné datové typy a jsou stejně uspořádané.
!=	Nerovnost	`$a != $b`	Hodnota `true`, pokud `$a` neobsahuje stejné prvky jako `$b`.
<>	Nerovnost	`$a <> $b`	Ekvivalentní s operátorem `!=`.
!==	Neidentita	`$a !== $b`	Hodnota `true`, jestliže `$a` neobsahuje stejné prvky jako `$b` nebo tyto prvky mají různé datové typy, případně jsou jinak uspořádané.

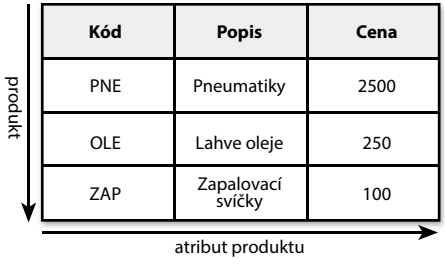
Tyto operátory není nutné příliš vysvětlovat, až na operátor sjednocení. Ten se pokouší vkládat prvky z pole `$b` na konec pole `$a`. Pokud mají prvky v poli `$b` klíče shodné s některými prvky z pole `$a`, tyto prvky nepřidá. Tj. žádný prvek pole `$a` nebude přepsaný.

V tabulce 3.1 jste si zajisté všimli, že k operátorům pro práci s poli existují ekvivalentní operátory pro skalární proměnné. Je logické, že operátor + představuje sčítání skalárních hodnot a sjednocení polí – nemusíte se ani zajímat o to, jak tyto operace fungují na pozadí, aby vám to dávalo smysl. Nemůžete však mezi sebou porovnávat pole se skalárními hodnotami.

Vícerozměrná pole

Pole nemusí být pouze jednoduché seznamy klíčů a hodnot – prvkem pole může být další pole. Tímto způsobem vytvoříte dvourozměrné pole. Dvourozměrné pole si můžete představit jako matici nebo mřížku s výškou a šířkou nebo řádky a sloupci.

Kdybyste chtěli ukládat více informací o Bobových produktech, mohli byste použít dvourozměrné pole. Obrázek 3.3 ukazuje dvourozměrné pole reprezentující Bobovy produkty, v němž každý řádek představuje produkt a každý sloupec ukládá nějaký atribut tohoto produktu.



Obrázek 3.3. Informace o Bobových produktech lze ukládat do dvourozměrného pole

V jazyce PHP byste uložili data do pole z obrázku 3.3 takto:

```
$produkty = array( array('PNE', 'Pneumatiky', 2500),
                   array('OLE', 'Lahve oleje', 250),
                   array('ZAP', 'Zapalovací svíčky', 100) );
```

Z této definice zjistíte, že pole `$produkty` obsahuje tři pole.

Určitě si vybavíte, že abyste mohli přistupovat k datům jednorozměrného pole, musíte použít název pole a indexové pořadí prvku. Dvourozměrné pole je podobné, ale liší se v tom, že každý prvek má dvě indexová pořadí – řádek a sloupec. (Vrchní řádek má indexové pořadí `0` a sloupec nejvíce vlevo má také indexové pořadí `0`.)

Obsah tohoto pole můžete vypsat tak, že postupně načtete všechny jeho prvky:

```
echo '|'.$produkty[0][0].'|'.$produkty[0][1].'|'.$produkty[0][2].'|<br />';
echo '|'.$produkty[1][0].'|'.$produkty[1][1].'|'.$produkty[1][2].'|<br />';
echo '|'.$produkty[2][0].'|'.$produkty[2][1].'|'.$produkty[2][2].'|<br />';
```

Případně byste mohli vložit cyklus `for` do druhého cyklu `for`:

```
for ($radek = 0; $radek < 3; $radek++) {
  for ($sloupec = 0; $sloupec < 3; $sloupec++) {
    echo '|'.$produkty[$radek][$sloupec];
  }
  echo '|<br />';
}
```

Obě verze produkují stejný výstup:

```
|PNE|Pneumatiky|2500|
|OLE|Lahve oleje|250|
|ZAP|Zapalovací svíčky|100|
```

Jediný rozdíl mezi nimi je, že druhý kód bude mnohem kratší u dlouhého pole.

