Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

ŠTÁTNY PEDAGOGICKÝ ÚSTAV
NATIONAL INSTITUTE FOR EDUCATION

Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Programovanie 5

Predmet: Programovanie

Línia: Vlastný odborový kontext informatiky a informatickej výchovy

Operačný program VZDELÁVANIE

EURÓPSKA ÚNIA
Európsky sociálny fond

Európska únia
Európsky sociálny fond

Moderné vzdelávanie pre vedomostnú spoločnosť/Projekt je spolufinancovaný zo zdrojov ES

Základy programovania v jazyku PHP

Garant predmetu:

Mgr. Ján Guniš
ODIPT ÚINF PF UPJŠ v Košiciach
jan.gunis@upjs.sk

Autori:

Mgr. Ján Guniš
ODIPT ÚINF PF UPJŠ v Košiciach

Identifikácia modulu

Aktivita projektu:	1.3 Ďalšie vzdelávanie kvalifikovaných učiteľov informatiky na 2. stupni ZŠ a na SŠ
Línia aktivity:	Vlastný odborový kontext informatiky a informatickej výchovy
Predmet:	Programovanie

Zaradenie modulu

<table>
<tr><td colspan="5">Línia: Vlastný odborový kontext informatiky a informatickej výchovy</td></tr>
<tr><td colspan="5">Predmet: Programovanie</td></tr>
<tr><td colspan="3"></td><td>alternatíva</td><td>alternatíva</td></tr>
<tr><td rowspan="3">Programovanie 1 (3Prog1)</td><td rowspan="3">Programovanie 2 (3Prog2)</td><td rowspan="3">Programovanie 3 (3Prog2)</td><td rowspan="2">Programovanie 4 (Delphi) (3Prog4A)</td><td>Programovanie 5 (Java) (3Prog5A)</td></tr>
<tr><td rowspan="2">Programovanie 5 (Python) (3Prog5B)</td></tr>
<tr><td>Programovanie 4 (Imagine) (3Prog4B)</td><td>Programovanie 5 (PHP) (3Prog5C)</td></tr>
</table>

Tento modul je posledným, piatym modulom predmetu Programovanie. Účastníci aktivity 1.3 si ho môžu vybrať ako jednu z troch ponúkaných alternatív (Java, Python, PHP).

Abstrakt modulu

Účastníci vzdelávania získajú základný prehľad o možnostiach programovania v jazyku PHP. Oboznámia sa so špecifikami tohto skriptovacieho jazyka a naučia sa vytvárať jednoduché dynamické webové aplikácie s ohľadom na bezpečnosť. Získajú prehľad o typoch úloh a problémoch, ktoré je možné riešiť v jazyku PHP.

Obsah

Úvod

http://www.php.net/

V module 3Prog5C sa oboznámime so základmi programovania v skriptovacom jazyku PHP. Jazyk PHP bol už od začiatku vyvíjaný ako nástroj pre tvorbu dynamických webových stránok. V súčasnosti patrí v tejto oblasti použitia medzi najpoužívanejšie nástroje.

Okrem samotného interpretera jazyka PHP budeme potrebovať editor PHP skriptov a HTTP server.

LMS Moodle, slobodná encyklopédia Wikipédia a množstvo ďalších populárnych projektov využíva skriptovací jazyk PHP.

Cieľ modulu

Získať prehľad o možnostiach využitia skriptovacieho jazyka PHP pri tvorbe dynamických webových stránok. Poznať princíp technológií na strane servera, rozumieť výhodám a nevýhodám týchto technológií. Vedieť inštalovať a konfigurovať HTTP server s podporou PHP (príp. niektorý z balíkov typu LAMP, WAMP). Naučiť sa vytvárať jednoduché webové aplikácie v jazyku PHP. Vedieť používať premenné a základné štruktúrované príkazy jazyka PHP. Vedieť vytvárať aplikácie s použitím formulárových prvkov. Naučiť sa využívať súbory na serveri na ukladanie a čítanie dát. Vytvárať a používať funkcie ako nástroj pre sprehľadnenie a zefektívnenie programového kódu. Poznať základné bezpečnostné riziká dynamických webových aplikácií a vedieť ich eliminovať.

