

PHP - Prv® kroky

Obsah

- [1. Úvod](#)
- [2. Klient a Server](#)
- [3. Inštalácia](#)
- [4. Prv® kroky](#)
- [5. Editor a prehliadač](#)
- [6. Databáza](#)
- [7. Základná syntax](#)
- [8. Typy](#)
- [9. Premenné](#)
- [10. HTML Formuláre](#)
- [11. Kontanty](#)
- [12. Výrazy](#)
- [13. Operátory](#)
- [14. Riadiace príkazy](#)
- [15. Funkcie](#)
- [16. Triedy a objekty](#)
- [17. PHP a Grafika](#)
- [18. PHP a Databázy](#)
- [19. PHP klub](#)

Autor: [Milan Gacik](#)

Reviewer:

Version:

1.1 initial 2002-10-08 + doplnenie IIS

1.0 draft 2002-07-30

Zdroj: <http://www.php.sk>

Diskusia: <http://www.php.sk/diskus>

Download: <http://www.php.sk/download.php>

Dokument je chránený autorským zákonom. Dokument alebo jeho časť je možné použiť len za účelom vlastného vzdelávania, resp. za účelom výuky, pričom musí byť explicitne uvedený jeho autor a odkaz na zdroj (www.php.sk). Bez podozrenia povolenia nesmie byť dokument pozmeňovaný ani zverejňovaný resp. jeho časť vydávaná pod vlastným copyrightom.

 [Stiahnuť celý dokument vo verzii PDF \(Zip 312kb\)](#)

1. Úvod

Ahojte,

vitajte v rodine PHP programátorov. Tento dokument vznikol ako výsledok otázok a odpovedí na problémy, s ktorými sa najčastejšie stretávajú najmä užívatelia a programátori v skriptovacom jazyku PHP. Nenájdete tu všetky otázky a informácie, ale je vhodným začiatkom a úvodom do problematiky, ktorú PHP prináša. Pri čítaní Vám prajem veľa trpezlivosti a zdaru.

Ak nájdete v tomto dokumente nezrovnalosti alebo máte niečo doplniť, neváhajte a pošlite mail na [moju adresu](#).

Neč sa hlbšie ponoríme do tajov jazyka PHP, bolo by vhodné najprv sa stručne zoznámiť s jeho históriou, ak sa to tak vôbec dá povedať. PHP sa zrodilo niekedy v roku 1994 a vznikol ako program v Perl, ktorý slúžil na evidenciu prístupov k WWW stránkam pán Rasmusa Lerdorfa, ktorý bol jeho prvým autorom. Nakoľko však Perl ako interpretovaný jazyk priveľmi zaťažoval WWW server, autor svoj software prepísal do jazyka C. Postupne ho používalo čoraz viac ľudí, stával sa viac a viac obľúbenejším a autor bol doslova donútený vydať k svojmu dielku dokumentáciu. Systém bol neskôr rozšírený o požiadavky jeho používateľov a bol uvoľnený pod názvom **Personal Home Page Tools**. Po zabudovaní SQL modulu, ktorý umožňoval PHP spolupracovať s databázami a vytvárať formuláre sa názov zmenil na **Personal Home Page Form Interpreter PHP/FI**. Tento systém vznikol ako kompilát prvotného PHP a programu, ktorý spracovával výsledky databázových dopytov na Webe (Form Interpreter). Tak vznikol PHP/FI 2.0. V roku 2000 bola vydaná posledná verzia rady PHP 3.x. Oproti svojim predchodcom sa celý systém zriechlil a rozšíril a kopec nových funkcií. Dnes najčastejšie používaná verzia PHP je verzia 4.x, ktorej základ tvorí jadro s názvom [Zend](#).

Príklad 1-1. Prvý príklad

```

1
2 <html>
3 <head>
4 <title>Príklad</title>
5 </head>
6 <body>
7 <?php echo "Hi, Som PHP script!"; ?>
8 </body>
9 </html>
10

```

PHP sa týmto líši od CGI skriptov napísaných v Perle alebo C jazyku -- namiesto písania množstva kódu pre výpis HTML stačí napísať HTML kód, medzi ktorými je zakomponovaný príkaz PHP (ktorý v tomto prípade vypíše text). PHP kód sa zaľna a končí špeciálnym štartovacím a ukončovacím tagom **<?php** a **?>** ktorými umožní skočiť do a z PHP módu. Aj keď je to stále smiešne - ale toto je jazyka umožní zaľiať nímkom písaných rýchlo výkonných PHP/HTML stránky - ale pri správe kódu je to veľmi nešťastné riešenie...

Čo odlišuje PHP od Javascriptu je, že tento jazyk je vykonávaný na strane servera. Ak sa spustí PHP skript na serveri, klientský prehliadač by dostal len výsledok behu skriptu, konkrétne kompletnú stranu v jazyku HTML. Jazyku HTML rozumie skoro každý prehliadač, čo sa o klient-side skriptoch akým je napríklad JAVASCRIPT povedať nedá.

Verzia jadra PHP, ktorú je vyvíjaná pod kódom názvom ZEND je zložením tohto populárneho skriptového jazyka už od verzie PHP 4.0. Dnes je Zend vo verzii 2.0 momentálne už treťou verziou jadra PHP. Predchádzajúca verzia ZENDu (verzia 1.0) bola postavená na rovnakých parsovacích pravidlách ako skriptovacie jadro PHP 3.x, ktorému dnes zvykneme hovoriť ZEND 0.5. To stále umožnilo veľmi ľahké migrovanie (prechod) z verzie PHP 3.x na verziu 4.0, ale zároveň zmemožňuje ďalší vývoj jazyka.

2. Klient a Server

PHP samotná funguje ako súčasť URLítého systému, ktorým tvorí webový server, PHP jadro a prehliadač. Doporučujem zároveň nainštalovať aj databázový systém.

- **Internetový server HTTP** je program, ktorým sa má spustiť na úbovoľnom počítači pripojenom na Internet. Pretože jeden počítač má plniť úlohu viacerých serverov, sú v rámci komunikačného protokolu zavedené tzv. porty. Porty tvoria prístupové body, služby. Web server beží zvyčajne na porte 80. K tomuto portu taktiež smerujú požiadavky klientov. Aby server mohol súčasne obsluhovať viacerých klientov, slúži port 80 iba pre nadviazanie spojenia a prenos dát prebieha na niektorom z prvej voľných portov. V architektúre klient server tvorí serverovú stranu
- **Databázový systém** je program ktorým slúži pre prácu s dátami. Spolupracuje pomocou natívnych driverov alebo ODBC či JDBC rozhrania s webovým serverom.
- **Prehliadač** alebo **browser** je program, ktorým slúži na zobrazenie dát na strane klienta.

Architektúra klient/server je založená na nasledujúcej myšlienke: dáta by sa mali spracovávať predovšetkým tam, kde sú k dispozícii. To znamená, že pokiaľ nie sú uložené na ústrednej stanici, nemali by sa tam spracovávať. Pokiaľ ich ale ústredná chce spracovávať zo svojej stanice, mala by nastaviť deba prácu - medzi miestom uloženia dát a ústrednou pracovnou stanicou.

Víhody spomáanej architektúry sú predovšetkým zamedzenie redundancii dát (dátá sú uložené centrálne) pri zachovaní možnosti viac užívateľskému prístupu za existencie viac užívateľskému spracovávania dát. Centrálnym lastiam, ktoré obsahujú dátá a poskytujú ich na spracovanie užívateľom hovoríme **server**. Naopak užívateľskými stanicami, ktoré vyžadujú služby serveru hovoríme **klient**. Týmto dostaneme názov celej architektúry : **klient** / **server**. Klient a server pracujú na sebe úplne nezávisle.

V dobe rozmachu budovania informačných systémov založených na využití osobných počítačov, bolo možné zachytiť tendencie mohutného prechodu k technológii klient / server. Najľastejšie sú uvádzané tieto dôvody :

- Zvýšenie flexibility a rýchlosti vývoja aplikácií

Toto je dané skutočnosťou, že úloha zodpovednosti za spracovanie je prenesená na server. Jedná sa predovšetkým o prístup k dátam, zaistenie ich integrity a dodržiavanie pravidiel pre prácu s nimi. Prvé tým, že klient môže využívať tieto služby servera, nemusia byť implementované na klientskej strane a tým je ich kód jednoduchý a ľahšie vytvorený a modifikovateľný.

- Zvýšenie bezpečnosti systému

Bezpečnosť systému je možné dosiahnuť tým, že prostriedky prístupu k dátam, ako i zaistenia ich integrity a konzistencie sú implementované na strane servera, a teda nemá byť dôležitý k ich obmedzeniu na strane klienta.

- Intuitívnejší vývoj a prevádzka

Použitie technológií klient/server bolo od začiatku spojené s použitím intuitívneho grafického užívateľského prostredia (GUI), ktoré je umiestnené na klientskej strane.

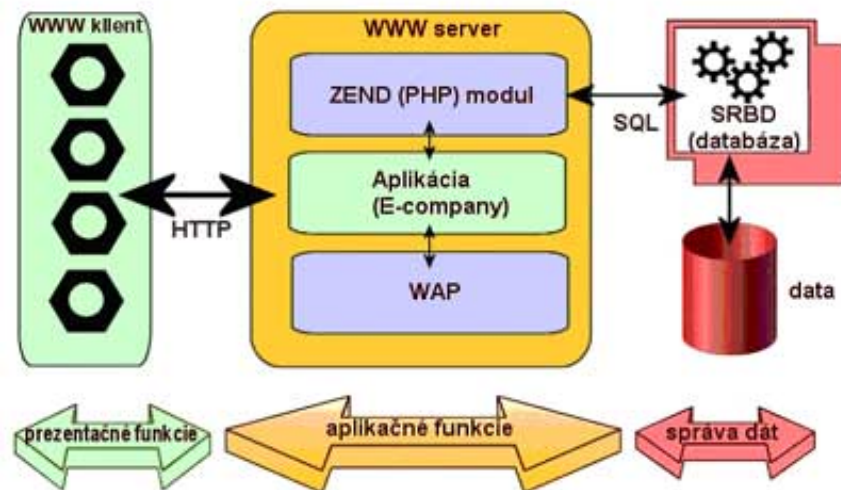
- Lepšia integrácia systému

Proces klienta môže naraz využívať informácie z niekoľkých serverov, bez toho aby sa musel prepájať medzi niekoľkými systémami.

- Podpora decentralizovaných organizačných štruktúr

Na rozdiel od architektúry host-terminal poskytuje filozofia klient/server omnoho lepšie možnosti na prechod od centralizovaného uloženia dát k spracovaniu a uloženiu dát tam, kde je to najvhodnejšie. Fyzické uloženie dát je pritom pre užívateľa úplne transparentné.

Na obrázku 1 vidíme typický príklad internetovej aplikácie typu klient/server.



Obr.1 - klient/server

Spôsob práce s objektami bol v novej verzii jadra kompletne prepracovaný. Zmeny prinášajú nielen nové funkcie, ale aj zvýšenie výkonu.

Pred nájzom ľtánm tohto dokumentu odporúam nainštalovať PHP na Váš počítač, preto mŤ Nájia kapitola nŤzov...

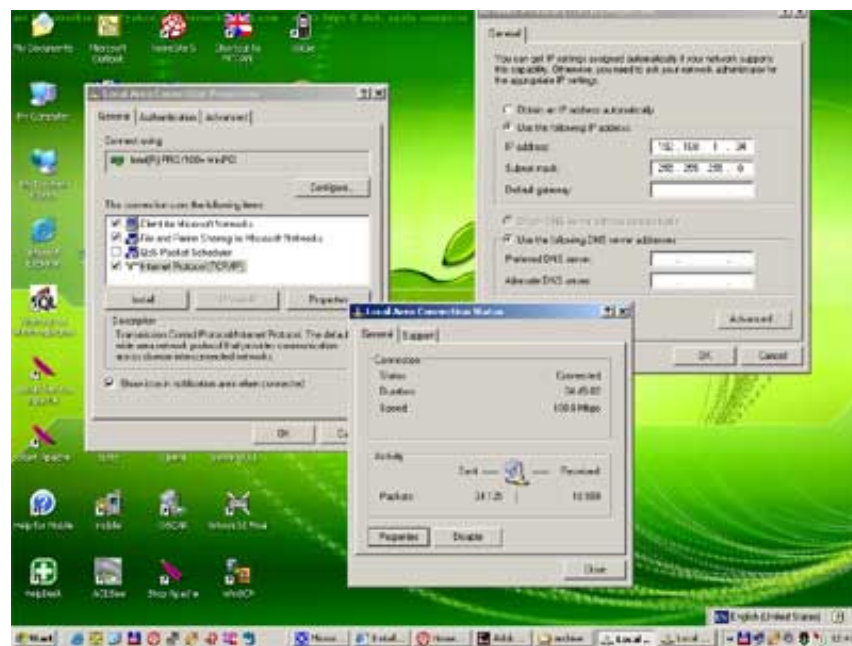
3. Inštalácia

Predpokladajme, Ťe PHP budete pouŤvať pre vŤvoj vlastných internetových aplikácií. Preto je nutné nainštalovať nielen samotný PHP, ale aj Web server, databázový server aj prehliadač. Zaľneme inštaláciou pod operaľným systŤm Microsoft Windows.

Microsoft Windows a Apache (PHP HOME)

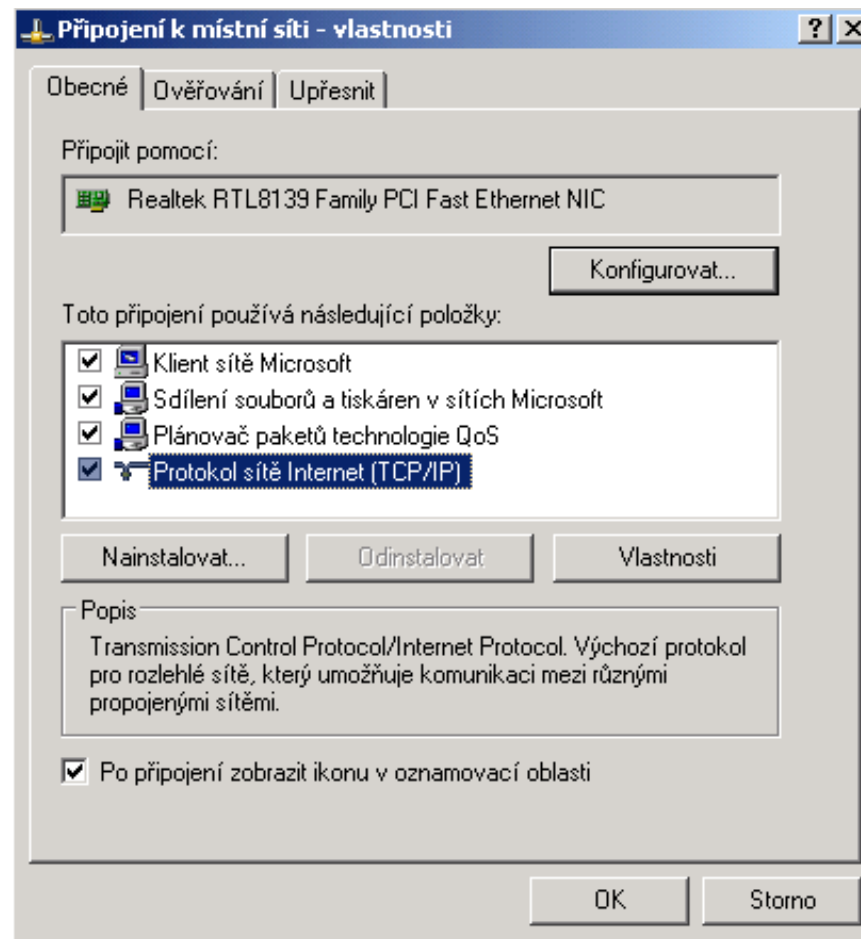
Ak pouŤvate systŤm MS Windows 95, aktualizujte si sieťovú vrstvu Winsock na verziu 2. Aktualizaľný sŤbor nŤjdete na stránkach [Microsoftu](#).

Pre správne fungovanie Web servera je nutné mať sieťovú podporu. Overt si, ľi Váš počítač má IP adresu, resp. disponuje sieťovým zariadením. Staľ sieťová karta alebo modem. Pre fungovanie domŤceho WWW servera nemusŤte byť pripojený na internet! **MusŤte si vŤak nainštalovať podporu pre TCP/IP protokol** (pozri obr. 2 a 2a)



Obr. 2 - Sieťová vrstva a protokol TCP/IP

Pre správu funkciu Web servera musíte mať nainštalovaný TCP/IP protokol.



Obr. 2a

Jednoduché overenie funkčnosti TCP/IP protokolu: V okne shellu (command.com, cmd.exe) napíšte **ping 127.0.0.1** a odpoveďou by mal byť správa reply from.... ako na obr. 3

```
C:\WINDOWS\System32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

d:\>ping 127.0.0.1

Pinging 127.0.0.1 with 32 bytes of data:

Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

d:\>
```

Obr.3 - ping

Pre inštaláciu PHP, webov@ho servera Apache a databázov@ho syst@mu MYSQL sme vytvorili balík PHP HOME. Jeho spustením na svojej PC nainštalujete zloženú komponenty pre vývoj aplikácií v PHP. Program automaticky skonfiguruje všetky potrebné nástroje. Program je možné stiahnuť si z www.php.sk alebo sa nachádza na CD PHP klubu (<http://www.php.sk/registracia.php>)



Obr. 4 - konfigurácia dial·g PHP Home 2

Po úspešnej inštalácii klikneme na ikonku Test My PHP alebo napíšeme do okna prehliadača URL: <http://127.0.0.1> a mala by sa zobraziť naša prvá testovacia PHP stránka. Web server musí samozrejme bežať, čo je pri Apache signalizované malou zelenou gúpkou v ikonke apache monitor v systáre.