Můžete rovněž upřednostnit názvy sloupců místo čísel jako na obrázku 3.3. Stejnou skupinu produktů je možné uložit s pojmenovanými sloupci následovně:

```
$produkty = array(array('Kód' => 'PNE',
                        'Popis' => 'Pneumatiky',
                        'Cena' => 2500
                        ),
                  array('Kód' => 'OLE',
                        'Popis' => 'Lahve oleje',
                        'Cena' => 250
                        ),
                  array('Kód' => 'ZAP',
                        'Popis' => 'Zapalovací svíčky',
                        'Cena' => 100
                        )
                 );
```

S takovým polem se snadněji pracuje, pokud chcete načíst konkrétní hodnotu. Zapamatovat si, že popis je uložený ve sloupci `Popis`, je jednodušší než si pamatovat, že je uložený ve sloupci s indexovým pořadím `1`. S popisnými klíči si nemusíte pamatovat, že hledaný prvek se nachází na pozici `[x][y]`. Data najdete jednoduše tak, že se odkážete na jejich umístění pomocí řádků a smysluplných názvů sloupců.

Nevýhodou je, že ztratíte možnost procházet skrz jednotlivé sloupce v cyklu `for`. Zde je jeden způsob, jak zobrazit toto pole:

```
for ($radek = 0; $radek < 3; $radek++) {
  echo '|'.$produkty[$radek]['Kód'].'|'.$produkty[$radek]['Popis'].
       '|'.$produkty[$radek]['Cena'].'|<br />';
}
```

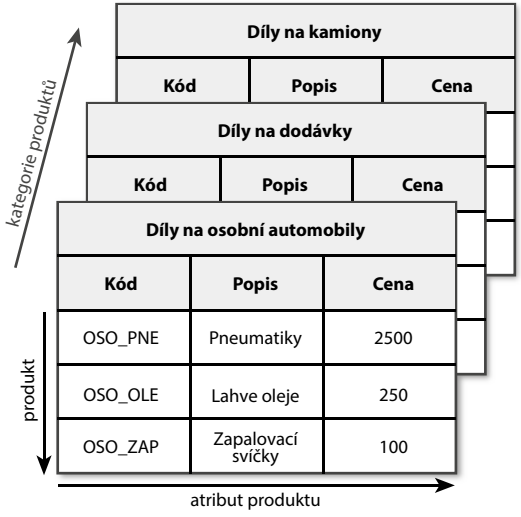
Pomocí cyklu `for` procházíte vnější číselně indexované pole. Řádkem pole `$produkty` je pole s popisnými klíči. Prostřednictvím funkcí `each()` a `list()` v cyklu `while` můžete procházet tato vnitřní pole. Můžete tudíž vepsat cyklus `while` do cyklu `for`:

```
for ($radek = 0; $radek < 3; $radek++) {
  while (list( $klic, $hodnota ) = each( $produkty[$radek])) {
    echo '|'.$hodnota;
  }
  echo '|<br />';
}
```

Nemusíte se zastavit na dvou rozměrech. Podobným způsobem, jakým můžete do prvků pole ukládat pole, můžete do těchto vnitřních polí ukládat další pole atd.

Třírozměrné pole má šířku, výšku a hloubku. Pokud si představujete dvourozměrné pole jako tabulku s řádky a sloupci, představte si třírozměrné pole jako stoh takových tabulek. Na prvek se odkazujete přes řádek, sloupec a vrstvu.

Kdyby Bob rozdělil své produkty do kategorií, mohli byste je uložit do třírozměrného pole. Na obrázku 3.4 můžete vidět Bobovy produkty v třírozměrném poli.