Vstupné vedomosti

Požadované prerekvizity

Hlbšia digitálna gramotnosť učiteľa informatiky (3DG2),

Programovanie 1, (3Prog1),

Programovanie 2, (3Prog2),

Programovanie 3, (3Prog2),

Programovanie 4 (Delphi) (3Prog4A) alebo Programovanie 4 (Imagine) (3Prog4B).

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

Účastník vzdelávania:

- pozná a správne používa elementy značkovacieho jazyka (X)HTML,
- vie vytvárať (X)HTML dokumenty priamou editáciou (X)HTML kódu,
- rozumie princípom komunikácie prehliadača webových stránok s HTTP serverom,
- pozná niektorý z programovacích jazykov,
- pozná a vie používať základné programové konštrukcie,
- pozná a vie používať cykly, podmienky vetvenia, zložené príkazy,
- vie navrhovať a vytvárať programy riešiace konkrétne problémy.

Preverenie vstupných vedomostí

Úspešné absolvovanie modulu 3DG2 a prvých štyroch modulov predmetu Programovanie, blesková diagnostika vstupných vedomostí lektorom.

Základy programovania v jazyku PHP

Študijný materiál je rozdelený do 9 častí, ktoré na seba logicky nadväzujú. V častiach 1 až 6 sa budeme venovať:

- inštalácii HTTP servera, spôsobu písania PHP skriptov,
- syntaxi jazyka PHP, práci s dátumom a časom a ich formátovaním,
- premenným, práci s premennými a spôsobom inicializácie premenných,
- podmieneným príkazom a zloženému príkazu,
- poliam a príkazom opakovania,
- spracovaniu dát z (X)HTML formulárov,

a budú realizované prezenčnou formou. V častiach 7 až 9 sa budeme venovať:

- práci so súbormi,
- vytváraniu vlastných funkcií,
- bezpečnosťou aplikácií založených na PHP,

a budú realizované dištančnou formou.

Kapitola 1: Stručná história PHP, inštalácia PHP a HTTP servera, prvé skripty

Dynamické webové stránky

Každý tvorca webových stránok začne byť časom obmedzovaný statickosťou (X)HTML formátu. (X)HTML v podstate neposkytuje žiadnu dynamickosť. (X)HTML stránka sa každému návštevníkovi a v každom čase zobrazí rovnako. Ak potrebujeme docieliť zmenu stránky, nezostáva nám nič iné len editovať zdrojový kód stránky a jej zmenenú podobu opäť uložiť na server. Pokiaľ sa náš web často nemení, nie je to až taký problém.

Čo ale v prípade, ak potrebujeme, aby sa naša stránka menila napr. v závislosti od času? Ukážkou môže byť stránka reštaurácie: ak si ju návštevník pozrie ráno, zobrazí sa mu ponuka raňajok, v čase obeda uvidí na tej istej adrese ponuku hlavného menu a neskoro popoludní mu stránka ponúkne program zábavného večera. Predstava, že niekto je poverený pravidelne, niekoľkokrát denne meniť obsah stránky, je ťažko uveriteľná. Naozaj, takto to v praxi nefunguje.

Na výber máme z niekoľkých riešení - technológií. Obhospodarovanie týchto technológií môže byť v kompetencii servera, hovoríme im technológie spúšťané na strane servera, alebo v kompetencii klienta, a vtedy sa im hovorí technológie na strane klienta.

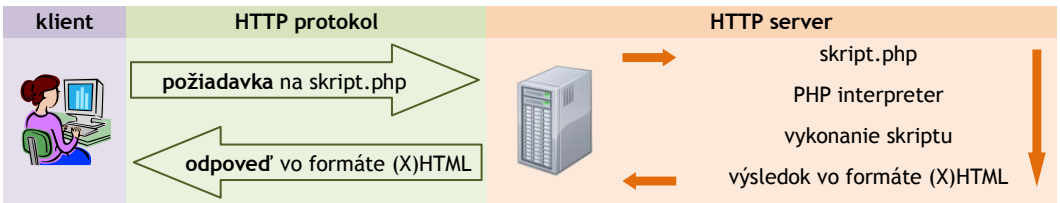
Každá z technológií, či už na strane servera alebo klienta, má svoj význam a využitie. Pri prevádzkovaní väčších webov sa väčšinou stretneme s kombináciou oboch. Časť problému sa vyrieši jednoduchšie na strane servera, iná časť má efektívnejšie riešenie na strane klienta. PHP zaraďujeme k technológiám na strane servera.