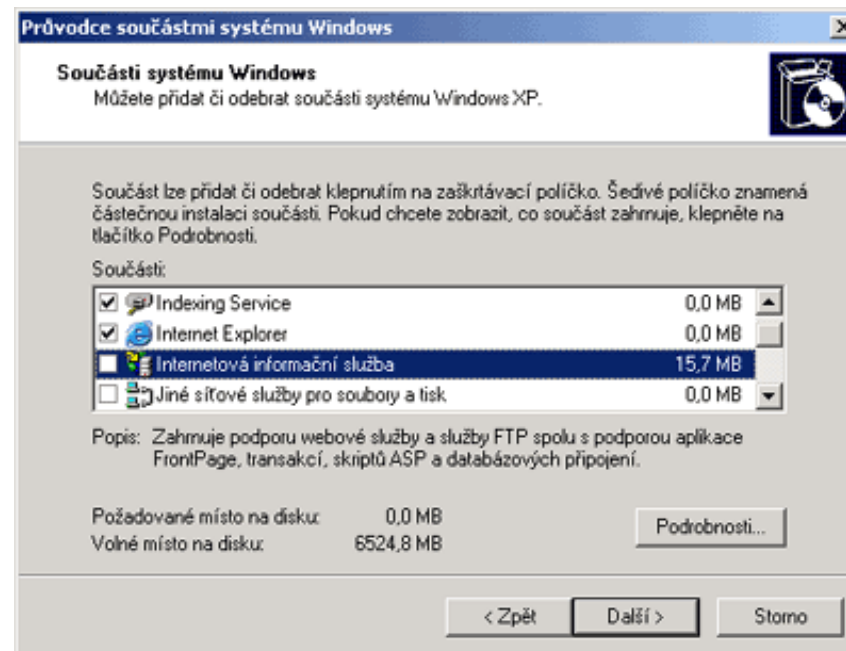
Poznámka: IP adresa 127.0.0.1 a localhost s¼ aliasy pre lok¼lnu IP adresu.

Okrem web servera Apache je mo¼n® na¼ntalovaš PHP ako modul web servera IIS od Microsoftu. V Ňal¼gom texte si uk¼š¼eme ako na¼ntalovaš IIS do Windows XP /2000/.

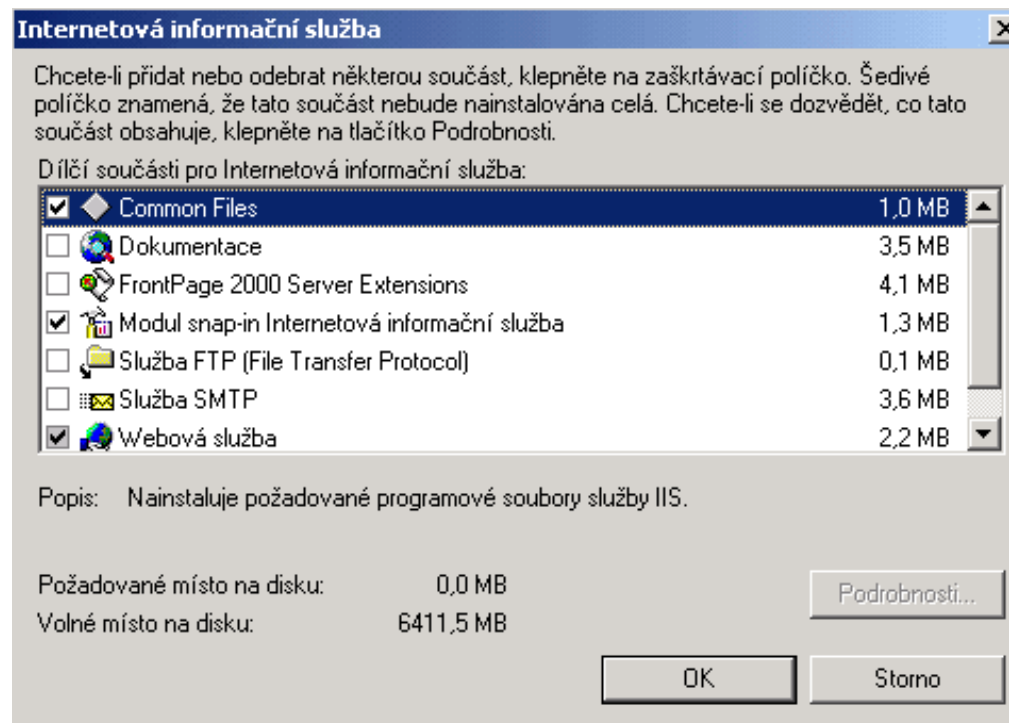
Upozornenie! IIS obsahuje mnoho ch¼b, in¼talašciu a jeho prev¼dzkovanie neodpor¼¼ame a nesupportujeme. Pre ak¼ko¼vek inform¼cie oh¼ladom IIS sa obr¼š¼te na Microsoft Corporation.

Microsoft Windows a Apache (PHP HOME)

Na¼ntalujeme IIS pomocou in¼tala¼n¼ho CD Windows. Zvol¼me ponuku START->CONTROL PANEL->ADD/REMOVE programs a zvol¼me z¼š¼lo¼ku SYSTEMS PROGRAMS a vyberieme slu¼bu IIS (obr. 4a). Z ponuky vyberieme menu Podrobnosti a vyberieme len polog¼ky Common files, snap-in Internetov¼ slu¼ba a Webov¼ slu¼ba (obr. 4b). IIS na¼ntalujeme a re¼tartujeme PC.



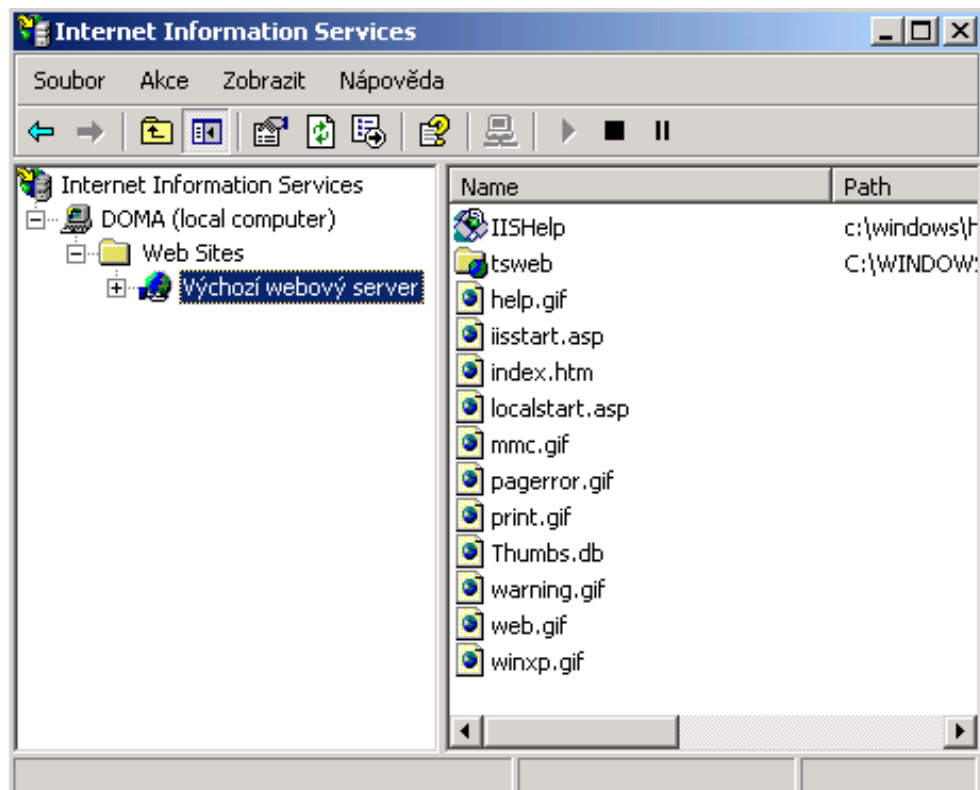
Obr 4a



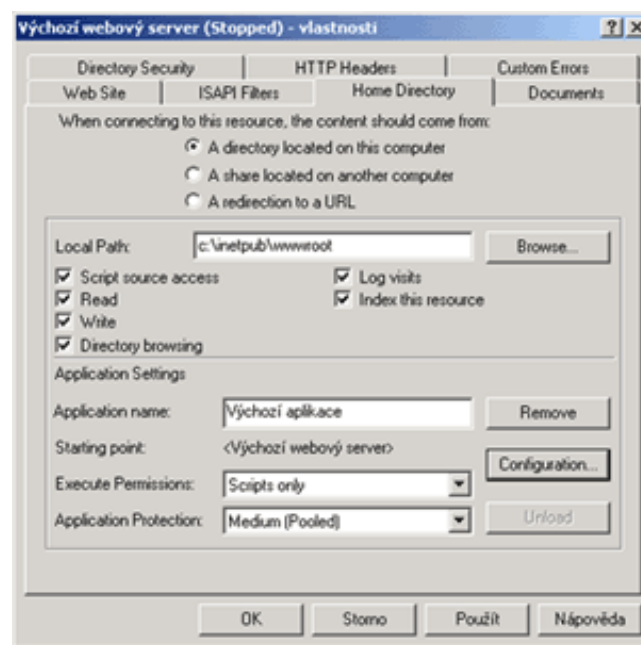
Obr 4b

Získáme soubor php-4.x.x-Win32.zip a rozpakujeme ho do adresáře napr. <c:\php>. Do adresáře <c:\windows\system32> skopírujeme knihovny php4ts.dll a php4ts.lib.

Potom spustíme ovládací konzolu IIS s názvem integrita a pro webový server nastavíme příslušné vlastnosti v menu Vlastnosti (klik pravým tlačítkem myši - obr 4c a 4d)



Obr 4c



Vlastnosti nastavíme nasledovne:

1. HOME DIRECTORY-> CONFIGURATION->ADD -> Executable: **C:\php\sapi\php4isapi.dll** Extension **.php**
2. ISAPI Filters -> ADD -> Filter name: **PHP** Executable: [C:\php\sapi\php4isapi.dll](#)
3. Nastavíme Directory security podľa manuálu k IIS, resp. necháme default.
4. Documents -> ADD **index.php**
4. Do adresára [c:\intepub\wwwroot](#) uložíme súbor index.php s obsahom

```
<?php
    phpinfo();
?>
```

a spustíme IIS (štart). Prípadné chyby riešime s pomocou logu alebo nainštalujeme Apache :).

Druhou možnosťou je nainštalovať PHP do IIS pomocou CGI inštalátora. Súbor sa volá php-4.x.x-installer.exe

OS Unix/Linux

Operačný systém pre pravých mužov a ženy umožňuje prevádzkovať služby a moduly, ktoré pod OS Windows ani neexistujú. Proces inštalácie a konfigurácie nie je triviálny, radšej však buď kompletne použijete RPM, DEB balíčky alebo všetko ručne kompilujete. Druhá možnosť je vhodná pre pravých expertov a "internet guru". Aj ja používam len ručnú kompiláciu Apache, PHP aj MySQL.

Inštalácia v skratke (ako modul Apache)

1. gunzip apache_2.0.x.tar.gz
2. tar xvf apache_2.0.x.tar
3. gunzip php-4.0.x.tar.gz
4. tar xvf php-4.0.x.tar
5. cd apache_2.0.x
6. ./configure --prefix=/usr/local/apache
7. cd ../php-4.2.x
8. ./configure --sysconfdir=/etc --with-mysql --with-gd --with-apache=../apache_2.0.x --enable-track-vars
9. make
10. make install
11. cd ../apache_2.0.x
12. ./configure --prefix=/usr/local/apache --activate-module=src/modules/php4/libphp4.a
13. make
14. make install

Pred týmto krokom musíte prekopírovať httpd binárny súbor cez už existujúcu verziu. Nezabudnite však predtým vypnúť WWW server.

15. cd /etc

16. cp php.ini-dist php.ini

Teraz editujte /etc/php.ini s4bor a nastavte PHP vo4by. Ak chcete maš tento s4bor inde, pou4ite --sysconfdir=/cesta v kroku 8.

17. Editujte httpd.conf a pridajte

AddType application/x-httpd-php .php .phtml .php3 .php4

Tu m4gete nastaviš pr2pony, ktor4 bude PHP spracovšvaš.

18. Pou4ite normšlny sp1sob zastavenia a spustenia WWW servera Apache nielen HUP alebo USR1 signšl.

cd /usr/local/apache/bin

apachectl stop

apachectl start

Pr2klad inštalscie pod OS LINUX Debian

Predpokladšm, že mšm server pripojen4 k internetu. Aktualizujem distrib4ciu pr2kazom

```
# apt-get update
# apt-get upgrade
```

a postupne kompilujem jednotliv4 moduly. Nezabudnem cez apt-get stiahnuš potrebn4 kni4nice, ak ch4baj4 hlavi4kov4 s4bory (*.h) tak aj ich dev verzie. Pr2klad je libgd2 a libgd2-dev

```
# apt-get install libgd2
```

Nšzov kni4nice sa dš zistiš napr2klad na www.debian.org v sekcii packages, pri4lom vyh4dšvam v4etky (ak testing) bal4ky.

1. MYSQL

```
# cd /tmp/mysql
# ./configure --prefix=/usr/local/mysql --sysconfdir=/etc --with-charset=win1250
# ./make
# ./make install
# cd ..
```

2. OPENSSL - SSL zaklad

```
# cd openssl
# ./config
# ./make
# ./make install
```

```
# cd ..
```

MODSSL

```
cd modssl
SSL_BASE=/usr/local/ssl
export SSL_BASE
./configure --with-apache=../apache_1.3.26 --with-ssl=../openssl-0.9.6d/ --prefix=/usr/local/apache
```

```
PHP 4.2.x
cd php-4.xx
```

```
./configure --with-apache=../apache_1.3.26 --with-mysql=/usr/local/mysql --with-gd --with-ttf --sysconfdir=/etc --disable-debug --enable-track-vars
--with-pdflib=/usr/local/pdflib --with-jpeg-dir --with-zlib-dir --enable-gd-native-ttf --with-tiff-dir --enable-ftp --enable-sockets
make
make install
```

APACHE 1.3.26

```
cd apache
./configure --prefix=/usr/local/apache --activate-module=src/modules/php4/libphp4.a --enable-module=ssl --enable-shared=ssl
make
make install
```

a spustíme Apache

/etc/init.d/apache start alebo **/usr/local/apache/bin/apactrl start**

Konfigurácia PHP

Šť tu dve možnosti.

Použitie "setup" skriptu, ktorú je priložený k PHP. Skript sa pýta na odpovede na otázky a vygeneruje MakeFile. Skript spustíte príkazom ./setup.

Skript tiež vygeneruje súbor "do-conf", ktorú obsahuje voľby predané configure. Môžete tento súbor meniť bez spustenia setupu, stačí napísať ./do-conf a spustíte configure s novými voľbami.

Pre spuštění konfigurácie **configure**. Všetky možné voľby vypíšete **./configure --help**.

Podrobnejšie je možné nastaviť PHP v konfiguračných súboroch php.ini a httpd.conf.

4. Prvé kroky.

Pri inštalácii PHP Home sme nainštalovali aj databázu MySQL. Pre jej správu slúži administrátorský nástroj [phpMyAdmin](#) naprogramovaný v PHP.

- Overíme si, či bežia daemony pre webový server apache a databázu mysql.

Windows: v systemtray musia byť zelená ikonka zo semaforom a zelenou gúpkou.
Unix: pomocou príkazu **ps aux**

Spustíme v prehliadači vodoň testovaciu stránku. PHP parsuje (spracováva) len stránky, ktoré prešli cez rozhranie WEB servera, takže PHP skript inicializujeme (spustíme) vždy len pomocou jeho internetovej URL adresy:

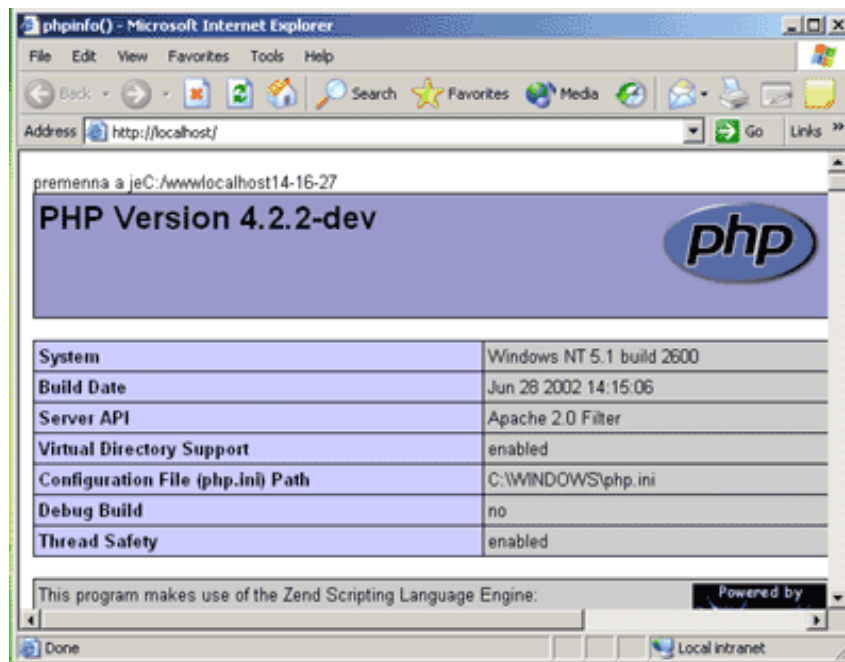
<http://localhost>
<http://localhost/phpMyAdmin>
<http://localhost/index.php>
http://localhost/projekt_omega/test.php

K parsovaniu samozrejme nedíjde, ak skript zavoláme pomocou jeho lokálnej adresy v súbrovom systéme, ako napr.

[C:\www\adresar_one\test.php](C:\www\adresar_one\test.php)

Správne volanie je

http://localhost/adresar_one/test.php



Pre testovanie konfigurácie PHP sa používa funkcia **phpinfo()**. Preto ako prvý naozajstný príklad použijeme túto funkciu na vypísanie konfigurácií nastavení PHP.