Obrázek 3.4. Toto třírozměrné pole umožňuje rozdělit produkty do kategorií

Z níže uvedeného kódu třírozměrného pole vám bude hned jasnější, že se jedná o pole obsahující pole polí:

```
$kategorie = array(array(array('OSO_PNE', 'Pneumatiky', 2500 ),
                         array('OSO_OLE', 'Lahve oleje', 250 ),
                         array('OSO_ZAP', 'Zapalovací svíčky', 100 )
                        ),
                   array(array('DOD_PNE', 'Pneumatiky', 3000 ),
                         array('DOD_OLE', 'Lahve oleje', 300 ),
                         array('DOD_ZAP', 'Zapalovací svíčky', 125 )
                        ),
                   array(array('KAM_PNE', 'Pneumatiky', 3750 ),
                         array('KAM_OLE', 'Lahve oleje', 375 ),
                         array('KAM_ZAP', 'Zapalovací svíčky', 150 )
                        )
                  );
```

Protože toto pole má jen číselný index, můžete zobrazit jeho obsah v zanořených cyklech `for`:

```
for ($vrstva = 0; $vrstva < 3; $vrstva++) {
  echo 'Vrstva'.$vrstva."<br />";
  for ($radek = 0; $radek < 3; $radek++) {
    for ($sloupec = 0; $sloupec < 3; $sloupec++) {
      echo '|'.$kategorie[$vrstva][$radek][$sloupec];
    }
    echo '|<br />';
  }
}
```

Podobným způsobem můžete vytvářet čtyřrozměrná pole, pětirozměrná pole, nebo dokonce šestirozměrná pole. Nejste nijak omezování počtem rozměrů, ale pro lidi je složité vizualizovat struktury s více než třemi rozměry. Většinu problémů skutečného světa lze logicky převést na strukturu s třemi nebo méně rozměry.

Řazení polí

Řazení dat v poli bývá často užitečné. Můžete vzít jednorozměrné pole a jednoduše ho seřadit.

Funkce sort()

Funkce `sort()` seřazuje prvky pole vzestupně podle abecedy:

```
$products = array('Pneumatiky', 'Lahve oleje', 'Zapalovací svíčky');
sort($products);
```

Prvky tohoto pole budou nyní uspořádané takto: `'Lahve oleje'`, `'Pneumatiky'`, `'Zapalovací svíčky'`.

Můžete řadit i číselné hodnoty. Když budete mít pole cen Bobových produktů, můžete ho seřadit vzestupně podle čísel:

```
$ceny = array(2500, 250, 100);
sort($ceny);
```

Ceny budou uspořádané následovně: `100`, `250`, `2500`.

Zapamatujte si, že funkce `sort()` rozlišuje velikost písmen. Všechna velká písmena řadí před malá písmena. To znamená, že *A* bude před *Z*, ale *Z* bude před a.

Tato funkce má také volitelný druhý parametr. Můžete jí předat jednu z konstant `SORT_REGULAR` (výchozí), `SORT_NUMERIC`, `SORT_STRING`, `SORT_LOCALE_STRING`, `SORT_NATURAL` nebo `SORT_FLAG_CASE`.

Možnost uvádět typ řazení je užitečná zejména tehdy, když řadíte textové řetězce obsahující čísla; například `'2'` a `'12'`. Číselně je 2 menší než 12, ale textový řetězec `'12'` je menší než `'2'`.

Konstanta SORT_LOCALE_STRING umožňuje řadit textové řetězce dle nastavené lokalizace, protože pravidla řazení se na různých místech světa liší.

Konstanta SORT_NATURAL definuje přirozené řazení. Případně můžete také použít funkci natsort(). Přirozené řazení se snaží intuitivně kombinovat řazení textových řetězců a čísel. Kdybyste kupříkladu seřadili prvky 'soubor1', 'soubor2' a 'soubor10' jako textové řetězce, získali byste pořadí 'soubor1', 'soubor10' a 'soubor2'. V případě přirozeného řazení budou uspořádány jako 'soubor1', 'soubor2' a 'soubor10', což je intuitivnější (nebo přirozenější) výsledek pro lidi.