Ďalšie technológie na strane servera:

- **SSI** (Server Side Includes) - jednoduché príkazy vložené priamo do (X)HTML kódu ako komentár; súbor má špeciálnu príponu (shtml), takže server vie, že pred odoslaním danej stránky má previesť všetky SSI príkazy,
- **SSJS** (Server Side JavaScript) - implementácia JavaScriptu na strane servera,
- **ASP** (Active Server Pages) - technológia Microsoftu vyvinutá ako reakcia na vyššie uvedené technológie (zvlášť na SSJS), je ale použiteľná len na platforme Windows.

Technológie na strane klienta:

- dynamické XHTML,
- JavaScript,
- Java Applet,
- Adobe Flash,
- Imagine Applet
- AJAX (Asynchronous JavaScript and XML).



Obrázok 1: Princíp fungovania technológie na strane servera (PHP)

Zadanie 1.1	Na webový stránkach sa často stretávame so zobrazením aktuálneho času a dátumu. Ktorá z technológií je na to vhodnejšia, technológia na strane servera alebo technológia na strane klienta? Prečo?

Stručná história PHP

PHP je nástupcom produktu PHP/FI (Personal Home Page/Forms Interpreter), ktorý vytvoril Rasmus Lerdorf v roku 1995. PHP/FI predstavoval jednoduchú sadu skriptov v jazyku Perl pre spracovanie záznamov o prístupoch k jeho webu. Keďže časté spúšťanie Perl skriptov značne zaťažovalo server, prepísal autor celý svoj systém do jazyka C. Systém sa stal obľúbeným a nachádzal si stále viac používateľov. Autor celý systém rozšíril a doplnil o dokumentáciu. Výsledný produkt nazval "Personal Home Page Tools". Systém neskôr dokázal komunikovať s databázami, a tak umožňoval používateľom vyvíjať jednoduché dynamické aplikácie pre web.

Rasmus sa rozhodol uvoľniť zdrojové kódy PHP/FI, takže systém mohol ktokoľvek nielen používať, ale aj opravovať chyby a vylepšovať kód. Tento systém už obsahoval základ funkcionality PHP tak, ako ho poznáme dnes.

V roku 1998 prišla verzia PHP 3, ktorá bola omnoho rýchlejšia a obsahovala množstvo nových funkcií. Na rozdiel od predchádzajúcej verzie dokázalo PHP 3 bežať aj na systémoch MS Windows.

PHP je skriptovací jazyk - jeho skripty sa vkladajú do (X)HTML kódu. PHP je interpreter - pri každom spustení skriptu je jeho zdrojový kód prevádzaný (vykonávaný) príkaz po príkaze do spustiteľného kódu. Neexistuje teda žiadna binárna verzia zdrojových súborov PHP skriptov.

Inštalácia PHP interpretera a HTTP servera

PHP je distribuované v licencii Open Source. Môžeme si ho teda slobodne inštalovať a používať. Pre plnohodnotné programovanie v PHP však samotný interpreter nestačí. Chýba nám prostredník, ktorý by doručil našu požiadavku (samotný skript) PHP interpreteru, a zároveň nám vrátil výsledok behu skriptu (vo formáte (X)HTML). Týmto prostredníkom je HTTP server. Aj keď si môžeme vybrať zo širokej ponuky HTTP serverov, PHP si najlepšie „rozumie“ s HTTP serverom Apache. Aj licencia servera Apache (Apache License, Version 2.0) nám dovoľuje softvér bezplatne prevziať a používať.

Pre vzájomnú spoluprácu PHP a HTTP servera je potrebná správna konfigurácia oboch produktov. Existuje pohodlné a rýchle riešenie. Na internete nájdeme kompletné balíčky obsahujúce PHP aj HTTP server Apache, ktoré sú nakonfigurované pre vzájomnú spoluprácu. Tieto balíčky obsahujú aj ďalšie užitočné súčasti (napr. systém riadenia bázy dát MySQL) a sú určené pre rôzne operačné systémy. Pre potreby tohto modulu budeme používať LAMP balíček „The Uniform Server“ (http://www.uniformserver.com/).

Program nie je nutné inštalovať. Obsah archívu stačí extrahovať do priečinka na disku počítača. Názov priečinka vrátane cesty by nemal obsahovať biele znaky (napr. medzeru) a znaky s diakritikou.