Príklad 4-1. test.php

```
<?php

// napis v notepade a uloz do webov@ho adresara ako subor
test.php (pozor! nie test.php.txt)

phpinfo();

?>
```

Po uložení súboru do webov@ho adresára (pozri httpd.conf, direktívu **DocumentRoot**, napr. DocumentRoot "C:/www" je webov@ adresár [c:\www](http://localhost/test.php)) spusti test.php cez prehliadač

<http://localhost/test.php>

V prípade úspechu uvidíme výstup funkcie phpinfo(), v prípade neúspechu popis chyby. Ak súbor neexistuje (alebo sme ho uložili s príponou txt, v notepade sa to stáva), server vypíše chybu 404.

5. Editor a prehliadač

Pre editáciu skriptov v PHP máme použiť profesionálne vývojárske prostredie typu IDE, rozšírené editory alebo aj základné editory typu notepad alebo vi. Každý nástroj má určité výhody a nevýhody a spôsob práce s ním je v drobnostiach odlišný. V dlhodobej vývojárskej praxi sa najviac osvedčili nasledujúce nástroje

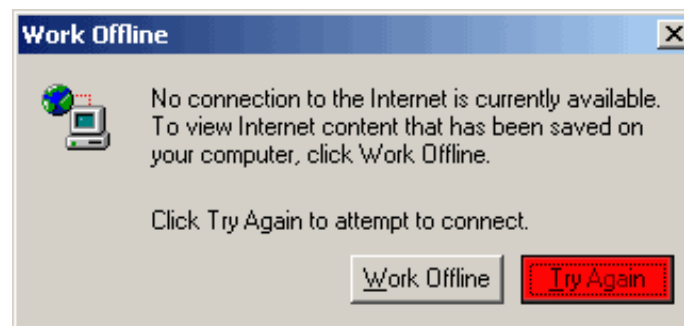
- **Zend Studio** - profesionálny IDE nástroj od www.zend.com, cena cca 150 USD/licencia, zabudovaný debugger, správa projektov, multiplatformný nástroj v Jave
- **Macromedia HomeSite** - profesionálny editor na tvorbu Webov, zvládanie syntaxe, nepodporuje debugger, cena od 5000 \$k/licencia, len pre Windows www.macromedia.com
- **UltraEdit** - programátorský editor, zvládanie syntaxe, bez podpory debuggera, len Windows verzia www.ultraedit.com
- **PHP Coder Pro** - IDE nástroj, podpora debuggera, zadarmo, len pre Windows www.phpide.de
- **Emacs, ViM** - editor, bez podpory debuggera, Windows aj Linux www.gnu.org

Pri debugingu v IDE je možné trasovať program priamo v okne IDE, pri použití editora bez podpory debuggera je nutné stránku reloadovať pomocou príslušnej klávesy (Refresh). profesionálny kompilátor ZEND Compiler je dodávaný s príslušným IDE.



Obr. 7 - nastavenie IE na nepripájanie sa k internetu

Ak sme pripojen² pomocou sieťov^oho pripojenia (poľtaľ mš sieťov⁴ kartu alebo podobn^o zariadenie) je nutn^o nastaviš PROXY server v nastaven² prehliadaľa tak, aby bola voľba **Bypass proxy server for local address** zaškrtnutá (Nepoužívaš proxy server pre lokálne WWW servery).



Obr. 8 - Dial·g pre pripojenie k lokálnemu WWW serveru

6. Databáza a jej správa

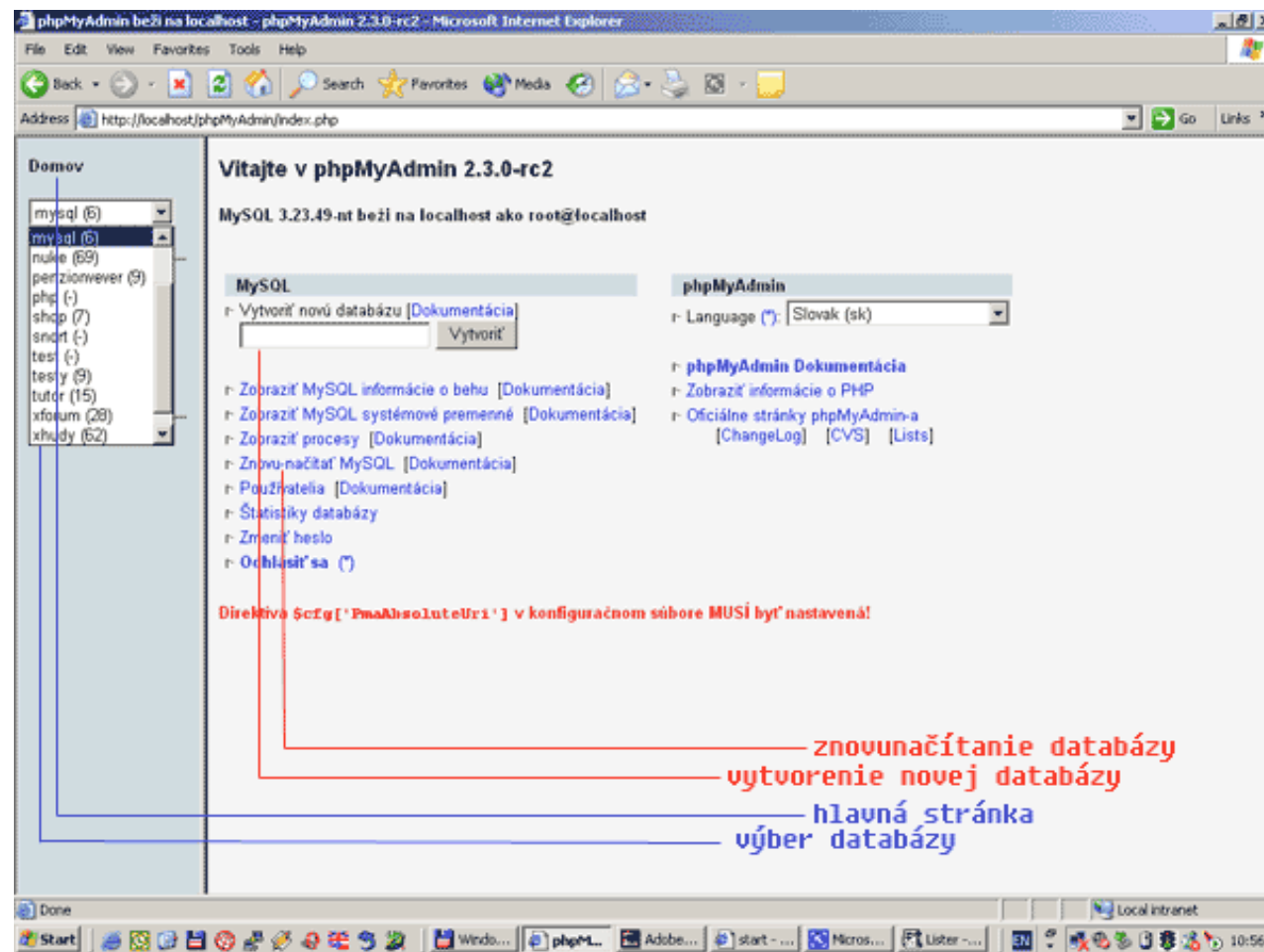
Pri práci s databázou MYSQL je vhodn^o použiť administračn^ý nástroj [phpMyAdmin](#), ktor^ý je samotn^ý naprogramovan^ý v PHP. Pre komunikáciu s databázou sl^uži buň nat²vny drajver (ovlšdaľ) alebo všeobecn^o ovlšdaľe typu ODBC a JDBC. Typ ovlšdaľa je nutn^o zvoliš pri kompilácii PHP alebo v s⁴bore php.ini. Zabudovan^o ovlšdaľe vypisuje funkcia **phpinfo()**.

Po spustení nástroja phpMyAdmin (URL <http://localhost/phpMyAdmin>) sa zobrazí okno pre zadanie loginu a hesla. Je nutné zadať rovnaký login a heslo ako pri vstupnom inicializovaní databázy MySQL, inak sa phpMyAdmin nemôže k databáze pripojiť. Prípadne zmeny nastavenia phpMyAdmin sa prevádzajú v súbore **config.inc.php**

Nastavenie phpMyAdmin-a v súbore config.inc.php, najdôležitejšie vlasti sú zvislým.

```
/**
 * Server(s) configuration
 */
$i = 0;
// The $cfg['Servers'] array starts with $cfg['Servers'][1]. Do not use $cfg['Servers'][0].
// You can disable a server config entry by setting host to ''.
$i++;
$cfg['Servers'][$i]['host'] = 'localhost'; // názov servera alebo jeho IP
$cfg['Servers'][$i]['port'] = ''; //
$cfg['Servers'][$i]['socket'] = ''; //
$cfg['Servers'][$i]['connect_type'] = 'tcp';
$cfg['Servers'][$i]['controluser'] = '';
$cfg['Servers'][$i]['controlpass'] = '';
$cfg['Servers'][$i]['auth_type'] = 'http'; // Authentication method (config, http or cookie based)?
$cfg['Servers'][$i]['user'] = 'užívateľ'; // MySQL user
$cfg['Servers'][$i]['password'] = 'heslo'; // MySQL password (only needed
```

Po spustení phpMyAdmin-a sa zobrazí hlavná stránka, z ktorej je možné riadiť databázu systému MySQL. Podrobnejší popis phpMyAdmin-a nájdete v dokumente [PhpMyAdmin - užívateľský sprievodca](#), ktorý je k dispozícii pre členov PHP klubu. Nečlenovia môžu využiť dokumentáciu k danému software.



Obr. 9 - phpMyAdmin

Dôležitá údaje sú v RDBMS MYSQL uložené v databáze mysql. Po akejkoľvek zmene údajov je nutné databázu MySQL znovunačítať (pozri obr. 9).

Dôležitá tabuľky v databáze MYSQL:

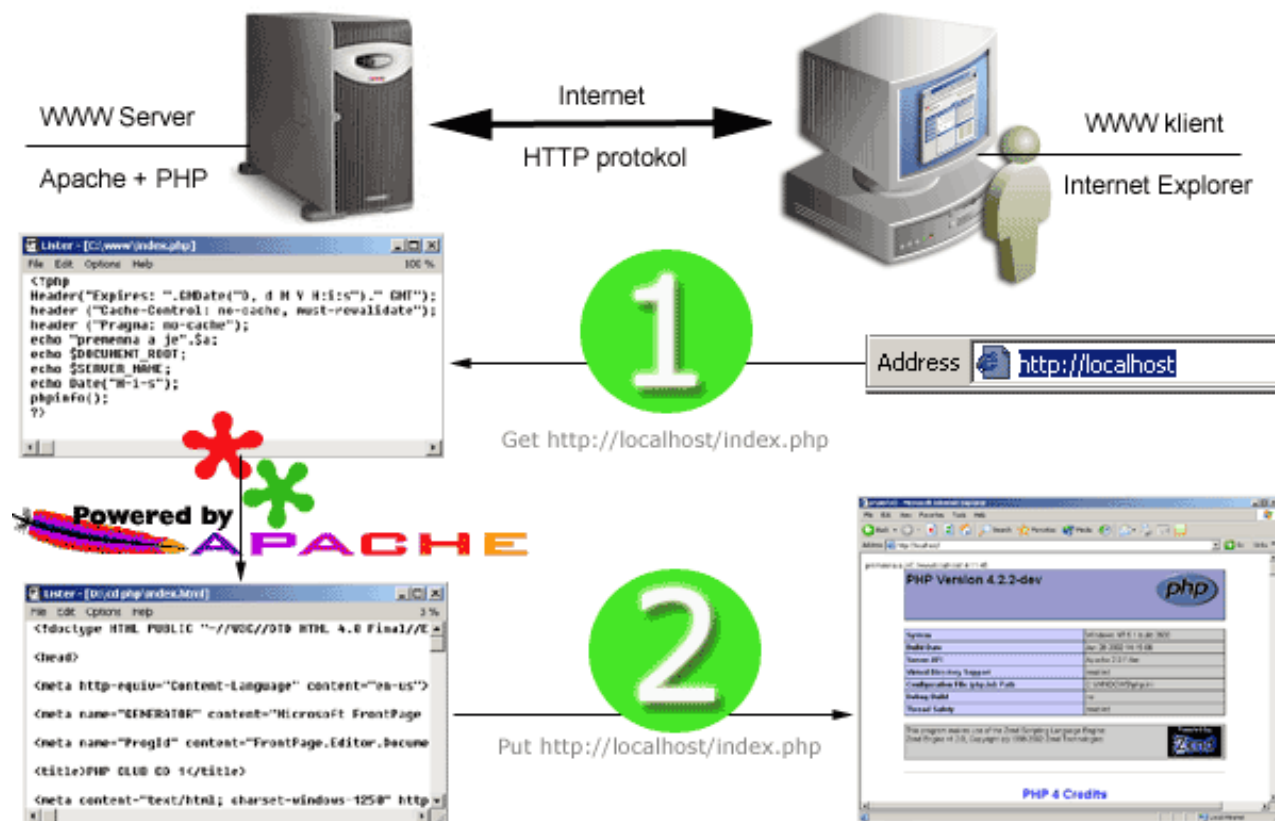
- **USER** - globálna tabuľka s užívateľmi a globálnymi prístupovými právami. Odporúčame všetkým užívateľom okrem admina nastaviť práva na N /No/ a práva k prístupnej databáze nastaviť v tabuľke DB.
- Pri ukladaní hesiel je nutné použiť funkciu PASSWORD!!!**
- **DB** - lokálna tabuľka s prístupovými právami k jednotlivým databázam.

7. Základná syntax jazyka

V nasledujúcich kapitolách je opísaná základná syntax a príklady jazyka PHP. Pri čítaní je vhodné ihneď odskúšať tu zverejnené príklady, resp. pokúšať sa o naprogramovanie podobných konštrukcií, aby Vám PHP prešlo ako sa hovorí "do krvi". Uloženie sa programovacieho

jazyka bez skľáňania a elaborovania nemš podľa mojich skľsenosť v a ľgť v Ľznam, vhodn® je len napr²klad pri prechode z jedn®ho programovacieho jazyka na druhĽ.

Ako som uť spom²nal, jazyk PHP sl¼gi na vygenerovanie HTML stršnky. Nakoľko ide o v a ľginou interpretovanĽ skript, vĽstupom je HTML dokument. Sp¹sob vygenerovania HTML dokumentu pomocou jazyka PHP je zobrazenĽ na obršzku 10.



Obr. 10 - sp¹sob generovania HTML dokumentu

Tvorba HTML dokumentu pomocou PHP prebieha vo viacerĽch fšzach. Pribliťne s¼ zobrazen® na predchšdzaj¼com obršzku:

1. Klient uvedie URL adresu dokumentu, ktorĽ chce zobrazĽš. PrehliadaĽ pomocou HTTP (Hyper Text Transfer protokol) poĽiada server o vrštenie poĽadovan®ho dokumentu.

GET <http://localhost/index.php>

2. Server zist², ě nejde o statickĽ dokument (podľa pr²pony .php) a zavolš modul PHP, ktorĽ interpretuje PHP dokument a vĽsledok vršti WWW prehliadaĽu (vršti vygenerovanĽ HTML dokument). Server vygenerovanĽ HTML dokument odoĽle pomocou met²dy PUT HTTP protokolu klientskej strane, ktorš ho zobraz². V pr²pade, ě pr²de poĽiadavka na generovanie statick®ho WWW dokumentu (napr²klad s pr²ponou .html), modul PHP nie je volanĽ. Nastavenie, ktorĽ dokument interpretovaš a ktorĽ nie, je moťn® v s¼bore **httpd.conf**

Jazyk PHP je samozrejme moťn® kombinovaš s klasickĽmi "HTML tagmi". Preto je znalosť HTML nevyhnutnš pre kaĽd®ho webov®ho programštora.

Zapízdrenie do HTML

Príklad 7-1. Zapízdrenie

```
1
2 1. <? echo ("toto je najjednoduché spísob\n"); ?>
3
4 2. <?php echo("ak chcete spracováš XML dokumenty, použite radšej tento\n"); ?>
5
6 3. <script language="php">
7     echo ("niektoré editory (ako FrontPage) neznášgají
8     predchádzajúci spísob");
9 </script>
10
11 4. <% echo ("Máte tiež použíš aj ASP tagy"); %>
12 <%= $variable; # Toto je skratka pre "<%= echo .." %>
13
```

Prvá možnosť je k dispozícii len ak je zapnutá možnosť spracovávania skrátených tagov. Zabezpečí to funkcia **short_tags()**, prípadne konfiguračná voľba **short_open_tag** v PHP konfiguračnom súbore, či pri kompilácii direktíva `--enable-short-tags`. Ďtvrtá možnosť je používať ASP tagy. Musí to byť povolené v nastavení **asp_tags**. Ak je to možné, vyhýbajte sa tomuto štýlu programovania, PHP je dosť dobrý na to, aby sa nemuselo po ASP opíliť.

Odporúčam používať tagy **<?php a ?>**

Oddelenie príkazov a komentáre

Príkazy sú od seba oddelené podobne ako v PERLE alebo jazyku C - bodkočiarkou. Koncový tag (**?>**) znamená tiež koniec príkazov, takže nasledujúce dva príklady sú rovnocenné. PHP samozrejme podporuje viac príkazov na jednom riadku.

Príklad 7-2. Oddelenie príkazov

```
1
2 <?php
3     echo "Toto je test";
4 ?>
5
6 <?php echo "Toto je test" ?>
7
```

PHP podporuje komentáre v štýle 'C', 'C++' a Unix.

Príklad 7-3. Komentáre

```

1
2 <?php
3     echo "test"; // Toto je jednoriadkoví komentšr
4     /* Toto je viacriadkoví
5        komentšr */
6     echo "iní test";
7     echo "poslední test"; # Toto je komentšr v štíle UNIXu
8 ?>
9

```

Jednoriadkoví komentšr zahŕňa blok do konca riadku alebo ukončovacieho tagu, podľa toho, čo prde skír.