Konstantu SORT_FLAG_CASE můžete kombinovat s konstantou SORT_STRING nebo SORT_NATURAL. Můžete je spojit pomocí logického operátoru AND:

```
sort($produkty, SORT_STRING & SORT_FLAG_CASE);
```

Nyní bude funkce sort() ignorovat velikost písmen, takže bude zacházet s písmenem 'a' stejně jako s písmenem 'A'.

Funkce asort() a ksort()

Pokud ukládáte produkty a ceny do asociativního pole s popisnými klíči, budete muset použít jiné řadicí funkce, aby klíče a hodnoty zůstaly pohromadě.

V níže uvedeném kódu vytváříme pole se třemi produkty a jejich cenami a potom ho řadíme vzestupně podle ceny:

```
$ceny = array('Pneumatiky' => 2500, 'Lahve oleje' => 250,
  'Zapalovací svíčky' => 100);
asort($ceny);
```

Funkcí asort() řadíme pole vzestupně podle hodnot. V našem poli jsou hodnotami ceny a klíči jsou textové popisy. Kdybychom chtěli seřadit toto pole podle popisů, zavolali bychom funkci ksort(), která řadí pole podle klíčů.

V níže uvedeném kódu řadíme klíče v poli abecedně – Lahve oleje, Pneumatiky, Zapalovací svíčky:

```
$ceny = array('Pneumatiky' => 2500, 'Lahve oleje' => 250,
  'Zapalovací svíčky' => 100);
ksort($ceny);
```

Řadíme obráceně

Funkce sort(), asort() a ksort() řadí pole vzestupně. Ke každé z nich existuje funkce, jež řadí pole obráceně – sestupně. Obrácené verze se nazývají rsort(), arsort() a krsort().

Tyto funkce lze používat stejným způsobem jako funkce pro vzestupné řazení. Funkce rsort() řadí jednorozměrné číselně indexované pole sestupně. Funkce arsort() řadí

jednorozměrné pole sestupně podle hodnot. Funkce `krsort()` řadí jednorozměrné pole sestupně podle klíčů.

Řazení vícerozměrných polí

Řazení polí s více rozměry nebo jinak než abecedně/číselně už je složitější. Jazyk PHP ví, jak srovnat dvě čísla nebo dva textové řetězce, ale ve vícerozměrném poli je prvkem další pole.

Máte na výběr ze dvou přístupů k řazení vícerozměrných polí: vytvořit si uživatelsky definované řazení nebo funkcí `array_multisort()`.

Funkce array_multisort()

Funkci `array_multisort()` můžete používat k řazení vícerozměrných polí nebo k řazení více polí současně.

Následuje dříve použitá definice dvourozměrného pole, do nějž jsme uložili tři produkty s kódem, popisem a cenou:

```
$produkty = array(array('PNE', 'Pneumatiky', 2500),
                  array('OLE', 'Lahve oleje', 250),
                  array('ZAP', 'Zapalovací svíčky', 100));
```

Kdybychom vzali funkci `array_multisort()` a aplikovali bychom ji následovně, seřadila by toto pole. Ale v jakém pořadí?

```
array_multisort($produkty);
```

Tak seřadíme pole `$produkty` vzestupně podle prvního prvku každého pole:

```
'OLE', 'Lahve oleje', 250
'PNE', 'Pneumatiky', 2500
'ZAP', 'Zapalovací svíčky', 100
```

Tato funkce má následující prototyp:

```
bool array_multisort(array &a [, mixed poradi = SORT_ASC [, mixed typ = SORT_
REGULAR [, mixed $... ]]])
```

Jako argument *poradi* můžete předat konstantu `SORT_ASC` pro vzestupné řazení nebo `SORT_DESC` pro sestupné řazení.