V rozbalenom priečinku nájdeme súbor *Start.exe*. Po jeho spustení sa v oznamovacej oblasti na hlavnom paneli systému objaví ikonka  . Kliknutím na túto ikonku sa zobrazí ponuka nástrojov pre správu činnosti a behu inštalovaného serveru UniTray. Pre samotné spustenie servera vyberieme ponuku „Start Uniserver (Apache MySQL)“.

Po naštartovaní servera sa automaticky spustí prehliadač webových stránok, v ktorom sa zobrazí webová stránka na adrese: http://localhost/index.php. Obsahom stránky sú informácie o obsahu balíčka Uniform Servera. Zároveň sa

V čase vzniku tohto materiálu bola najnovšou verziou jazyka PHP verzia PHP 5.2.12.

Skratka PHP pomaly stráca svoj pôvodný zmysel a dnes sa skôr interpretuje ako rekurzívny akronym "PHP: Hypertext Preprocessor".

Skript (angl. script) predstavuje zdrojový kód programu, ktorý je čítaný a vykonávaný za behu špeciálnym procesom, tzv. interpreterom. Prívlastok „skriptovací“ teda nerobí z jazyka PHP niečo menejcenné.

Originálne znenie licencie pre používanie PHP nájdeme na webovej stránke http://www.php.net/license/.


PHP interpreter môžeme prevziať z webovej stránky http://www.php.net/.


HTTP server Apache môžeme prevziať z webovej stránky http://httpd.apache.org/.

LAMP – Linux, Apache, MySQL, PHP
WAMP – Windows, Apache MySQL, PHP
XAMPP – Apache, MySQL, PHP, Perl pre rôzne platformy (X - multiplatformný).

„The Uniform Server“ môžeme prevziať z webovej stránky http://www.uniformserver.com/.
V čase vzniku tohto materiálu bola najnovšou verziou 5.5-Nano (*UniServer5_5.exe*).

Server je sieťová aplikácia. Jeho činnosť môže byť „podozrivá“ pre inštalovaný firewall. Ak firewall zobrazí upozornenie, stačí povoliť aplikácii prístup na internet.

Uniform Server „počúva“ na požiadavky klientov na niektorom zo sieťových portov (napr. 80). Ak tento port používa aj iná aplikácia (napr. Skype), dôjde ku konfliktu a server sa nespustí. Riešením je ukončenie danej aplikácie.


PSPad môžeme prevziať z webovej stránky http://www.pspad.com/.

dozvieme, že práve zobrazená stránka je na počítači uložená v priečinku „/www/“. Všetko, čo uložíme do tohto priečinka, nám server vie ponúknuť na adrese http://localhost/.

Zadanie 1.2	Inštalujte HTTP server na lokálny počítač. Spustite server a overte si, že server beží.
Zadanie 1.3	Súbor *index.php* sa nachádza v priečinku, kde sme rozbalili Uniform Server (napr. *D:\UniServer\www*). Čo zobrazí prehliadač, ak namiesto adresy http://localhost/index.php zobrazíme súbor ako lokálny (z ponuky Súbor|Otvoriť súbor ...)? Viete to zdôvodniť?

Prvý skript

Samotný kód PHP skriptov je obyčajný text uložený v textovom dokumente. Pre písanie skriptov by sme mohli použiť ľubovoľný textový editor (napr. aj Poznámkový blok). Výhodnejšie je však použiť niektorý zo špecializovaných nástrojov, ktoré dokážu farebne označiť syntax PHP skriptov, uložiť PHP skript vo vybranom kódovaní, prípadne doplniť zvyšnú časť príkazu. Pre naše potreby použijeme freewarový editor PSPad.

Súčasťou PHP skriptov môže byť aj (X)HTML kód. Aby PHP interpreter rozpoznal časti PHP kódu od samotného (X)HTML, musíme ich v dokumente správne označiť. Vytvoríme nový dokument typu PHP. Všimnime si, že editor automaticky doplnil do nového dokumentu reťazce „<?php“ a „?>“. Prvý reťazec označuje začiatok PHP kódu, druhý jeho koniec. Samotný PHP kód umiestnime medzi tieto dva reťazce. Texty, ktoré nie sú „obalené“ začiatkom a koncom PHP kódu, bude PHP interpreter ignorovať. Náš prvý skript bude robiť jednoduchú vec: vypíše naše meno. Na výpis textových reťazcov sa používa jazykový konštrukt `echo`.