Príklad 7-4. Komentšre

```

1
2 <h1>Toto je <?# echo "jednoduchy";?> príklad.</h1>
3 <p>Hlavička vypge 'Toto je príklad'.
4

```

PHP však nepodporuje tzv. hniezdenie komentšrov, nasledujúci príklad je preto nefunkčný.

Príklad 7-5. Komentšre

```

1
2 <?php
3 /*
4     echo "Toto je test"; /* Tento komentšr robí problém
5 */
6 ?>
7

```

8. Typy

Všetky premenné v PHP sú určitého typu, podobne ako čísla v reálnom svete. PHP podporuje nasledujúce typy

- **pole**
- **desatinné číslo**
- **celé číslo**
- **objekty**
- **reťazce**

Typ použitej premennej spravídla nezadáva programštor, ale automaticky vyberš PHP podľa povahy uloženej informšcie. Ak chcete sami stanoviš typ použitej premennej, je možné **pretypovaš** premennú pomocou funkcie **settype()**. Nezabudnite, že premenná sa môže správaš rôzne podľa ich typu a situšcie.

- **Integer - celé číslo**

Celé číslo môže byť použit s nasledujúcou syntaxou:

Príklad 8-1. Rôzne typy

```
1
2 $a = 1234; # desatinné číslo
3 $a = -123; # záporné číslo
4 $a = 0123; # osmičkový číslo (rovnaké ako 83d)
5 $a = 0x12; # šestnáctkový číslo (rovnaké ako 18d)
6
```

• Desatinné číslo

Desatinné číslo ("double") môže byť definované nasledovne:

Príklad 8-2. Desatinné číslo do premennej

```
1
2 $a = 1.234; $a = 1.2e3;
3
```

• Rešazce

Rešazce môže byť ohraničené dvomi druhmi oddeľovačov.

Ak je rešazec ohraničený úvodzovkami ("), premenná v jeho tele sa spracováva. Ako v jazyku C a PERL, spätné lomítko ("\\") je nutné použiť pri definovaní špeciálnych znakov.

Tabuľka 8-3. Špeciálne znaky

sekvencia	význam
\\n	nový riadok
\\r	vozík
\\t	vodorovný tabuľtor
\\\\	spätné lomítko
\\\$	znak dolára
\\"	úvodzovka
\\[0-7]{1,3}	reg. výraz /oktálny/
\\x[0-9A-Fa-f]{1,2}	reg. výraz /hexadecimálny/

Druhou možnosťou je uzavrieť rešazec do apostrofov ("""). Tento rešazec nebude vo vnútri parovaní, špeciálne znaky nie je možné použiť - buď zobrazené tak ako sú.

Ďalšou možnosťou je použiť dokumentálnu syntax. ("<<<"). Dokumentácia začína za týmto znakom a končí pri tagu, ktorý je uvedený za ním (napríklad EOD). Tento tag musí byť na začiatku riadka.

Príklad 8-4. Rôzne typy

```
1
2 $str = <<<EOD
3 Príklad of string
4 spanning multiple lines
5 using heredoc syntax.
6 EOD;
7
```

Pozn: Pridané v PHP 4.

Reťazce je možné spájať pomocou operácie čiarky '.' Znak reťazca s¼ vlastne znakovým poľom, preto k nim môžeme aj takto pristupovať:

Príklad 8-5. Reťazec

```
1
2 <?php
3 /* priradenie. */
4 $str = "This is a string";
5
6 /* pridanie. */
7 $str = $str . " with some more text";
8
9 /* iné pridanie */
10 $str .= " and a newline at the end.\n";
11
12 /* Tento reťazec končí číslom '<p>Number: 9</p>' */
13 $num = 9;
14 $str = "<p>Number: $num</p>";
15
16 /* Toto bude '<p>Number: $num</p>' */
17 $num = 9;
18 $str = '<p>Number: $num</p>';
19
20 /* Vráti prvý znak reťazca */
21 $str = 'This is a test.';
22 $first = $str[0];
23
24 /* Vráti posledný znak reťazca. */
25 $str = 'This is still a test.';
26 $last = $str[strlen($str)-1];
27 ?>
28
```

Reťazcové konverzie

Ak je reťazec vrátený ako číslo, jeho typ bude nasledujúci:

Bude typ double (desatinný) ak bude obsahovať jeden zo znakov '.', 'e', alebo 'E'. Inak bude celočíselný (integer).

Hodnota čísla je daná úvodnou hodnotou reťazca. Ak je to slo, bude toto slo, v inom prípade to bude nula. (0). Platí dĺžka za najnepovinným znamienkom, po ktorom nasleduje jedno alebo viac čísič. (voliteľne aj s desatinnou čiarkou), nasledovaný nepovinným exponentom. Exponent je znak 'e' alebo 'E' nasledovaný jednou alebo viacerými číslami.

Ak je prvý výraz reťazec, typ premennej bude závisieť na druhom reťazci.

Príklad 8-6. Reťazcová konverzia

```
1
2 $foo = 1 + "10.5";           // $foo je double (11.5)
3 $foo = 1 + "-1.3e3";         // $foo je double (-1299)
4 $foo = 1 + "bob-1.3e3";      // $foo je integer (1)
5 $foo = 1 + "bob3";           // $foo je integer (1)
6 $foo = 1 + "10 malých svin"; // $foo je integer (11)
7 $foo = 1 + "10 malých sviniek"; // $foo je integer (11)
8 $foo = "10.0 svin " + 1;     // $foo je integer (11)
9 $foo = "10.0 svin " + 1.0;   // $foo je double (11)
10
```

Ak si chcete otestovať tento príklad, za každú riadok zadajte nasledujúci príkaz:

Príklad 8-7. Zistenie typu

```
1
2 echo "\$foo==\$foo; typ je " . gettype( $foo ) . "<br>\n";
3
```

Polia

• Jednorozmerné polia

PHP podporuje skaláry a asociatívne polia. V podstate medzi nimi nie je žiadny rozdiel, môžete ich vytvoriť pomocou funkcie **list()** alebo **array()** alebo explicitne definovať každý prvok poľa:

Príklad 8-8. Naplnenie poľa

```
1
2 $a[0] = "abc";
3 $a[1] = "def";
4 $b["foo"] = 13;
5
```

Vytvoríš m¹žete tiež pole prost¹m pridan²m hodn¹t do po¹a. V pr²pade ě pouĝijete pr¹zdne zloĝen¹ z¹tvorky, prvok bude zaraden¹ na koniec po¹a.

Pr²klad 8-9. Naplnenie po¹a bez pouĝitia indexu

```
1
2 $a[] = "ahoj"; // $a[2] == "ahoj"
3 $a[] = "svet"; // $a[3] == "svet"
4
```

Polia m¹ĝeme triedi¹ pomocou **asort()**, **arsort()**, **ksort()**, **rsort()**, **sort()**, **uasort()**, **usort()** a **uksort()** funkci².

Funkcia **count()** vr¹ti po¹et prvkov po¹a.

Po poli m¹ĝete prech¹zda¹ pomocou funkcie **Next()** a **Previous()**. Inou ob¹benou moĝnos¹ou je zavola¹ funkciu **each()**.

▮ Viacrozmern¹ polia

Viacrozmern¹ polia s¹ ve¹mi jednoduch¹, sta¹ prida¹ in¹ [k¹ov¹] hodnotu na koniec. Viacrozmern¹ pole je moĝn¹ naplni¹ r¹znymi sp¹sobmi.

Pr²klad 8-10. Naplnenie viacrozmern¹ho po¹a

```
1
2 $a[1]      = $f;           # jednorozmerné polia
3 $a["foo"]  = $f;
4
5 $a[1][0]   = $f;           # dvojrozmerné pole
6 $a["foo"][2] = $f;         # (mix asociatívneho a numerického po1a)
7 $a[3]["bar"] = $f;         #
8
9 $a["foo"][4]["bar"][0] = $f; # štvorrozmerné pole
10

1
2 $a[3]['bar'] = 'Bob';
3 echo "This will work: {$a[3][bar]}";
4
```

Pr²klad 8-11. Naplnenie viacrozmern¹ho po¹a

```

1
2 # Príklad 1:
3
4 $a["color"] = "red";
5 $a["taste"] = "sweet";
6 $a["shape"] = "round";
7 $a["name"]  = "apple";
8 $a[3]      = 4;
9
10 # Príklad 2:
11 $a = array(
12     "color" => "red",
13     "taste" => "sweet",
14     "shape" => "round",
15     "name"  => "apple",
16     3      => 4
17 );
18

```

Funkcia **array()** môže byť vložená aj pre viacrozmerné polia

Príklad 8-12. Naplnenie viacrozmerného poľa pomocou array

```

1
2 <?
3 $a = array(
4     "apple" => array(
5         "color" => "red",
6         "taste" => "sweet",
7         "shape" => "round"
8     ),
9     "orange" => array(
10        "color" => "orange",
11        "taste" => "tart",
12        "shape" => "round"
13    ),
14    "banana" => array(
15        "color" => "yellow",
16        "taste" => "paste-y",
17        "shape" => "banana-shaped"
18    )
19 );
20
21 echo $a["apple"]["taste"];    # vypíše "sweet"
22 ?>
23

```

Pretypovanie

PHP nepodporuje explicitné určenie typu premennej pri jej deklarácii, jej typ je určený pri jej prvom použití². Znamená to, že ak do

premennej priradíte na začiatku rešazec, bude typu rešazec.

Príklad automatickej konverzie typov je operácia sčítania. '+'. Ak je jeden z operandov double, všetky operandy sú vyhodnotené ako double. a výsledok bude tiež double. Inak budú operandy interpretované ako celočíselný typ integer a výsledok bude tiež integer.

Príklad 8-13. Automatické pretypovanie

```
1
2 $foo = "0"; // $foo je rešazec (ASCII 48)
3 $foo++; // $foo je rešazec "1" (ASCII 49)
4 $foo += 1; // $foo je teraz integer (2)
5 $foo = $foo + 1.3; // $foo je teraz double (3.3)
6 $foo = 5 + "10 malých svin "; // $foo je integer (15)
7 $foo = 5 + "10 malých svin"; // $foo je integer (15)
8
```

Ak chcete vyskúšať jeden z príkladov, doplňte nasledujúci riadok za každí príkaz.

Príklad 8-14. Určenie typu

```
1
2 echo "\$foo==\$foo; type is " . gettype( $foo ) . "<br>\n";
3
```

Pretypovanie

Príklad 8-15. Pretypovanie

```
1
2 $foo = 10; // $foo is an integer
3 $bar = (double) $foo; // $bar is a double
4
```

Povolené pretypovanie je:

- (int), (integer) - na integer
- (real), (double), (float) - na double
- (string) - na rešazec
- (array) - na pole
- (object) - na objekt

Nezabudnite, že tabuľstory a medzery sú povolené.

Príklad 8-16. Pretypovanie

```
1
2 $foo = (int) $bar;
3 $foo = ( int ) $bar;
4
```

Nie je Všetm jasné, čo sa deje? Napríklad pri pretypovaní skaláru na pole sa stane premenná prvým prvkom poľa:

Príklad 8-17. Pretypovanie na pole

```
1
2 $var = 'ahoj';
3 $arr = (array) $var;
4 echo $arr[0]; // vypise 'ahoj'
5
```

Podobne je to aj s objektom.:

Príklad 8-18. Pretypovanie na objekt

```
1
2 $var = 'zdar';
3 $obj = (object) $var;
4 echo $obj->scalar; // vypise 'zdar'
5
```

9. Premenné

Premenná sa v PHP reprezentovaná znakom dolný, za ktorým nasleduje názov premennej. Názov je citlivý na veľkosť písmen. Nezabudnime na to, že protokol HTTP je bezstavový, čo znamená, že všetky premenné strácajú svoju platnosť po ukončení behu skriptu. Jedinou možnosťou je uložiť premennú pomocou cookie na strane klienta alebo pomocou session na strane servera.

Príklad 9-1. Naplnenie premennej

```
1
2 $var = "Bob";
3 $Var = "Joe";
4 echo "$var, $Var"; // vypíše "Bob, Joe"
5
```

V PHP sa implicitne použije premenná, ktorá sa vždy volaná hodnotou. Znamená to, že pri priradení jednej premennej do druhej je vždy skopírovaná jej hodnota zo zdrojovej premennej do cieľovej. Neskoršia zmena jednej hodnoty však neovplyvní hodnotu druhej premennej, zostávajú na sebe nezávislé. Previac informácií o tomto type priradenia pozri kapitolu [Výrazy](#).

PHP ponúka inú možnosť volania premennej - odkazom. Znamená to, že pri priradení jednej premennej do druhej sa nová premenná stáva akýmsi aliasom, obidve premenné ukazujú na rovnaké miesto v pamäti, a to vlastne pointre. Pri zmene jednej premennej sa automaticky zmení aj obsah druhej premennej. Priradenie je tiež rýchlejšie, nakoľko nie je nutné ťahanie kopírovanie hodnoty, toto

zrýchlenie je však merateľné len vo veľkých cykloch, prípadne pri prstupe k veľkým objektom či poliam.

Pre volanie odkazom je nutné pred názov premennej zaradiť znak ampersand (&).

Príklad 9-2. Volanie odkazom

```
1
2 <?php
3 $foo = 'Bob';           // prirad hodnotu 'Bob' do $foo
4 $bar = &$foo;           // referencia $foo via $bar.
5 $bar = "My name is $bar"; // zmen $bar...
6 echo $foo;              // $foo sa zmenilo tiež
7 echo $bar;
8 ?>
9
```

Nezabudnite, že len pomenované premenné môžu byť volané odkazom! Sú to vlastne premenné, ktoré majú meno, nie je to napr. matematický výraz.

Príklad 9-3. Pomenované premenné

```
1
2 <?php
3 $foo = 25;
4 $bar = &$foo;           // Platné priradenie.
5 $bar = &(amp;24 * 7);    // Chyba; volanie nepomenovanej premennej
6
7 function test() {
8     return 25;
9 }
10
11 $bar = &test();         // Invalid.
12 ?>
13
```

Preddefinované premenné

PHP poskytuje bežiacim skriptom veľké množstvo preddefinovaných premenných. Veľa z nich však nemôže byť úplne popísané, nakoľko závisia od servera, na ktorom bežia, od verzie OS a od iných faktorov. Niektoré z nich nie sú k dispozícii, ak beží PHP ako CGI wrapper, prípadne ako program spustený z príkazového riadku. Pre úplný zoznam preddefinovaných premenných prosím pozrite a použite funkciu **phpinfo()**.

Platnosť premennej

Platnosť premennej závisí od kontextu, v akom bola definovaná. Na príklad:

Príklad 9-4. Platnosť premennej

```
1
2 $a = 1;
3 include "b.inc";
4
```

Tu máš premennú \$a platnú aj v includovanom súbore **b.inc**. Napríklad v užívateľsky definovanej funkcii máš premennú aj lokálnu platnú - je inicializovaná len v tele funkcie.

Príklad 9-5. Globálna a lokálna premenná

```
1
2 $a = 1; /* globálna premenná */
3
4 Function Test () {
5     echo $a; /* referencia na lokálnu premennú */
6 }
7
8 Test ();
9
```

Tento skript nevypíše nič, nakoľko lokálna premenná \$a neobsahuje nič, navyše nebola ani inicializovaná. Tu je známy jeden rozdiel oproti jazyku C, kde globálne premenné sú automaticky prístupné aj v lokálnej funkcii, ak neboli prepísané lokálnou premennou. V PHP musia byť globálne premenné deklarované pomocou slova global v tele funkcie:

Príklad 9-6. Premenná typu global

```
1
2 $a = 1;
3 $b = 2;
4
5 Function Sum () {
6     global $a, $b;
7
8     $b = $a + $b;
9 }
10
11 Sum ();
12 echo $b;
13
```

Skript vypíše "3". Deklaráciou \$a a \$b ako globálne premenné v tele funkcie umožnilo ich naplnenie aj napriek tomu, že sú použité v tele funkcie. Ak dôjde k ich zmene v tele funkcie, táto zmena bude platná aj mimo nej. Druhou možnosťou použitia globálnych premenných je využitie špeciálne definovaného asociatívneho poľa \$GLOBALS:

Príklad 9-7. Pretypovanie na objekt

```

1
2 $a = 1;
3 $b = 2;
4
5 Function Sum () {
6     $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
7 }
8
9 Sum ();
10 echo $b;
11

```

Inou možnosťou je použitie statickej premennej. Statické premenné existujú len v tele funkcie, nestratí však platnosť, ak ju program opustí:

Príklad 9-8. Nefunkčná statická premenná

```

1
2 Function Test () {
3     $a = 0;
4     echo $a;
5     $a++;
6 }
7

```

Funkcia je nefunkčná, nakoľko vždy vypíše nulu a nastaví \$a na 0. Skúste však toto:

Príklad 9-9 Statická premenná

```

1
2 Function Test () {
3     static $a = 0;
4     echo $a;
5     $a++;
6 }
7

```

Statické premenné ponúkajú možnosť práce s rekurziou. Rekurzívna funkcia je funkcia, ktorá volá sama seba. Nasledujúca funkcia rekurzívne počítá do 10 a zastane.

Príklad 9-10. Statické počítadlo

```

1
2 Function Test () {
3     static $count = 0;
4
5     $count++;
6     echo $count;
7     if ($count < 10) {
8         Test ();
9     }
10    $count--;
11 }
12

```

Premenná premenných

Niekedy je užitočné použiť aj meno premennej, napríklad pri dynamickej inicializácii premenných. Normálna premenná je naplnená takto:

Príklad 9-11. Premenná premennej

```

1
2 $a = "hello";
3

```

Premenná premennej zoberie hodnotu premennej a spracuje ju ako názov premennej, takto:

Príklad 9-12. Premenná premennej

```

1
2 $$a = "world";
3

```

V tomto bode boli definované dve premenné v PHP. \$a obsahuje "hello" a premenná \$hello obsahuje "world". Preto príkaz:

Príklad 9-13. Premenná premennej

```

1
2 echo "$a ${$a}";
3

```

bude generovať rovnaký výsledok ako

Príklad 9-14. Premenná premennej

```
1
2 echo "$a $hello";
3
```

Ľo znamenš, ģe oba vyp²ĝu *hello world*.