Funkce `array_multisort()` podporuje stejnou skupinu konstant u argumentu *typ* jako funkce `sort()`.

Důležitou vlastností funkce `array_multisort()` je, že zachovává vazby klíč-hodnota, pokud jsou klíči textové řetězce, ale ne v případě, že se jedná o čísla.

Uživatelsky definované řazení

Když vezmeme v úvahu pole z předchozího příkladu, najdeme přinejmenším dvě smysluplná uspořádání. Buď můžeme řadit produkty abecedně podle popisů, nebo je můžeme řadit číselně podle cen. Alternativně můžeme pomocí funkce `usort()` popsat, jak bychom chtěli porovnávat prvky tohoto pole. K tomu však musíme napsat naši vlastní porovnávací funkci.

Níže uvedeným kódem seřadíme toto pole abecedně podle druhého sloupce – popisu:

```
function porovnej($x, $y) {
  if ($x[1] == $y[1]) {
    return 0;
  } else if ($x[1] < $y[1]) {
    return -1;
  } else {
    return 1;
  }
}
usort($produkty, 'porovnej');
```

V této knize jsme volali spoustu vestavěných funkcí jazyka PHP. Abychom seřadili toto pole, musíme definovat vlastní funkci. Jak psát funkce, si budeme podrobně vysvětlovat v kapitole 5, ale zde si uvedeme stručný úvod.

Funkci definujete klíčovým slovem `function`. Za něj musíte uvést název funkce. Měli byste volit smysluplné názvy funkcí, jako například `porovnej()` v tomto příkladu (případně anglicky `compare()`). Spousta funkcí má nějaké parametry. Funkce `porovnej()` má dva – jeden pojmenovaný `$x` a druhý `$y`. Tato funkce tudíž přijímá dvě hodnoty, které mezi sebou porovnává za účelem jejich seřazení.

V tomto případě jsou `$x` a `$y` dvě pole z hlavního pole, přičemž každé z nich reprezentuje nějaký produkt. K popisu tohoto pole přistupujete zápisem `$x[1]`, protože popis je jeho druhý prvek a číslování začíná nulou. Pomocí výrazů `$x[1]` a `$y[1]` porovnáváte popisy z polí předaných této funkci.

Až provádění funkce skončí, vrátí odpověď kódu, jenž ji zavolal. Tento proces se nazývá **vrácení hodnoty**. K vrácení hodnoty slouží klíčové slovo `return` zapsané uvnitř funkce. Například příkaz `return 1;` odesílá hodnotu `1` zpět kódu, který tuto funkci zavolal.

Abyste mohli použít funkci `porovnej()` ve funkci `usort()`, musíte mezi sebou porovnávat argumenty `$x` a `$y`. Funkce `porovnej()` musí vracet hodnotu `0`, jestliže se `$x` a `$y` rovnají, dále zápornou hodnotu, pokud je `$x` menší, a kladnou hodnotu v případě, že je `$y` větší. Vracíte z ní tudíž `0`, `1` nebo `-1` v závislosti na hodnotách `$x` a `$y`.

Na posledním řádku výše uvedeného kódu voláte vestavěnou funkci `usort()`, které předáváte řazené pole (`$produkty`) a název porovnávací funkce (`porovnej()`).

Kdybyste chtěli uspořádat toto pole jinak, mohli byste si napsat jinou porovnávací funkci. Například pro seřazení podle ceny byste v této funkci porovnávali třetí sloupec:

```
function porovnej($x, $y) {
  if ($x[2] == $y[2]) {
    return 0;
  } else if ($x[2] < $y[2]) {
    return -1;
  } else {
    return 1;
  }
}
```

Po zavolání `usort($produkty, 'porovnej')` by bylo toto pole seřazené vzestupně podle ceny.