Jazykovým konštruktom `echo` vieme vypísať aj hodnoty premenných a konštánt. Viac sa dozvieme v kapitole 3.

meno.php
```
<?php
  echo "Jožko Mrkvička";
?>
```

Skript uložíme do súboru *meno.php* v priečinku www. Výsledok behu skriptu zobrazíme v prehliadači. Adresa skriptu je http://localhost/meno.php.

Pri ukladaní dokumentu je dôležité zvoliť vhodnú znakovú sadu (ponuka Formát editora PSPad). Rovnakú znakovú sadu však musíme použiť aj pri zobrazení v prehliadači. Pre naše potreby sú vhodné znakové sady „UTF-8“ alebo „ISO 8859-2“. Ak sa aj napriek tomu niektoré znaky nezobrazujú správne, je pravdepodobné, že server používa inú, prednastavenú znakovú sadu. V UniTray, v ponuke Advanced | Edit Apache Confuguration upravíme konfiguračný súbor servera. Zmeníme na komentár (znakom #) riadok začínajúci textom „AddDefaultCharset ...“.

Zadanie 1.4
Otestujte, či váš prehliadač zobrazuje správne všetky znaky. V prípade potreby upravte konfiguračný súbor HTTP servera Apache. Po zmene konfiguračného súboru je potrebné HTTP server reštartovať (zastaviť a opätovne spustiť).

PHP skripty bežne obsahujú aj časti (X)HTML kódu. Skript dokonca môže obsahovať niekoľko blokov (X)HTML kódu a niekoľko blokov s PHP kódom v jednom súbore. Skript *meno.php* môžeme upraviť nasledovne.

zvyrazni_meno.php
```
<?php
  echo "Jožko ";
?>
<strong>
<?php
  echo "Mrkvička";
?>
</strong>
```

-> PHP interpreter ->

(X)HTML kód v prehliadači
```
Jožko <strong>
Mrkvička</strong>
```

HTTP server, ktorý sme spustili, funguje rovnako ako väčšina HTTP serverov

v internete. V prípade, že náš počítač je pripojený do siete internet a má pridelenú verejnú IP adresu, môže sa k serveru pripojiť ktokoľvek. V prednastavenej konfigurácii servera však nie sú nastavené prístupové heslá potrebné k manažovaniu servera. Preto by sme takto nastavený server nemali nikdy nasadiť do ostrej prevádzky. Využívať ho budeme len na testovanie a ladenie našich skriptov.

Všimnime si ešte jednu dôležitú vec, príponu PHP skriptov. Podľa prípony súboru HTTP server vie, že klient si vyžiadal PHP skript a ešte pred tým, ako ho klientovi odošle, nechá ho spracovať PHP interpreteru. Klientovi sa tak nedostane samotný skript, ale výsledok jeho behu. Naše skripty by sme preto mali vždy ukladať do súborov s príponou registrovanou pre php interpreter (php). V opačnom prípade sa klientovi odošle priamo zdrojový kód skriptu.

V závislosti od konfigurácie servera môžu mať PHP skripty niekoľko rôznych prípon. Najčastejšie sú to prípony php, phtml a php3. Odporúčame používať príponu php.

Čo sme sa naučili

Oboznámili sme sa s princípom tvorby dynamických webových stránok. Ukázali sme si ako pracuje PHP, jedna z technológií na strane servera. Naučili sme sa ako spustiť HTTP server. Ukázali sme si ako oddeliť PHP kód od (X)HTML kódu a vytvorili sme prvý php skript. Vieme, že HTTP server v prednastavenej konfigurácii nie je vhodný do ostrej prevádzky. Poznáme dôvod, prečo PHP skripty musia mať príponu php.

Kapitola 2: PHP a (X)HTML, syntax jazyka PHP, znovu použitie PHP kódu, dátum a čas

Spolužitie PHP a (X)HTML

V predchádzajúcej kapitole sme si ukázali ako vzájomne kombinovať PHP kód s (X)HTML. Toto spojenie môže byť ešte tesnejšie. Samotný PHP kód môže generovať (X)HTML časti výsledného dokumentu.

zvyrazni_meno.php

```
<?php
  echo "Jožko <strong>Mrkvička</strong>";
?>
```

Na otázku, ktorý spôsob zvoliť, zrejme neexistuje jednoznačná odpoveď. Záleží na tom, aké veľké a ako štruktúrované časti (X)HTML kódu chceme generovať. Postupom času však získame cit pre to, ako sa rozhodnúť.