V pr²pade, ģe pouĝ²vate premenn¼ premennej v s¼vislosti s poóom, je nutn® parseru urĽiš, Ľo je pole a Ľo premennš premennej. Pre pole `$$a[1]` mus²te zadaš : `${$a[1]}` alebo `${$a}[1]` - podôa toho, Ľo chcete.

10. HTML formuláre

HTML Formuláre (GET a POST)

Jazyk HTML obsahuje podporu formulšrov, ktor® sl¼ĝia pre naĽ²tavanie dšt od pouĝ²vateôa. Formulšr vyzerš ako dialogovĽ box v okne prehliadaĽa, m¹ĝe obsahovaš tlaĽ²tka, editovacie boxy, listboxy, zaĝkrtšvacie a prep²nacie tlaĽ²tka a in® komponenty. Script z²skan® dšta z formulšra spracuje a vĽsledok odoĝle na okno prehliadaĽa.

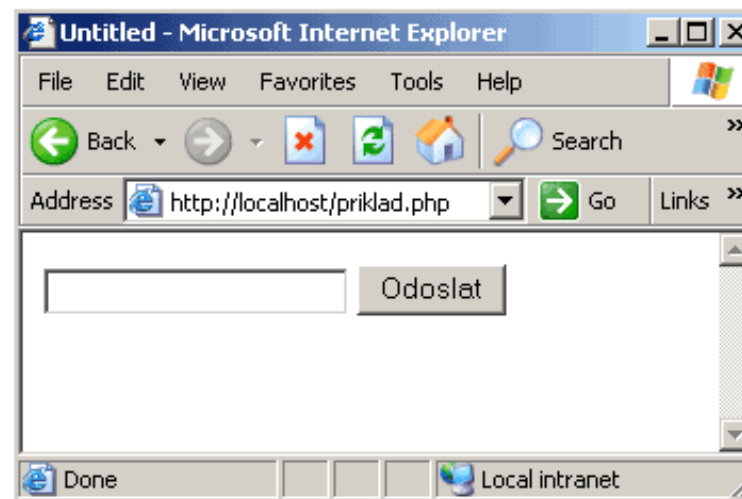
V jazyku HTML sl¼ĝi pre vĽoĝenie formulšra do stršnky tag **FORM**. Mš dva atrib¼ty **ACTION** a **METHOD**. Atrib¼t **ACTION** urĽ² URL adresu skriptu, ktorĽ sl¼ĝi pre spracovanie dšt z²skanĽch z dan®ho formulšra. Atrib¼t **METHOD** urĽ², akĽm sp¹sobom bud¼ dšta z formulšra predan® serveru, buĽ met²dou **GET**, ktorš pripoj² prenššgan® dšta za URL odsluĝn®ho skriptu alebo met²dou **POST**, ktorš zakomponuje prenššgan® dšta do tela HTTP. Zšklad formulšra by mal dodrĝiavaš nasleduj¼cu syntax:

```
<FORM ACTION=ôURL skriptuň METHOD=ňGET|POSTň>
...telo formulšra...
</FORM>
```

V pr²pade, ak URL skriptu nezadšme, spracovšva formulšr rovnakĽ skript, Ľo sa Ľasto vyuĝ²va. Ak je odoslanĽ HTML formulšr, vĝetky premenn® z tohto formulšra je moĝn® vyuĝiš a pouĝiš v PHP:

Pr²klad 10-1. JednoduchĽ HTML formulšr odoslanĽ pomocou met²dy GET a POST

```
1
2 <form action="skript.php" method="get" enctype="multipart/form-data">
3 <input type="text" name="meno">
4 <input type="submit" name="akcia" value="Odoslat">
5 </form>
6
7 <form action="skript.php" method="post" enctype="multipart/form-data">
8 <input type="text" name="meno">
9 <input type="submit" name="akcia" value="Odoslat">
10 </form>
11
```



Obr. 11 - HTML formulár

Vo formulároch je možné používať rôzne elementy, napríklad: TEXTAREA, INPUT, SELECT, LABEL, FIELDSET, BUTTON. Doporučujem preštudovať HTML príručku (www.w3c.org)

Po odoslaní PHP vytvorí premennú **\$name**, ktorá bude obsahovať to, čo ste zadali do políčka *Meno*: vo Vašom formulári.

Upozornenie!

PHP od verzie 4.2.x používa iný typ odovzdávanie premenných. Horeuvedený typ je nutné implicitne povoliť voľbou Register Globals = On v súbore php.ini.

V novej verzii PHP je nutné pristupovať k získaným premenným pomocou asociatívnych polí **\$HTTP_POST_VARS**, **\$HTTP_GET_VARS** a **\$HTTP_POST_FILES**

Príklad 10-2. Jednoduchý HTML formulár odoslaný pomocou metódy GET a POST

```

1 <form action="" method=post enctype="multipart/form-data">
2 <table frame=box rules=none border=1 cellpadding=4>
3 <tr><td> Meno: </td><td><INPUT TYPE=TEXT NAME=Meno SIZE=30></td></tr>
4 <tr><td> Priezvisko: </td><td><INPUT TYPE=TEXT NAME=Priezvisko SIZE=30></td></tr>
5 <tr><td> E-mail: </td><td><INPUT TYPE=TEXT NAME=Email SIZE=20></td></tr>
6 </table>
7
8 <B> Suhlasim s odoslaním tohto dotazníka: </B><BR><BR>
9
10 <input type=Submit name="Potvrdenie" value="Ano">
11 <input type=Submit name="Potvrdenie" value="Nie"><BR>
12 </form>

```

Ďalšie spracovanie premenných môže vyzeráť napríklad takto:

Príklad 10-3. Spracovanie jednoduchého formulára

```
1
2 <?php
3 echo $Meno; //povolene Register Globals = on
4 echo $HTTP_POST_VARS["Meno"]; // nove PHP
5 if ($HTTP_POST_VARS["Potvrdenie"]=="Ano")
6 echo "Dakujem za udaje";
7 else
8 echo 'No nic...';
9 ?>
10
```

PHP tiež podporuje polia vo forme formulárových premenných. Príklad získania poľa z multiselectu:

Príklad 10-4. Multiselect

```
1
2 <form action="array.php" method="post" enctype="multipart/form-data">
3     Meno: <input type="text" name="personal[name]"><br>
4     Email: <input type="text" name="personal[email]"><br>
5     Pivo: <br>
6     <select multiple name="beer[]">
7         <option value="tatran">Tatran
8         <option value="bazant">Zlaty Bazant
9         <option value="krusovice">Krusovice
10    </select>
11    <input type="submit">
12 </form>
13
```

Ak je zapnutá konfigurácia voľba PHP [track_vars](#) , potom je možné k získaným premenným pomocou metód POST či GET pristupovať aj v globálnych asociatívnych poliach **\$HTTP_POST_VARS**, **\$HTTP_GET_VARS**, **\$HTTP_POST_FILES**. V nových verziách je to možné kvôli bezpečnosti zapnúť **implicitne**.

OBRÁZKOVÉ TLAČIDLO

Pri odosielaní formulára je možné namiesto klasického textového tlačítka použiť aj obrázkové, napríklad využitím tagu:

Príklad 10-5. Obrázkové tlačidlo

```
1
2 <form action="" method="post" enctype="multipart/form-data">
3 <input type="image" src="image.gif" name="sub">
4 </form>
5
```

Pri stlaŕen² tlaŕidla bude serveru odovzdan^š aj poloha stlaŕenia pomocou premenn^šch **\$sub_x** a **\$sub_y**.

Pr²klad 10-6. Kompletn² HTML formul^šr so spracovan²m skript.php

```
1 // Register global = on
2 <?php
3 function spracujForm() {
4 global $odosli,$meno,$priezvisko,$heslo;
5 echo "Meno - ".$meno.NL;
6 echo "Priezvisko - ".$priezvisko.NL;
7 echo "Heslo - ".$heslo.NL;
8 echo '<a href="skript.php">Reload</a>';
9 } //spracujForm
10
11 function showForm() {
12 ?>
13 <html>
14 <head>
15 <title>Zadavanie dat</title>
16 </head>
17 <body>
18 <h1>Zadaj data</h1>
19 <hr>
20 <form action="" method="post">
21 Meno:      <input type="text" name="meno"><br>
22 Priezvisko: <input type="text" name="priezvisko"><br>
23 Heslo:     <input type="password" name="heslo"><br>
24 <input type="submit" name="odosli" value="Odoslat">
25 </form>
26 </body>
27 </html>
28 <?php
29 }
30
31 define("NL","<br/>\n");
32 if ($odosli=="Odoslat") spracujForm();
33 else showForm();
34 ?>
35
```

HTTP su^šienky (cookies)

PHP transparentne podporuje HTTP su^šienky pod²ľa defin²cie v [špecifik^šcii NETSCAPE](#). Su^šienky alebo Cookies je mechanizmus ur²len² pre uloŕenie d^št na vzdialenom prehliada²li a t²ŕm ozna²lenie a identifik^šciu u²ivate²ov. Su^šienku (cookie) je mo²ŕn² nastavi^š funkciou **SetCookie()**. Su^šienky s²ŕ ŕas^šou HTTP hlavi²ky, tak²e volanie SetCookie mus² nasta^š pred osolan²m hocak²ch d^št prehliada²u - toto obmedzenie plat² aj pre funkciu **Header()**. V²e²ky su^šienky s²ŕ automaticky konvertovan² na zodpovedaj²ce premenn² PHP - podobne ako pri met²·dach GET a POST.

Ak chcete nastaviť viac hodnôt v jednej sušienke pridajte `[]` k názvu sušienky. Príklad:

Príklad 10-7. Nastavenie cookie s platnosťou 1 hodina

```
1
2 SetCookie ("MojaCookie[]", "Test", time()+3600);
3
```

Nezabudnite, že každá sušienka nahradí predchádzajúcu s rovnakým názvom, pokiaľ nie je zmenený prehliadač ani doména. Príklad pre nákupnú košík, ktorý bude rešpektovať rozdielnych klientov - obmedzene...

Príklad 10-8. Cookie pre podporu košíka

```
1
2 $Count++;
3 SetCookie ("Count", $Count, time()+3600);
4 SetCookie ("Cart[$Count]", $item, time()+3600);
5
```

Premenné prostredia

PHP automaticky konveruje premenné prostredia na normálne PHP premenné.

Príklad 10-9. Výpis premennej HOME

```
1
2 echo $Home; /* zobrazí nastavenie premennej prostredia Home, ak je nastavená */
3
```

Nakoľko sú však informácie získané pomocou sušienok alebo metód GET a POST automaticky konvertované priamo na premenné, je niekedy užitočné explicitne preložiť premenné prostredia, napríklad pomocou funkcie **getenv()**. Premenné prostredia tiež môžete nastaviť pomocou funkcie **putenv()**.

Bodka v získaných premenných

PHP obvykle neparsuje získané premenné, pretože však bodka nie je povoleným znakom v názve premenných, PHP bude generovať chybu, ak Vaša premenná bude v názve obsahovať bodku.

Príklad 10-10. Nesprávny názov premennej

```
1
2 $varname.ext; /* chybný názov premennej */
3
```

PHP automaticky nahradí včetyky bodky znakom podtržník.

Získanie typu premennej

Nakoľko PHP neposkytuje mechanizmus explicitného určenia typu, je možné typ premennej indikovať volaním funkcie **gettype()**, **is_long()**, **is_double()**, **is_string()**, **is_array()**, a **is_object()**.

11. Konštanty

PHP definuje viac konštánt a poskytuje mechanizmus k ich definovaniu pri behu programu. Konštanty sa používajú rovnako ako premenné, inicializované sú však použitím funkcie **define()** a po inicializácii už nemôžu zmeniť hodnotu.

Predefinované konštanty sú:

__FILE__

Názov skriptu, ktorý je v súčasnosti spracovávaný. V prípade, že je použitý v includovanom súbore, je vrátené meno includovaného súboru, nie súboru, ktorý includovaný súbor zavola.

__LINE__

Číslo riadku v skripte, ktorý je práve parsovaný.

PHP_VERSION

Vráti číslo verzie PHP, napríklad '4.2.2'.

PHP_OS

Vráti názov operačného systému, na ktorom PHP beží, napríklad 'Linux'.

TRUE

hodnota true

FALSE

hodnota false.

E_ERROR

Označuje inú chybu než chybu pri parsovaní skriptu, nie je možné zotavenie.

E_WARNING

Označuje stav, kde PHP vie, že niečo nie je v poriadku, ale pokračuje ďalej v spracovaní skriptu. Príkladom môže byť chybný regex výraz vo funkcii **ereg()**.

E_PARSE

Parser oznámil chybu pri parsovaní skriptu. Zotavenie nie je možné.

E_NOTICE

Stalo sa niečo, čo môže byť chybou a nemusí. Spracovanie skriptu pokračuje aj ďalej.

Konštanty **E_*** sú používané vo funkcii **error_reporting()** pre definovanie úrovne chýb. Môžete tiež definovať aj iné konštanty použitím funkcie **define()**. Nezabudnite, že konštanty nie sú C-ľakovým makrom, iba platný skalár môže byť konštantou.

Príklad 11-1. Definovanie konštánty

```
1
2 <?php
3 define("CONSTANT", "Ahoj svet.");
4 echo CONSTANT; // vypise "Ahoj svet."
5 ?>
6
```

Príklad 11-2. Použitie __FILE__ a __LINE__

```
1
2 <?php
3 function reportError($file, $line, $message) {
4     echo "Nastala chyba v skripte $file na riadku $line: $message.";
5 }
6
7 reportError(__FILE__, __LINE__, "Niečo sa stalo!");
8 ?>
9
```

12. Výrazy

Výrazy sú základným stavebným kameňom PHP. V PHP je možné skoro všetko zapísať ako výraz. Najjednoduchším spôsobom, ako definovať výraz je "všetko čo má hodnotu".

Najľahšou formou výrazov sú konštanty a premenné. Ak napíšete "\$a = 5", priradíte hodnotu '5' do premennej \$a. '5', má obvykle hodnotu 5, alebo inými slovami '5' je výraz s hodnotou 5 (v tomto prípade je '5' celistvým konštantou).

Po tomto priradení máme oľakšovať, že premenná \$a bude mať tiež hodnotu 5, takže ak napíšeme \$b = \$a, bude to mať rovnakú hodnotu ako by sme napísali \$b = 5. Inými slovami, \$a je výraz s hodnotou 5.

Zložitejšie príklady výrazov sú **funkcie**:

Príklad 12-1. Jednoduchá funkcia vracajúca 5

```
1
2 function foo () {
3     return 5;
4 }
5
```

Predpokladám, že už rozumiete princípu funkcie. Ak nie, pozrite si prosím kapitolu venovanú funkciám. Ak napíšete **\$c = foo()** je to rovnaké akoby ste napísali **\$c = 5**. Funkcie sú teda výrazmi s hodnotou ich návratovej hodnoty. Keďže foo() vracia 5, hodnota výrazu 'foo()' je 5. Obvykle funkcie namiesto statického výrazu vracajú vypočítanú hodnotu.

PHP nepodporuje len celol²slen[®] hodnoty, ale aj in[®] typy skalšrov - necelol²seln[®] a rešazce. Skalšr je hodnota, ktorš uŕ nem¹ŕe byš rozdelen[®] na menšie ľasti, napr²klad ako pole. to znamenš, ŕe pole nie je skalšr. PHP tieŕ podporuje dva kompozitn[®] typy, ktor[®] nie s¼ skalšrmi - objekty a polia.

Uŕivatelia starŕch verzi² PHP/FI 2/PHP3 sa nemusia obšvaš. PHP spracovšva vřrazy rovnako ako in[®] jazyky. PHP je vřrazovo orientovanř jazyk, ľo znamenš, ŕe skoro vŕetko je vřraz. Napr²klad nieľo ako '\$b = (\$a = 5)' je vlastne skrštenš forma '\$a = 5; \$b = 5;' (bodkoľiarka znamenš koniec pr²kazu). nakoľko vřrazy s¼ spracovšvan[®] sprava doľava, m¹ŕeme tieŕ nap'saš: '\$b = \$a = 5'.

Inř dobrř pr²klad na pouŕitie vřrazov je inkrementšcia a dekrementšcia. Poznšme pre a post verziu třchto operšci². Jedinř rozdiel je v porad², v akom d¹jde k predaniu hodnoty a zvřŕeniuznřŕeniu pršluŕnej premennej.

ňalŕm znšmym typom vřrazov je porovnšvanie vřrazov. Tieto vřrazy vracaj¼ buŕ 0 alebo 1, ľo znalř FALSE /nepravda/ alebo TRUE /pravda/ y). PHP podporuje > (v^aľŕ neg), >= (v^aľŕ alebo rovnř), == (rovnř), != (nerovnř), < (menŕ) and <= (menŕ alebo rovnř). Tieto vřrazy sa naľastejŕie pouŕvaj¼ v podmienkach, napr²klad pri vetven² programu.

ňalŕm typom vřrazov je ternšrny operštor:

Pr²klad 12-2. Ternšrny operštor

```
1
2 $first ? $second : $third
3
```

Ak je hodnota prv[®]ho vřrazu nenulovš /pravda - true/, je vyhodnotenř druhř vřraz, inak je vyhodnotenř 3 vřraz.