POZNÁMKA

Kdybyste spustili výše uvedené bloky kódu, abyste je otestovali, neviděli byste žádný výstup. Tyto fragmenty kódu můžou být součástí většího kusu kódu, například ve vašem projektu.

Písmeno *u* v názvu funkce `usort()` označuje uživatele, protože se neobejde bez uživatelsky definované porovnávací funkce. Funkce `uasort()` a `uksort()`, což jsou alternativní verze funkcí `asort()` a `ksort()`, také vyžadují uživatelsky definované porovnávací funkce.

S funkcí `uasort()` byste měli řadit nečíselně indexovaná pole podle hodnot – tj. podobně jako s funkcí `asort()`. Funkci `asort()` zvolte, pokud hodnotami jsou čísla nebo textové řetězce. Funkci `uasort()` společně s uživatelsky definovanou funkcí použijte pro řazení složitějších objektů – kupříkladu polí.

S funkcí `uksort()` můžete řadit nečíselně indexovaná pole podle klíčů – tedy podobně jako s funkcí `ksort()`. Funkci `ksort()` si vyberte, jestliže hodnotami jsou čísla nebo textové řetězce. Funkci `uksort()` společně s uživatelsky definovanou funkcí použijte pro řazení složitějších objektů – například polí.

Obrácené uživatelské řazení

K funkcím `sort()`, `asort()` a `ksort()` existují příbuzné funkce pro řazení v opačném směru, které nesou ve svých názvech písmeno *r*. Funkce uživatelsky definované takové alternativní varianty nemají, ale i tak můžete uspořádat vícerozměrné pole obráceně. Protože definujete porovnávací funkci, můžete ji napsat tak, aby vracela opačné hodnoty. Kdybyste chtěli seřadit pole obráceně, museli byste vrátit hodnotu `1` pro `$x` menší než `$y` a hodnotu `-1` pro `$x` větší než `$y`.

```
function porovnej_obracene($x, $y) {
  if ($x[2] == $y[2]) {
    return 0;
  } else if ($x[2] < $y[2]) {
    return 1;
```

```
    } else {
      return -1;
    }
  }
```

Kdybychom zavolali `usort($produkty, 'porovnej_obracene')`, seřadili bychom pole `$produkty` sestupně podle ceny.

Přeuspořádání polí

V některých aplikacích pravděpodobně můžete chtít přeuspořádat pole jinými způsoby než řazením. Funkce `shuffle()` uspořádá prvky v poli náhodně. Funkce `array_reverse()` vám vrátí kopii pole se všemi prvky v opačném pořadí.

Funkce shuffle()

Bob chce prezentovat malé množství svých produktů na úvodní stránce svých webových stránek. Má v nabídce velké množství produktů, ale na úvodní stránce by rád prezentoval tři náhodně vybrané produkty. Aby se návštěvníci nenudili, chtěl by při každé návštěvě zákazníka vybrat tři jiné produkty. Můžeme snadno splnit jeho požadavky, pokud uložíme produkty do pole. Výpis 3.1 zobrazuje tři náhodně vybrané obrázky tak, že náhodně přeuspořádá (zamíchá) pole a potom zobrazí první tři prvky.

Výpis 3.1. bobs_front_page.php – V jazyce PHP vytváříme dynamickou úvodní stránku pro Bobovy autodíly

```
<?php
  $pictures = array('brakes.png', 'headlight.png',
                    'spark_plug.png', 'steering_wheel.png',
                    'tire.png', 'wiper_blade.png');

  shuffle($pictures);
?>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Bobovy autodíly – Výsledek objednávky</title>
  </head>
  <body>
    <h1>Bobovy autodíly</h1>
    <div align="center">
    <table style="width: 100%; border: 0">
      <tr>
      <?php
        for ($i = 0; $i < 3; $i++) {
          echo "<td style=\"width: 33%; text-align: center\">
                <img src=\"";
          echo $pictures[$i];
```