Jazyk PHP je pri volaní funkcií „case-insensitive“. V názve príkazov (funkcií) nerozlišuje veľké a malé písmená. Príkazy `echo`, `ECHO` alebo aj `eChO` PHP interpretuje rovnako. Napriek tomu je však vhodné, ak príkazy píšeme jednotne, napr.: `prikaz()`, `zlozenyPrikaz()`.

POZOR! V názve premenných (konštánt) PHP rozlišuje medzi veľkými a malými písmenami. Viac o názvoch premenných sa dozvieme v Kapitole 3.

Prehľadnejší zápis PHP kódu a jeho znovu použitie

PHP nám umožňuje vytvárať dlhé zápisy skriptov. Takýto spôsob je však príliš neprehľadný. Vyznať sa po nejakom čase v niekoľko sto riadkovom kóde už môže byť dosť náročné. Rozsiahly PHP kód môžeme rozdeliť na niekoľko menších, logických častí a každú z nich uložiť do samostatného súboru. Výsledok potom „vyskladáme“ z týchto častí. Jednotlivé časti a miesto ich vloženia je vhodné označiť komentárom.

Dokumenty, ktoré sme generovali v predchádzajúcich častiach, neboli validné. Chýbali im povinné časti z (X)HTML. Vytvoríme si súbory, ktoré vygenerujú začiatočnú a koncovú časť (X)HTML dokumentu a medzi ne vložíme náš samotný skript.


Užitočným pomocníkom pri písaní skriptov je manuál jazyka PHP. Jeho aktuálnu verziu môžeme prevziať z webovej stránky http://www.php.net/download-docs.php. Najpohodlnejšie sa pracuje s manuálom vo formáte chm (HTML Help). Pred prvým spustením v OS Windows XP SP2 (alebo novšom) je potrebné tento súbor odblokovať (vlastnosti súboru - tlačidlo odblokovať). Najnovšie verzie už nie sú lokalizované do slovenčiny. V prostredí LMS Moodle si môžeme prevziať jednu zo starších verzií čiastočne lokalizovanú do slovenčiny.

hlavicka.php

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="sk" lang="sk">
  <head>
    <meta http-equiv="Content-Type" content="application/xhtml+xml;
UTF-8" />
    <title>Programovanie v PHP</title>
```

```
    <meta name="language" content="sk" />
  </head>
  <body>
```

pata.php

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

validny_dokument.php

```
<?php
  /*
  tento dokument je "zložený" z troch dokumentov, hlavička a päta
  obsahujú začiatočnú a koncovú časť validného (X)HTML dokumentu
  */
  include "hlavicka.php"; // hlavicka (X)HTML dokumentu
  echo "Validný (X)HTML dokument.";
  include "pata.php";     // pata (X)HTML dokumentu
?>
```

Namiesto príkazu `include()` by sme mohli použiť aj `require()`. Rozdiel je v tom ako príkazy spracovávajú chyby (napr. chýbajúci súbor). Ak nastane chyba, `include()` vygeneruje varovanie, ale vykonávanie skriptu pokračuje ďalej. `require()` by v tejto situácii vygeneroval chybové hlásenie a vykonávanie skriptu by sa ukončilo. Ak aplikácia dokáže fungovať aj v prípade, že sa súbor nepodarilo vložiť, použijeme `include()`. Ak bez vloženého súboru aplikácia nedokáže pokračovať, použijeme `require()`.

Ak HTTP server dostane požiadavku na zobrazenie dokumentu *validny_dokument.php*, PHP interpreter „vyskladá“ výsledok tak, že spojí všetky časti dokopy a ako jeden celok vráti serveru. Rozdelenie skriptu do viacerých dokumentov má aj ďalšiu výhodu. Hlavičku a pätu dokumentu, ktoré sme si týmto spôsobom definovali, môžeme používať aj v ďalších skriptoch, a to bez toho, aby sme ich museli opäť vytvárať.

Všimnime si aj komentáre v skripte. PHP umožňuje vkladať dva typy komentárov. Jednoriadkový (`//`) a viacriadkový (`/* ... */`). Jednoriadkový komentár sa ukončí znakom konca riadku (alebo reťazcom ukončenia aktuálneho bloku PHP kódu `?>`), viacriadkový má svoj ukončovací reťazec (`*/`).

Zadanie 2.1

Horlivý programátor, ktorý si chcel vyskúšať písanie komentárov do skriptov, to trochu prehnal. Zdôvodnite a opravte chybu v jeho skripte (výsledkom behu skriptu mal byť reťazec „Test komentárov“):