Pr²klad 12-3. Vřrazy

```
1
2 function double($i) {
3     return $i*2;
4 }
5 $b = $a = 5;          /* priradi hodnotu 5 do premennych $a a $b */
6 $c = $a++;             /* post-inkrement, priradi povodnu hodnotu $a
7                        (5) do $c */
8 $e = $d = ++$b;        /* pre-inkrement, priradi zvysenu hodnotu
9                        $b (6) do $d a $e */
10
11 /* v tomto bode su $d a $e rovnake a maju hodnotu 6 */
12
13 $f = double($d++);     /* priradi dvakrat hodnotu $d pred zvysenim,
14                        2*6 = 12 do $f */
15 $g = double(++$e);     /* priradi dva krat hodnotu $e po zvyseni
16                        2*7 = 14 do $g */
17 $h = $g += 10;         /* najprv je $g zvysene na 10 a skonci s hodnotou 24.
18                        potom je tato hodnota priradena do
19                        premennej $h, a $h skonci tiez s hodnotou
20                        24. */
21
```

13. Operátory

Aritmetické operátory

Pamätajte sa ešte na zšklady zo školy?

Tabuľka 13-1. Aritmetické operátory

Príklad	názov	výsledok
$a + b$	Sčítanie	súčet a a b .
$a - b$	Odsčítanie	rozdiel a a b .
$a * b$	Násobenie	súčin a a b .
a / b	Delenie	podiel a a b .
$a \% b$	Modulo	zvyšok a pri delení b .

Operátory priradenia

Zškladní operátor priradenia je znak " $=$ ". Všet prvý dojem môže byť, že to znamená rovnosť. Nie. Znamená to, priradiť hodnotu výrazu na pravej strane výrazu na ľavej strane.

Príklad 13-2. Priradenie

```
1
2 $a = ($b = 4) + 5; // $a je teraz rovné 9, $b je nastavené na 4.
3
```

Okrem zškladného operátora priradenia existujú tiež "kombinované operátory" pre všetky binárne aritmetické operácie a operácie s reťazcami.

Príklad 13-3. Aritmetické operátory

```
1
2 $a = 3;
3 $a += 5; // nastaví $a na 8, tak ako keby sme povedali: $a = $a + 5;
4 $b = "Ahoj ";
5 $b .= "svet!"; // nastaví $b na "Ahoj svet!", rovnako ako $b = $b . "svet!";
6
```

PHP podporuje aj odkaz volaním, napríklad takto **\$premenna = &\$inapremenna**; Referencia odkazom znamená, že obidve premenné smerujú na rovnakú pamäťovú bloku, dŕža medzi nimi nie sú kopírované, ide vlastne o dva pointre, ktoré smerujú na rovnaké miesto a tým pádom majú aj rovnakú hodnotu. Zmenou hodnoty jedného automaticky zmeníme hodnotu druhého.

Logické a bitové operátory

Umožňujú zapnúť špecifický bit v INTEGERi.

Tabuľka 13-4. Logické operátory

Príklad	Názov	Výsledok
$a \& b$	And	Bity rovnaké v a aj b sú nastavené
$a b$	Or	Bity ktoré sú nastavené v a alebo b sú nastavené.
$a \wedge b$	Xor	Bity ktoré sú nastavené v a alebo b ale nie sú rovnaké.
$\sim a$	Not	Bity ktoré sú nastavené v a nie sú nastavené
$a \ll b$	posun vľavo	posunie bity doľava ***
$a \gg b$	posun vpravo	posunie bity doprava ***

Porovnávacie operátory

Umožňujú porovnať dve hodnoty

Tabuľka 13-5. Porovnávacie operátory

Príklad	Názov	Výsledok
$a == b$	rovné	True ak je a rovné b .
$a === b$	identické	True ak je a rovné ako b , a sú rovnakého typu. (len v PHP4)
$a != b$	nerovné	True ak nie je a rovné b .
$a < b$	menej	True ak je a menej než b .
$a > b$	väčšie	True ak je a väčšie než b .
$a <= b$	menej alebo rovné	True ak je a menej alebo rovné než b .
$a >= b$	väčšie alebo rovné	True ak je a väčšie alebo rovné než b .

Iný porovnávací operátor je "?:" (alebo ternárny) operátor, ktorý pracuje rovnako ako v jazyku C

Príklad 13-6. Ternárny operátor

```
1
2 (vyraz1) ? (vyraz2) : (vyraz3);
3
```

Tento operátor vykoná *expr2* ak *expr1* je true, inak vykoná *expr3*.

Operátory kontroly chýb

PHP podporuje jediný takýto operátor, operátor (@). Ak je vložený pred výraz v PHP, hociak chybové hlásenie generované týmto výrazom bude ignorované.

Ak je zapnuté **sledovanie_chýb**, všetky chybové hlásenia budú uložené do premennej **\$php_errormsg**.

Príklad 13-7. Porovnávacie operátory

```
1
2 <?php
3 /* vnútorná SQL chyba (extra apostrof): */
4 $res = @mysql_query( "select name, code from 'namelist' ) or
5     die( "Dopyt zlyhal - chyba je '$php_errormsg'" );
6 ?>
7
```

Výkonávacie operátory

PHP podporuje výkonné operátory (`). PHP sa pokúsi spustiť príkaz uvedený v týchto operátoroch ako príkaz SHELLu.

Príklad 13-8. Výkonný operátor

```
1
2 $output = `ls -al`;
3 echo "<pre>$output</pre>";
4
```

Inkrementačné/dekrementačné operátory

PHP podporuje syntax jazyka C pre inkrementačné a dekrementačné operátory.

Tabuľka 13-9. Inkrementačné/dekrementačné operátory

Príklad	názov	efekt
++\$a	Pre-inkrement	zvýši \$a o jeden a vráti \$a.
\$a++	Post-inkrement	vráti \$a, potom zvýši \$a o jeden
--\$a	Pre-dekrement	zníži \$a o jeden, potom vráti \$a.
\$a--	Post-decrement	vráti \$a, potom zníži \$a o jeden.

Nejaký užitočný príklad na precvičenie

Príklad 13-10. Inkrementácia a dekrementácia

```

1
2 <?php
3 echo "<h3>Postinkrement</h3>";
4 $a = 5;
5 echo "Má byť 5: " . $a++ . "<br>\n";
6 echo "Má byť 6: " . $a . "<br>\n";
7
8 echo "<h3>Preinkrement</h3>";
9 $a = 5;
10 echo "Má byť 6: " . ++$a . "<br>\n";
11 echo "Má byť 6: " . $a . "<br>\n";
12
13 echo "<h3>Postdekrement</h3>";
14 $a = 5;
15 echo "Má byť 5: " . $a-- . "<br>\n";
16 echo "Má byť 4: " . $a . "<br>\n";
17
18 echo "<h3>Predekrement</h3>";
19 $a = 5;
20 echo "Má byť 4: " . --$a . "<br>\n";
21 echo "Má byť 4: " . $a . "<br>\n";
22 ?>
23

```

Logické operátory

Tabuľka 13-11. Logické operátory

Príklad	názov	výsledok
\$a and \$b	And	True ak \$a aj \$b sú true.
\$a or \$b	Or	True ak jeden z \$a alebo \$b sú true.
\$a xor \$b	Or	True ak jeden z \$a alebo \$b sú true, ale nie obidva
! \$a	Not	True ak \$a nie je true.
\$a && \$b	And	True ak \$a aj \$b sú true.
\$a \$b	Or	True ak jeden z \$a alebo \$b sú true.

Príčina, že existujú rôzne operátory pre "and" a "or" je že pracujú s rôznou prioritou. (Pozri [Priorita operátorov](#).)

Priorita operátorov

Priorita operátorov definuje, v akom poradí sa budú vo výraze vyhodnocovať. Napríklad výraz `1 + 5 * 3`, je výsledok 16 namiesto 18 pretože operácia násobenie ("*") má vyššiu prioritu ako operácia sčítanie ("+").

Nasledujúca tabuľka udáva prioritu operátorov od najvyššej po najnižšiu.

Tabuľka 13-12. Priorita operátorov

Spracovanie	Operátor
zloava	,
zloava	or
zloava	xor
zloava	and
sprava	print
zloava	= += -= *= /= .= %= &= = ^= ~= <<= >>=
zloava	? :
zloava	
zloava	&&
zloava	
zloava	^
zloava	&
neasociatvna	== != ===
neasociatvna	< <= > >=
zloava	<< >>
zloava	+ - .
zloava	* / %
sprava	! ~ ++ -- (int) (double) (string) (array) (object) @
sprava	[
neasociatvna	new

Rešazcové operátory

Existujú dva rešazcové operátory. Prvým je spojovací operátor ('.'), ktorý vršti spojenie pravého a ľavého argumentu. Druhým je operátor spojovacieho priradenia ('.='). Pre viac informácií si prečítajte [Priradovacie operátory](#).

Príklad 13-13. Spájanie rešazcov

```

1
2 $a = "Ahoj ";
3 $b = $a . "svet!"; // teraz $b obsahuje "Ahoj svet!"
4
5 $a = "Ahoj ";
6 $a .= "svet!"; // teraz $a obsahuje "Ahoj svet!"
7

```

14. Riadiace príkazy

Všetky skripty PHP tvoria s@rie pr²kazov. Pr²kazom m¹ge byš priradenie, v¹raz, volanie funkcie, cyklus alebo podmienen¹ pr²kaz, pr²padne aj pršzdny pr²kaz. Pr²kazy sa obvykle kon¹ia bodko¹iarkou. Pr²kazy m¹gu byš združen® do blokov pr²kazov, ktor® s¼ ohranilen® zložen¹mi zštvorkami { }.

Riadiace pr²kazy tvoria logiku programu alebo skriptu. Podla ¼rovne zšpisu je možn® us¼diš, ak¹ vyspel¹ bol programštor, ktor¹ dan¼ aplikšciu vytvšral. Preto je vhodn® dodržovaš pšr zškladn¹ch pravidiel pri tvorbe programov. Tieto pravidlš zvyknem naz¹vaš aj...

Štábna kultúra:

T¼to kapitolu som sem zaradil schvšlne, nako¼ko si to ğiada situšcia. Zo vgeobecn¹ch sk¼senost² je znšme, ğe najme odpor¼lania t¹kaj¼ce sa ¼pravy programov vyvolšvaj¼ nechuš a odpor. Pokia¼ vğak chceme p²saš v¹lğie programy, je nutn® zaviesš nejak¹ poriadok a ak uğ nejak¹ zaviesš, tak je zbyto¼n® vym¹ğaš uğ vymyslen®. Takmer za kağd¹m odporu¼en²m sk¼sim uviesš aj pr²klad.

- Odsazujte o 2 medzery kağd¹ logick¹ ¼sek programu

Pr²klad 14-1. Odsadzovanie o 2 medzery

```
1
2 if ($a == $b) {
3     //vetva cislo 1
4     echo "A je rovne B";
5 }//end vetva 1
6 else {
7     //vetva 2
8     echo "A je ine ako B";
9 }//endif
```

- Nezab¼dajte vkladaš pršzdne riadky, zlepğuj¼ litate¼nosš programu
- Unšrne operšrory neodde¼ujte od operandu medzerou!

Pr²klad 14-2. Unšrne operštory

```
1
2 $i++;
3 $j = $i --    //zle!!!
4
```

- ěavš zloženš zštvorka ozna¼uj¼ca blok je vğdy v riadku sama. Podobne aj ukon¼uj¼ca zloženš zštvorka. Ďvodnš zštvorka m¹ge byš spolu s **if**, **else**, **for**, **while**, **switch**, **return** (pozri pr²klad 14-1)
- Kağd¹ v¹raz (priradenie) na samostatn¹ riadok. V¹nimku tvoria pr²kazy s tesnou s¼vislosšou.

Pr²klad 14-3. Tesnš s¼vislosš

```
1
2 $x = GetPositionX; $y = GetPositionY;
3
```

- Menš objektov, premenných či funkcií pŕŕte zŕsadne malšími pŕsmenami (prŕpadne prvŕ veľkŕ ĩ to je moja odchŕlka od normy, budem sa ŕalej snaŕiŕ). Vyuŕŕvajte podtrhovnŕk. Nepouŕŕvajte podobnŕ identifikŕtory a identifikŕtory odlŕŕenŕ len veľkosŕou pŕsma (syst, ssyt, Meno,meno).
- Menš objektov musia maŕ vŕznam, napr. **meno** a nie **m1**. Ak sa ku programu neskrŕ vrŕtite, nemusŕte zabiŕ 30 minŕt dumanŕm, ŕo premennŕ **x1zf4** vlastne znamenŕ. Nehovorŕm uŕ o prŕpadnŕch spolupraco vnŕkoch, ktorŕ majŕ chuŕ vraŕdiŕ pri lŕŕŕtenŕ Vŕŕho ŕpagetovŕho kŕdu.
- Tvorte menŕ racionŕlne, podŕa urŕitŕho predpisu, napr:

View_file, print_file, Get_Your_Name()

- vyuŕŕvajte tzv. prefixy napr C ĩ class, O ĩ object, a ĩ atribŕt, T ĩ typ, R ĩ record F-pole, P-pointer

Beŕne pouŕŕvanŕ vŕznamovŕ identifikŕtory

i,j,k - indexy, parametre cyklu

c,ch - znaky

m,n - ŕŕtaŕe

f,r - reŕŕlne ŕŕsla

p - pointer

s - reŕazec

- Nezabŕdajte na komentŕre

if

Prŕkaz if je jeden z najdŕŕŕŕtejŕch prŕkazov vŕetkŕch programovacŕch jazykov, vrŕtane PHP. Umoŕnuje vykonŕvaŕ ŕsek kŕdu v zŕvislosti od splnenia podmienky. Syntax je rovnakŕ ako v jazyku C.

Prŕklad 14-4. Œtruktŕra prŕkazu if

```
1
2 if (vŕraz1)
3     prŕkaz1
4
```

Ak je vŕraz *vŕraz1* vyhodnotenŕ ako pravdivŕ (TRUE) PHP vykonŕ prŕkaz *prŕkaz1*. Ak je vŕraz vyhodnotenŕ ako nepravdivŕ (FALSE), PHP bude prŕkaz1 ignorovaŕ - preskoŕŕ ho.

Nasledujŕci prŕkaz vypŕŕe a je vŕŕŕŕie ako b ak je premennŕ \$a vŕŕŕŕie ako premennŕ \$b.

Prŕklad 14-5. Jednoduchŕ if

```
1
2 if ($a > $b)
3     print "a je väčšie ako b";
4
```

V prípade, že chcete v závislosti od splnenia podmienky vykonať nielen jeden, ale viac príkazov, uzavrite ich do príkazového bloku:

Príklad 14-6. If s blokom

```
1
2 if ($a > $b) {
3     print "a je väčšie ako b";
4     $b = $a;
5 }
6
```

Príkaz `if` môže byť aj súčasťou iného `if`-u.

else

Najčastejšie sa však stretávame so situáciou, ak pri splnení podmienky potrebujeme vykonať príkaz A a pri nesplnení príkaz B. Nato slúži výborne príkaz **else**. Blok `else` sa vykoná vtedy, ak podmienka v prvej časti príkazu `/if/` nie je splnená (`FALSE`).

Príklad 14-7. if-else

```
1
2 if ($a > $b) {
3     print "a je väčšie ako b";
4 } else {
5     print "a je menšie alebo rovné b";
6 }
7
```

elseif

Táto konštrukcia opäť rozširuje možnosti príkazu **if**. Príkaz uvedený za **elseif** sa vykoná, ak nie je splnená podmienka pred **if** a zároveň je splnená podmienka za **elseif**. Môžeme použiť aj viackrát. Príkaz **else** sa vykoná len vtedy, ak nie je splnená žiadna podmienka **elseif**.

Môžeme použiť aj zložený príkaz, či už pomocou zložených zástavek alebo dvojbodky a príkazu `endif`.

Príklad 14-8. If-elseif-else

```

1
2 if ($a > $b) {
3     print "a je vacsie ako b";
4 } elseif ($a == $b) {
5     print "a je rovne";
6 } else {
7     print "a je mensie ako b";
8 }
9

```

Alternatívna syntax pre riadiace príkazy

PHP ponúka alternatívnu syntax pre riadiace príkazy ako `if`, `while`, `for`, a `switch`. V zložení je to zmena úvodnej zloženej zástvorky `{` za dvojbodku `:` a koncovej zloženej zástvorky `}` za `endif;`, `endwhile;`, `endfor;`, alebo `endswitch;`, - podľa toho, akú konštrukciu používate. Alternatívna syntax je v PHP len z historických dôvodov, v budúcnosti nebude ďalej podporovaná.

Príklad 14-9. Alternatívna syntax - blok

```

1
2 <?php if ($a == 5): ?>
3     A je rovné 5-tim
4 <?php endif; ?>
5

```

V predôhľadnom príklade, HTML blok "A = 5" je vložený vo vnútri príkazu `if` napísaného v alternatívnej syntaxi. HTML blok bude vypísaný len v prípade, keď \$a sa rovná číslu 5.

Podobne je to aj pri použití **else** a **elseif**.

Príklad 14-10. Alternatívna syntax if-elseif-else

```

1
2 if ($a == 5):
3     print "a je rovné 5";
4     print "...";
5 elseif ($a == 6):
6     print "a je rovné 6";
7     print "!!!";
8 else:
9     print "a nie je ani 5 ani 6";
10 endif;
11

```

while

`while` je najjednoduchší cyklus v PHP. Správa sa rovnako ako `while` v jazyku C.

Príklad 14-11. Syntax príkazu while

```
1
2 while (výraz) príkaz
3
```

Tento iteratívny príkaz testuje podmienky cyklu ešte pred prechodom cyklu, čo znamená že cyklus nemusí prebehnúť ani raz. Príkaz **while** použijeme, ak závisí ukončovací podmienka na nejakom príkaze v tele cyklu. Zadaný cyklus sa teda prevšdza len vtedy, ak je splnená podmienka na začiatku, preto sa mu aj hovorí cyklus s podmienkou na začiatku.

Pri tomto príkaze môžeme použiť aj alternatívnu syntax:

Príklad 14-12. Alternatívna syntax príkazu while

```
1
2 while (expr): statement ... endwhile;
3
```

Vypisí čísla od 1 do 10:

Príklad 14-13. Cyklus s podmienkou na začiatku

```
1
2 /* Príklad 1 */
3
4 $i = 1;
5 while ($i <= 10) {
6     print $i++; /* vypis čísla $i
7                 pred jeho inkrementáciou, tzv.
8                 post-inkrementácia */
9 }
10
11 /* Príklad 2 */
12
13 $i = 1;
14 while ($i <= 10):
15     print $i;
16     $i++;
17 endwhile;
18
```

do..while

Tento príkaz je ekvivalent Paskalovskému REPEAT-UNTIL. Podmienka sa tu testuje až po prechode cyklom, teda cyklus prebehne najmenej raz a hovorí sa mu cyklus s podmienkou na konci.

Jediná možná syntax cyklu do..while.

Príklad 14-14. Cyklus s podmienkou na konci

```
1
2 $i = 0;
3 do {
4     print $i;
5 } while ($i>0);
6
```

K príkladu **do-while** neexistuje alternatívna syntax. Pozor! Medzi Pascalom a PHP je podstatný rozdiel vo vyhodnocovaní zšvereľnej podmienky. Pascal opätovne vyhodnotí podmienku, PHP prvé naopak, pri jej nesplnení.

```
do
    i++ ;
while ( i<100 );
```

Pokiaľ **do-while** obsahuje zložený výraz, je vhodná práve podmienka **while** za uzavierajúcou zloženou zátvorkou } zloženého príkladu.

Znalci jazyka C môžu použiť aj rozdielne použitie cyklu `do..while`, ktorý umožňuje zastaviť vykonšvanie cyklu v strede bloku príkazov pomocou `do..while(0)`, a použitím príkladu [break](#). Viac asi napovie príklad:

Príklad 14-15. Prerušenie cyklu

```
1
2 do {
3     if ($i < 5) {
4         print "i nieje dost vela";
5         break;
6     }
7     $i *= $factor;
8     if ($i < $minimum_limit) {
9         break;
10    }
11    print "i je ok";
12
13    ...spracuj i...
14
15 } while(0);
16
```

for

Je príkaz cyklu, ktorý použijeme v prípade, keď je predom známy počet prechodov cyklom. Jeho syntax je nasledovná:

Príklad 14-16. Syntax cyklu s pevným počtom iterácií - for

```

1
2 for (vyraz1; vyraz2; vyraz3) prikaz
3

```

vĚraz1 je vyhodnotenĚ pred zaĤiatkom prevĚdzania cyklu. Potom je vyhodnotenĚ **vĚraz2**. Aj je vrĚtenĚ TRUE, prevĚdza sa telo cyklu **prĚkaz**. Na konci kaĤdĚho prechody cyklom sa vyhodnotĚ **vĚraz3**, Ĥo sa deje tak dlho, pokiaĤ mĚ **vĚraz2** hodnotu TRUE. Obvykle sa **vĚraz1** pouĤĚva na inicializĚciu riadiacej premennej cyklu, **vĚraz2** je podmienka prevĚdzania cyklu a **vĚraz3** slĤĤi na pravidelnĚ aktualizĚciu riadiacej premennej cyklu.

KaĤdĚ vĚraz /1,2,3/ mĚĤe byĚ prĚzdzny.

VĚpis ĤĚsel od 1 do 10:

PrĚklad 14-17. Cyklus for

```

1
2 /* PrĚklad 1 */
3
4 for ($i = 1; $i <= 10; $i++) {
5     print $i;
6 }
7
8 /* PrĚklad 2 */
9
10 for ($i = 1;;$i++) {
11     if ($i > 10) {
12         break;
13     }
14     print $i;
15 }
16
17 /* PrĚklad 3 */
18
19 $i = 1;
20 for (;;) {
21     if ($i > 10) {
22         break;
23     }
24     print $i;
25     $i++;
26 }
27
28 /* PrĚklad 4 */
29
30 for ($i = 1; $i <= 10; print $i, $i++) ;
31

```

PHP tieĤ umoĤňuje pouĤiĚ alternatĚvnu syntax:

Príklad 14-18. Alternatívna syntax cyklu for

```
1
2 for (expr1; expr2; expr3): statement; ...; endfor;
3
```

foreach

PHP4 (PHP3 nie) zahŕňa konštrukciu foreach, podobne ako perl a iné jazyky. Umožňuje jednoducho obsluhovať celé polia. Má dve syntaxe, aj keď druhá je len vedľajším rozšírením prvej:

Príklad 14-19. Syntax Foreach

```
1
2 foreach(pole as $value) prikaz
3 foreach(pole as $kluc => $hodnota) prikaz
4
```

Prvý riadok definuje cyklus v poli *pole*. Počas každého cyklu je aktuálnemu prvku poľa priradená hodnota *\$value* a index aktuálneho prvku poľa je zvýšený o jeden. Druhý riadok robí to isté, s tým rozdielom, že aktuálnemu kľuču prvku poľa bude priradená premenná *\$hodnota* počas každého priebehu cyklu.

Pri prvom volaní funkcie **foreach** je index poľa automaticky nastavený na jeho prvý prvok, takže volanie **reset()** je zbytočné.

Tieto dve konštrukcie sú teda úplne rovnaké.

Príklad 14-20. Konštrukcia Foreach 1

```
1
2 reset ($arr);
3 while (list(, $value) = each ($arr)) {
4     echo "Hodnota: $value<br>\n";
5 }
6
7 foreach ($arr as $value) {
8     echo "Hodnota: $value<br>\n";
9 }
10
```

Podobne aj tieto:

Príklad 14-21. Konštrukcia Foreach 2

```

1
2 reset ($arr);
3 while (list($key, $value) = each ($arr)) {
4     echo "Kluc: $key; Hodnota: $value<br>\n";
5 }
6
7 foreach ($arr as $key => $value) {
8     echo "Kluc: $key; Hodnota: $value<br>\n";
9 }
10

```

Pre demonštráciu použitia nasledujú 3 príklady:

Príklad 14-22. Konštrukcia Foreach 3

```

1
2 /* foreach Príklad 1: value only */
3
4 $a = array (1, 2, 3, 17);
5
6 foreach ($a as $v) {
7     print "Current value of \$a: $v.\n";
8 }
9
10 /* foreach Príklad 2: value (with key printed for illustration) */
11
12 $a = array (1, 2, 3, 17);
13
14 $i = 0; /* for illustrative purposes only */
15
16 foreach($a as $v) {
17     print "\$a[$i] => $v.\n";
18 }
19
20 /* foreach Príklad 3: key and value */
21
22 $a = array (
23     "one" => 1,
24     "two" => 2,
25     "three" => 3,
26     "seventeen" => 17
27 );
28
29 foreach($a as $k => $v) {
30     print "\$a[$k] => $v.\n";
31 }
32

```

break

break umožňuje okamžitě prerušení provádění struktury **if**, **for**, **while** alebo **switch**.

break má nepovinný numerický argument, který udává, z kolik započatých cyklů alebo iných struktur má vyskočit.

Průklad 14-23. Break v praxi

```
1
2 $i = 0;
3 while ($i < 10) {
4     if ($arr[$i] == "stop") {
5         break; /* mozem napisat aj 'break 1;'. */
6     }
7     $i++;
8 }
9
10 /* Pouzitie volitelneho parametru o kolko urovni vyskocit. */
11
12 $i = 0;
13 while ( ++$i ) {
14     switch ( $i ) {
15         case 5:
16             echo "At 5<br>\n";
17             break 1; /* Ukonci len prikaz switch. */
18         case 10:
19             echo "Pri 10; koncim>\n";
20             break 2; /* Ukonci switch aj while. */
21         default:
22             break;
23     }
24 }
25
```

continue

continue okamžite preskočí na začiatok cyklu a začne provádět další iteraci cyklu.

continue má nepovinný číselný argument, který udává, kolik cyklů sa má preskočit.

Průklad 14-24. Continue v praxi

```

1
2 while (list ($key, $value) = each ($arr)) {
3     if (!$key % 2) { // preskoc parne clen
4         continue;
5     }
6     do_something_odd ($value);
7 }
8
9 $i = 0;
10 while ($i++ < 5) {
11     echo "Vonkajsi<br>\n";
12     while (1) {
13         echo "  Stredny<br>\n";
14         while (1) {
15             echo "    Vnutorny<br>\n";
16             continue 3;
17         }
18         echo "Toto sa nikdy nevypise.<br>\n";
19     }
20     echo "Neither does this.<br>\n";
21 }
22

```

switch

V pr2pade, ěe mšme viac podmienkovŃch vetiev, oplat2 sa pouĝ2vaš pr2kaz **switch**. Jeho syntax je nasledovnš:

```

switch ( vŃraz1 )
{
    case HODNOTA A :
        pr2kazA1;
        pr2kazA2;
        .....
        break;
    case HODNOTA B :
        pr2kaz B1;
        pr2kaz B2;
        ...
        break;
    .....
    default:
        pr2kaz XXX1;
}

```

Ako to funguje? Najprv sa vyhodnot2 **vŃraz1** a potom s¼ postupne prechšdzan® hodnoty za **case**, aĝ kŃm sa nenšjde hodnota, ktorš sa rovnš **vŃrazu1**. Nšsledne sa vykonšvaj¼ pr2kazy, aĝ kŃm sa nenaraz2 na pr2kaz **break** alebo koniec pr2kazu **switch**. UpozorŤujem na to, ěe sa spracovšvaj¼ vŃetky pr2kazy (aj za **case**, pokiaŦ sa nenaraz2 na **break**). Ľasš **default** sa vykonš vtedy, ak ani jedna hodnota sa nerovnš **vŃrazu1**. Ľasš **default** je nepovinnš.

Pr2klad 14-25. KaskšdovŃ if a jeho nahradenie cez pr2kaz switch

```

1
2 if ($i == 0) {
3     print "i je rovne 0";
4 }
5 if ($i == 1) {
6     print "i je rovne 1";
7 }
8 if ($i == 2) {
9     print "i je rovne 2";
10 }
11
12 switch ($i) {
13     case 0:
14         print "i je rovne 0";
15         break;
16     case 1:
17         print "i je rovne 1";
18         break;
19     case 2:
20         print "i je rovne 2";
21         break;
22 }
23

```

require()

Príkaz **require()** doplní miesto seba obsah zadaného súboru, podobne ako v jazyku C príkaz `#include`.

Nezabudnite, že súbory spracovávané príkazom **include()** alebo **require()** sú spracovávané ako PHP dokument, čo znamená, že lasti ohraničené platňami PHP štart/stop tagom sú spracované ako PHP skript, ostatok ako HTML dokument. PHP lasš musš začaš a konšiš **platňm PHP štart/stop tagom**

Tšto **require()** nevracia žiadnu hodnotu.

Na rozdiel od **include()**, **require()** vğdy ľta cieľovľ sľbor, aj keď sa prľsluğňľ riadok kľdu vo Vağom skripte nikdy nevykonšva. Pre podmienenľ includovanie sľboru pouğite radğej **include()**.

Podobne, cykly takisto neovplyvľujľ spršvanie **require()**. Ak je príkaz **require()** sľlasš cyklu, includovanie sľboru sa vykonš len raz!

Ak chcete includovaš rľzne sľbory v cykle, pouğite **include()**.

Príkklad 14-26. Zavolanie hlaviľky

```

1
2 require ('header.inc');
3

```

Nezabudnite, že **include()** aj **require()** zšskajľ obsah sľboru vlastnou cestou cez lokšlny sľborovľ systľm, nie pomocou HTTP .

Príklad 14-27. Pripojenie súboru

```
1
2 require ("file.php?varone=1&vartwo=2"); /* nepracuje. */
3
4 $varone = 1;
5 $vartwo = 2;
6 require ("file.php"); /* $varone a $vartwo budú k dispozícii v file.php */
7
```

15. Funkcie

Užívateľsky definované funkcie

Funkcie slúžia na zjednodušenie našich programov. V zhrňme skrátenia a spehľadnenia kódu delíme bloky kódu do jednoduchých častí, ktoré nesú názov podľa funkcie ktorú vykonávajú. Tieto programové bloky voláme procedúry alebo funkcie. Názov procedúry alebo funkcie by mal výstižne popisovať to, čo funkcia vykonáva. Funkcie môžu volať s viacerými vstupnými parametrami, funkcia vracia jednu návratovú hodnotu. Príkladom môže byť funkcia pre sčítanie dvoch čísel, ktorá vracia výsledok:

Príklad 15-1. Sčítanie dvoch čísel

```
1
2 function scitaj ($scitanec_1, $scitanec_2) {
3     $vysledok = $scitanec_1 + $scitanec_2;
4     return $vysledok;
5 }
6
7 //príklad použitia
8 $aritmetika = scitaj(5,5);
9 echo "Vysledok je".$aritmetika;
10 //skratene
11 echo "Skrateny vysledok je".scitaj(5,5);
```

Platí kód musí byť telom funkcie, podobne ako pri iných funkciách alebo triedach. Telo funkcie začína a končí blokovými operátormi { a }.

Argumenty funkcií

Informácie pre funkciu môžu byť dodané ako zoznam argumentov, ktorý je vlastne zoznamom premenných alebo konštánt oddelených čiarkami.

PHP podporuje volanie argumentov hodnotou (implicitne), volanie odkazom, a **implicitné hodnoty argumentov**. Zoznam argumentov s premenlivou dĺžkou je podporovaný len v PHP 4.xx. Pre viac informácií pozri **zoznam argumentov s premenlivou dĺžkou** a popis funkcií **func_num_args()**, **func_get_arg()** a **func_get_args()**. Rovnaký efekt môžu dosiahnuť aj v PHP 3 predaním podľa ako argumentu.

Príklad 15-2. Funkcia s argumentom typu pole

```
1
2 function vezme_pole($vstup) {
3     echo "$vstup[0] + $vstup[1] = ", $vstup[0]+$vstup[1];
4 }
5
6 $vstup = array ('jano', 'Fero');
7 vezme_pole ($vstup);
```

Predávanie argumentov odkazom

Implicitne sú argumenty funkcie volané hodnotou, čo v skratke znamená, že ak zmeníme hodnotu argumentu vo vnútri funkcie, neovplyvní to hodnotu argumentu mimo nej. V opačnom prípade musíte použiť volanie funkcie odkazom. Ak chcete, aby bol argument funkcie volaný odkazom, musíte pred ním dať znak **ampersand (&)**.

Príklad 15-3. Volanie odkazom

```
1
2 function pridaj(&$string) {
3     $string .= 'a niečo extra';
4 }
5 $str = 'Toto je rečazec ';
6 pridaj($str);
7 echo $str;    // vypíše 'Toto je rečazec a niečo extra'
8
```

Ak chcete volať aj normálnu funkciu (volanú hodnotou) odkazom, musíte pripojiť znak ampersand pred jej názov pri jej volaní.

Príklad 15-4. Volanie odkazom

```
1
2 function foo ($bar) {
3     $bar .= ' a niečo extra.';
4 }
5 $str = 'Toto je rečazec, ';
6 foo ($str);
7 echo $str;    // vypíše 'Toto je rečazec'
8 foo (&$str);
9 echo $str;    // vypíše 'Toto je rečazec a niečo extra'
10
```

Implicitné hodnoty argumentov

Funkcia môže použiť implicitnú hodnotu argumentov v prípade, že argument pri volaní neinicializujeme (nedodáme hodnotu). V prípade, že varíme kšvu v reštaurácii /pomocou funkcie urobKavu/, predpokladáme, že 95% hostí si objedná kapuľno. To umožní zjednodušiť funkciu urobKavu tak, že ak nezadáme iný vstupný argument - iný typ kšvy, tak nám implicitne uvarú kapuľno.

Príklad 15-5. Funkcia urobKavu s implicitným argumentom

```
1
2 function urobKavu ($type = "cappuccino") {
3     return "Urob salku kavy $type.\n";
4 }
5 echo urobKavu ();
6 echo urobKavu ("espresso");
7
```

Výstup bude nasledujúci

```
Urob salku kavy cappuccino.
Urob salku kavy espresso.
```

Implicitná hodnota musí byť konštanta, nemôže to byť premenná alebo člen triedy. Znamená to, že v tejto definícii funkcie sa implicitná hodnota môže definovať len raz. **Nezabudnite, že implicitné argumenty môžu byť zaradené aj za "klasickými" argumentmi, teda vždy v pravo v zozname argumentov.** Nasledujúci príklad nebude preto fungovať.

Príklad 15-6. Chybny implicitný argument

```
1
2 function urobJogurt ($type = "probio", $prichut) {
3     return "Urob pohar jogurtu typu $type $flavour.\n";
4 }
5
6 echo urobJogurt ("jahoda");    // nefunguje
7
```

Výpise

```
Warning: Missing argument 2 in call to urobJogurt() in
xxx.php on line 6
Urob pohar jogurtu typu probio jahoda
```

A takto je to správne:

Príklad 15-7. Správny implicitný argument

```

1
2 function urobJogurt ($prichut, $type = "probio") {
3     return "Urob pohar jogurtu $type $flavour.\n";
4 }
5
6 echo urobJogurt ("jahoda");    // funguje
7

```

Výstup:

```
Urob pohar jogurtu probio jahoda.
```

Návratové hodnoty

Návratová hodnota funkcie je inicializovaná nepovinným príkazom **return**. Vrstení môže byť hocaký typ, vrstane zoznamu či objektu.

Príklad 15-8. Návratová hodnota

```

1
2 function mocnina ($num) {
3     return $num * $num;
4 }
5 echo mocnina (4);    // vypíše '16'.
6

```

Aj keď v PHP môžete získať viac premenných z funkcie jednotlivo, ale môžete ich vrátiť pomocou poľa.

Príklad 15-9. Viac návratových hodnôt

```

1
2 function male_cisla() {
3     return array (0, 1, 2);
4 }
5 list ($nula, $jedna, $dve) = male_cisla();
6

```

Funkcie premenných

PHP podporuje aj názvy funkcií premenných. Znamená to, že názov premennej je použitý ako funkčný volanie funkcie.

Príklad 15-10. Funkcia premennej

```
1
2 <?php
3 function foo() {
4     echo "Vo funkcii foo()<br>\n";
5 }
6
7 function bar( $arg = '' ) {
8     echo "Vo funkcii bar(); argument bol '$arg'.<br>\n";
9 }
10
11 $func = 'foo';
12 $func();
13 $func = 'bar';
14 $func( 'test' );
15 ?>
16
```

16. Triedy a objekty

Prívrženci OOP (objektovo orientovaného programovania) sa doteraz určite len pousmievali a zabývali sa nad týmto aprízemnosťami. Mäť tu však aj niečo pre nich. Jazyk PHP prináša možnosť definície tried. Trieda je určitý typ, ktorý obsahuje dšta a dštové štruktúry (ale aj nemusí) a funkcie (metódy), ktoré sú nad danými dštovými štruktúrami definované. Inštancia triedy v programe sa nazíva objekt. Objekt je teda fyz. štruktúra v pamäti počítača, pričom trieda je len akási predpis, ako objekt vytvoriť. Pomocou tried a objektov dosiahneme väčšiu istotu a bezpečnosť programu, nakoľko to vychádza z objektovo orientovaného programovania a jeho troch základných vlastností:

- dedičnosti
- polymorfizmu
- zapízdrenosti.

Trieda má v jazyku PHP nasledovnú šchematickú tvar:

```
class meno_triedy
{
    definícia premenných [ = inicializačná hodnota ] ;
    definícia funkcií ;
}
```

Trieda je súhrn premenných a funkcií, ktoré s týmto premennými pracujú. Triedu definujeme pomocou nasledujúcej syntaxe:

Príklad 16-1. Definícia triedy s 2 metodami

```

1
2 <?php
3 class Kosik {
4     var $polozky; // polozky v nakupnom kosiku
5
6     // pridaj $pocet poloziek tovaru $cislo_vyrobu do kosika
7
8     function pridaj_do_kosika ($cislo_vyrobu, $pocet) {
9         $this->polozky[$cislo_vyrobu] += $pocet;
10    }
11
12    // zobor $pocet kusov tovaru $cislo_vyrobu z kosika
13
14    function vyber_z_kosika ($cislo_vyrobu, $pocet) {
15        if ($this->polozky[$cislo_vyrobu] > $pocet) {
16            $this->polozky[$cislo_vyrobu] -= $pocet;
17            return true;
18        } else {
19            return false;
20        }
21    }
22 }
23 ?>
24

```

Toto definuje triedu s názvom Kosik, ktorá pozostáva z asociatívneho poľa tovarov a dvoch funkcií - na pridávanie do košíka a vyberanie z košíka.

Triedy sú typy, čo znamená, že ak chceme vytvoriť premennú (inštanciu) triedy, musíme použiť operátor **new**.

Príklad 16-2. Volanie triedy

```

1
2 $kosik = new Kosik;
3 $kosik->pridaj_polozku("10", 1);
4

```

Ten vytvorí objekt \$kosik triedy Kosik. Potom je zavolaná funkcia pridaj_do_kosika() pre vloženie 1 kusu tovaru číslo 10 do košíka.

Triedy môžu byť tiež rozšíreniami iných tried. Rozšírená alebo odvodená trieda má všetky premenné a funkcie pôvodnej triedy. Je nutné použiť kľúčové slovo **extended**. Viacnásobné dedenie nie je zatiaľ povolené, povolené sú len vnorené triedy.

Príklad 16-3. Definícia triedy Upraveny_Kosik

```

1
2 class Upraveny_Kosik extends Kosik {
3     var $vlastnik;
4
5     function nastav_vlastnika ($meno) {
6         $this->vlastnik = $meno;
7     }
8 }
9

```

Tento príklad definoval triedu Upraveny_Kosik ktorá má všetky premenné a funkcie triedy Kosik plus dodatočnú premennú \$vlastnik a dodatočnú funkciu nastav_vlastnika(). Vytvoríte pomenovaný košík klasickou cestou a môžete nastaviť a získať názov vlastníka.

Príklad 16-4. Použitie triedy Upraveny_Kosik

```

1
2 $novy_kosik = new Upraveny_Kosik;    // vytvorí pomenovaný košík
3 $novy_kosik->nastav_vlastnika ("kris"); // nastaví vlastníka
4 print $novy_kosik->vlastnik;         // vypíše vlastníka
5 $novy_kosik->pridaj_polozku ("10", 1); // (zdedená vlastnosť od triedy cart)
6

```

Vo funkcii triedy premenná \$this značí tento objekt. Musíte použiť \$this-> niečo pre prístup ku premenným vo Vašom objekte.

Konštruktor je funkcia v triede, ktorá je automaticky zavolaná pri vytvorení inštancie objektu. Už jadro Zend 1.0 zaviedlo známu objektovú metódu konštruktor, ktorá je volaná pri vytvorení objektu. Jej opak však v tejto verzii jadra chýbal, ale novšia verzia Zend 2.0 prináša podporu deštruktora, čo je metóda volaná pri zrušení objektu. Deštruktor je automaticky volaný pri zrušení poslednej referencie na objekt. V predchádzajúcej verzii mal konštruktor meno danej triedy, v jadre 2.0 sa volá **__construct()**. Podobne má deštruktor meno **__destruct()**.

Príklad 16-5. Definícia triedy pre Zend 1 a Zend 2

```

1
2 class MojaTriedaZend1 {
3     function MojaTriedaZend1() { //stary typ konštruktora
4         print "Konštruktor pre Zend 1.0";
5         $this->meno = "MojaTriedaZend1";
6     }
7 }//Zend 1.0
8
9 class MojaTriedaZend2 {
10     function __construct() {
11         print "Konštruktor\n";
12         $this->meno = 'MojaTriedaZend2';
13     }//nový konštruktor
14     function __destruct() {
15         print 'Zrusim ' . $this->meno . "\n";
16     }//nový deštruktor
17 }//Zend 2.0
18

```

Podobne ako konštruktor ani deštruktor nevolš implicitne svoj rodišovskš konštruktor alebo deštruktor. V pr²pade, ģe chcete zavolaš aj rodišovskš met²du, pouĝite konštrukciu vo Vaĝom k²de:

```
    b rodišovskš_objekt::__construct()
```

alebo

```
    b rodišovskš_objekt::__destruct()
```

Zend 1.0 neposkytuje ĝiadnu moĝnosš odstršnenia objektu, na ktorš ģte existuje referencia. Novozavedenš pr²kaz **delete** umoĝnuje n²tene zavolaš deštruktor objektu a uvošniš objekt z pam²te. Referencie, ktor² odkazovali na tento objekt bud² neplatn² a pokus o pr²stup k objektu bude sprevšdzanš kritickou chybou.

Ak mšte uĝivatešsk² funkciu **delete()**, bude ju nutn² premenovaš, nakoško toto kšlov² slovo je rezervovan² pre n²ten² volanie deštruktora. Pre zmeny v jadre ZEND 2 a novinky v objektovom modeli PHP 4.2 a vyĝĝie si preš²tajte dokument [Zend 2 - Zmeny v PHP](#)

17. PHP a Grafika

Knšnica GD umoĝnuje v spolupršci s PHP šahko generovaš dynamick² obršzky a v spolupršci s inšmi knšnicami aj grafy. Sp²sob pouĝitia knšnice GD na dynamick² generovanie oršzkov je vešmi jednoduch².

Zškladnou podmienkou pre funkšnosš knšnice GD je jej aktivšcia pri konfiguršcii PHP. Ak pouĝ²vame syst²m PHP Home 2, staš² povoliš v s²bore php.ini extension=php_gd.dll:

V s²bore php.ini upravte šasš extension na:

```
extension_dir = "C:\Program Files\PHP Home Edition 2\Apache2\php\extensions"
...
;extension=php_fdf.dll
;extension=php_filepro.dll
extension=php_gd.dll
```

V pr²pade rušnej kompilšcie je nutn² PHP skompilovaš napr²klad s ššmito vošbami:

```
./configure --with-gd --with-ttf --sysconfdir=/etc --disable-debug --enable-track-vars --with-pdflib=/usr/local/pdflib --with-jpeg-dir --with-zlib-dir --enable-gd-native-ttf --with-tiff-dir --enable-ftp --enable-sockets --with-ttf --with-freetype
```

respekt²ve trochu okresane:

```
./configure --with-gd --with-ttf --disable-debug --enable-track-vars --with-jpeg-dir --with-zlib-dir --enable-gd-native-ttf --with-tiff-dir
```

Potrebn² knšnice je nutn² na syst²m nainštalovaš napr²klad cez pr²kaz apt get install <nazov kniznice> prišom inštalujeme nielen zškladn² verziu knšnice, ale aj jej všvojšrsku verziu konšiacu na dev. Nšzov knšnice zšskame na www.debian.org v sekcii Packages, dšme all Packages. Pr²klad inštalšcie knšnice GD:

```
# apt-get install libgd2
# apt-get install libgd2-dev
```

Príklad 17-1. Subor obrazok.php

```
1
2 <?php
3 Header("Content-type: image/png");
4 $string=implode($argv, " ");
5 $im = ImageCreateFromPng("images/button_pozadie.gif");
6 $orange = ImageColorAllocate($im, 220, 210, 60);
7 $px = (imagesx($im)-7.5*strlen($string))/2;
8 ImageString($im,3,$px,9,$string,$orange);
9 ImagePng($im);
10 ImageDestroy($im);
11 ?>
```

Knihnica GD umožňuje generovať obrázky typu JPG, PNG a staršie verzie knihnice aj GIF. Podpora posledného formátu bola ukončená vzhľadom na patentové problémy, je však možné knihnicu získať a skompilovať vlastnými silami.

Použitie predchádzajúceho príkladu je potom veľmi jednoduché, príkladom je tzv. dynamicky generované menu:

Príklad 17-2. Použitie dynamicky generovaného obrázka

```
<!-- menu -->
<table border="0">
<tr>
  <td><a href="auta.html"></a></td>
  <td><a href="vlaky.html"></a></td>
  <td><a href="lietadla.html"></a></td>
  <td><a href="bicykle.html"></a></td>
</tr></table>
<!-- koniec menu -->
```

18. PHP a databázy

Pre správu databázového systému slúžia špecializované nástroje. Viac informácií o relačných a objektových databázach je možné nájsť v príslušnej literatúre. Taktiež je vhodné rozumieť správe databáz, resp. poriadne si preštudovať kapitolu číslo 6.

V nasledujúcich príkladoch bude pre správu DBS použitý nástroj phpMyAdmin a databáza MySQL.

- V hlavnom menu vytvoríme databázu **test**
- Po úspešnom vytvorení databázy test ju vyberieme v ľavom selectboxe a klikneme na záložku SQL, kde nakopírujeme dole uvedený text do poľa **Spustiť SQL dotaz/dotazy na databázu test** a klikneme na tlačidlo **Vykonať**. Rovnaký efekt dostaneme ručnou definíciou tabuliek, resp. importom súboru test.sql

- Nastavíme prŕva pre prŕstup k databŕze test pre webovŕho uŕivateĽa (pre viac informŕciŕ pozri manuŕl k MySQL) a reloadneme databŕzu v hlavnom menu: (**Domov->Znovu-naĽtŕŕ MySQL**). Vhodnŕ prŕva pre naŕu aplikŕciu sŕ SELECT,INSERT,UPDATE,DELETE

Po tŕchto krokoch je databŕza pripravenŕ k pouŕitiu.

```
# subor test.sql
# phpMyAdmin MySQL-Dump
# http://phpwizard.net/phpMyAdmin/
#
# HostiteĽ: localhost Databŕza : test

# -----
#
# Ľtruktŕra tabulky pre tabulku 'predmet'
#

USE DATABASE test;

CREATE TABLE predmet (
  cp int(4) NOT NULL auto_increment,
  nazov varchar(30) NOT NULL,
  kredity tinyint(4) DEFAULT '0' NOT NULL,
  PRIMARY KEY (cp)
);

#
# Dumping dŕŕt pre tabuĽku 'predmet'
#

INSERT INTO predmet VALUES ( '1', 'Matematika', '4');
INSERT INTO predmet VALUES ( '2', 'Fyzika', '8');
INSERT INTO predmet VALUES ( '3', 'Operacna analyza', '100');
INSERT INTO predmet VALUES ( '4', 'Psychologia', '10');
INSERT INTO predmet VALUES ( '5', 'Malovanie', '3');

# -----
#
# Ľtruktŕra tabulky pre tabulku 'student'
#

CREATE TABLE student (
  id_st int(11) NOT NULL auto_increment,
  meno varchar(30) NOT NULL,
  priezvisko varchar(30) NOT NULL,
  rc varchar(11) NOT NULL,
  adresa varchar(50) NOT NULL,
  stav char(1) DEFAULT 's',
  PRIMARY KEY (id_st)
);

#
# Dumping dŕŕt pre tabuĽku 'student'
#
```

```

INSERT INTO student VALUES ( '1', 'Milan', 'Gacik', '7701116236', '191', '');
INSERT INTO student VALUES ( '2', 'Janko', 'Hraško', '7688252222', '515', '');
INSERT INTO student VALUES ( '3', 'Janka', 'Nezabudkova', '7999223333', '454', 's');
INSERT INTO student VALUES ( '4', 'Janko', 'Luptak', '7810102222', 'Fazulova 25', 's');
INSERT INTO student VALUES ( '5', 'Ferkova', 'Bozenkova', '48489488484', 'Nevadzova 25', 's');

# -----
#
# Ľuktra tabulky pre tabulku 'zap_pred'
#

CREATE TABLE zap_pred (
id_st int(11) DEFAULT '0' NOT NULL,
cp int(11) DEFAULT '0' NOT NULL,
PRIMARY KEY (id_st, cp)
);

#
# Dumping dát pre tabuľku 'zap_pred'
#

INSERT INTO zap_pred VALUES ( '1', '1');
INSERT INTO zap_pred VALUES ( '1', '2');
INSERT INTO zap_pred VALUES ( '2', '2');

```

Príklad 18-1. Vypis všetkých údajov z tabuľky predmet

```

1 <?php
2 do {
3     $conn = Mysql_Connect("localhost", "uzivatel", "heslo"); //tu nastav
4     if (!$conn){
5         echo "Nepodarilo sa pripojiť k databaze";
6         break;
7     }//endif
8     Mysql_Select_DB("test");
9     $vysledok = @mysql_query("SELECT * FROM predmet");
10    if (!$vysledok) {
11        echo "Chybny dopyt SQL";
12        break;
13    }//endif;
14    echo "<table border='1'";
15    while($riadok = mysql_fetch_array($vysledok)) {
16        echo "<tr>\n";
17        echo "<td>".$riadok["cp"]."</td>";
18        echo "<td>".$riadok["nazov"]."</td>";
19        echo "<td>".$riadok["kredity"]."</td>";
20        echo "</tr>\n";
21    } // endwhile
22    echo "</table>";
23    } while (false);
24 ?>

```

Niekedy je vhodné prepísať viac tabuliek, tento typ výpisu sa volá MASTER-DETAIL výpis:

Príklad 18-2. Master-detail výpis

```
<?php
    Header("Expires: ".GMMDate("D, d M Y H:i:s")." GMT");
?>
<h2>Vypis studentov s predmetmi</h2>
<?php
    function showForm() {
        //vypise formularik
        do {
            $conn = @Mysql_Connect("localhost","root","root");
            if (!$conn){
                echo "Nepodarilo sa pripojit k databaze";
                break;
            } // endif;
            Mysql_Select_DB("test");
            $query = "SELECT priezvisko FROM student ";
            $vysledok = @mysql_query($query);
            if (!$vysledok) {
                echo "Chybny dopyt";
                break;
            } //endif;
        } //endif;
    }

    <form action="" method="post">
    <select name="priezvisko">

    <?php
        while($riadok = mysql_fetch_array($vysledok)) {
            echo "<OPTION VALUE=\"". $riadok["priezvisko"]." \">";
            echo $riadok["priezvisko"]." \"\n";
        } //endwhile
    } while (false);
    ?>

    </select>
    <input type="submit" name="odosli" value="Odosli">
    </form>

    <?php
    } //end show form

    function spracuj() {
        global $priezvisko;
    }

    <table border="1">
    <?
    do {
        $conn = @Mysql_Connect("localhost","root","root");
        if (!$conn) {
            echo "Nepodarilo sa pripojit k databaze";
            break;
        }
    }
}
```

```

} // endif;
Mysql_Select_DB("test");
if (!isset($tried)) $tried="ASC";
$query="SELECT nazov,kredity FROM predmet,student,zap_pred WHERE priezvisko='$priezvisko'
        AND student.id_st=zap_pred.id_st AND zap_pred.cp=predmet.cp ";
// echo $query; //<!-- ak testujes, vyremuj komentar
$vysledok = @mysql_query($query);
if (!$vysledok) {
    echo "Chybny dopyt";
    break;
} //endif;
while($riadok = mysql_fetch_array($vysledok)) {
    echo "<tr>\n";
    echo "<td>".$riadok["nazov"]."</td>";
    echo "<td>".$riadok["kredity"]."</td>";
    echo "</tr>\n";
} //endwhile
} while (false);
echo "</table>";
}
?>
<div align="center"><a href="index.php">INDEX</a></div>

<?php
//end function Spracuj
if (!isset($priezvisko)) showForm();
else spracuj();

?>

```

19. Záver a PHP Klub

Všetky potrebné informácie je možné získať aj na doleuvedených odkazoch:

- www.php.sk
- www.zend.com
- www.mysql.com
- www.apache.org
- www.phpbuilder.de

Vďaka PHP Vám pomôže manuál a rôzne informatívne materiály, ktoré je možné získať na internete. Ak máte záujem o ich zaslanie, je možné si ich objednať na našej stránke spolu s členstvom v našom PHP klube na stránke

<http://www.php.sk/registracia.php>

Finančným príspevkom za členstvo /jednorázovo/ prispievate na prevádzku slovenského servera www.php.sk venovaného problematike jazyka PHP a vývoja webových aplikácií. Získate dve CD plné software pre vývoj PHP a XML aplikácií a prístup k tutorialom pre členov PHP klubu.