```
<?php
  /*
  echo "Test komentárov"; /* vypise text */
  */
?>
```

Pracujeme s dátumom a časom

Skripty, ktoré sme zatiaľ vytvorili, veľa dynamiky neobsahovali. Výsledok ich behu bol vždy rovnaký. Jednou z najjednoduchších možností pre zdynamizovanie obsahu webových stránok je využiť čas a jeho zmenu.

Operačný systém UNIX bol vyvíjaný v 60-tych a 70-tych rokoch 20. storočia. Začiatok UNIX-ového času bol stanovený na 1. 1. 1970 00:00:00 a tento čas predstavuje začiatok tzv. UNIX-ovej epochy.

PHP dokáže pomocou funkcie `time()` prečítať systémový čas servera (tzv. timestamp - časová pečiatka). Výsledkom je však pre človeka trochu zvláštny formát. Funkcia `time()` vracia čas ako 32 bitové číslo vyjadrujúce počet sekúnd od 1. 1. 1970 00:00:00 v meste Greenwich. Napr. začiatok roka 2010 je reprezentovaný číslom 1262304000. Reprezentácia času v tomto formáte má však aj svoje výhody. S časom vieme pracovať ako s číslom (porovnávanie, sčitovanie, časový posun a pod.).

```
echo "Stretneme sa o ho
dinu, t.j. o ";
echo date("H:i", time()
+60*60);
```

☹

```
The end is near:
January 19, 2038
  03:14:07 GMT
```

Ako by ste vysvetlili vyššie uvedený „vtip“?

Zobraziť čas na webovej stránke v tomto formáte by bolo prakticky nepoužiteľné. Na úpravu času do zrozumiteľnejšej podoby využijeme funkciu `date()`. Funkcia `date()` vracia čas formátovaný podľa zadaného formátovacieho reťazca využitím aktuálnej alebo zadanej časovej pečiatky. Aktuálny čas na webovej stránke zobrazíme jednoduchým skriptom.

cas.php

```
<?php
  include "hlavicka.php";
  echo "Dobrý deň, presný čas: ";
  date_default_timezone_set("Europe/Bratislava");
  echo date("H:i:s"); //to isté ako echo date("H:i:s", time());
  include "pata.php";
?>
```

Formátovací reťazec je v našom skripte reťazec `"H:i:s"`:

- `"H"` - 24 hodinový formát hodiny aj s úvodnými nulami,
- `"i"` - minúty aj s úvodnými nulami,
- `"s"` - sekundy aj s úvodnými nulami,
- `":"` - nemá špeciálny význam, slúži ako oddeľovač časových údajov.

Ďalšie formátovacie znaky príkazu `date()` nájdeme v manuáli PHP http://sk.php.net/manual/en/function.date.php.

V príkaze `date()` sme nezadali časovú pečiatku. Použila sa teda aktuálna časová pečiatka. Keďže aktuálny čas sa mení v závislosti od časového pásma, na jeho spresnenie sme použili príkaz `date_default_timezone_set()`.

Príkaz `date()` vie formátovať aj iné časové pečiatky. Dátum „narodenia“ systému UNIX vypíšeme nasledovne:

narodenie_unix.php

```
<?php
  include "hlavicka.php";
  echo "UNIX sa podľa nášho času narodil: ";
  date_default_timezone_set("Europe/Bratislava");
  echo date("j. n. Y \o H:i:s.",0);
  include "pata.php";
?>
```

Všimnime si znaky „`\o`“. Znak „`o`“ formátuje rok. My však potrebujeme znak „`o`“ ako predložku pre časový údaj hodín. Aby ho príkaz `date()` neakceptoval ako formátovací, doplníme pred neho znak „`\`“.

Viac o špeciálnych znakoch v reťazcoch sa dozvieme v Kapitole 3.

Zadanie 2.2	Vytvorte PHP skript, ktorý okrem aktuálneho dátumu a času vo zvolenom formáte vypíše aj číslo týždňa v roku. Hlavičku a pätu dokumentu uložte do samostatných súborov.
Zadanie 2.3	V akom formáte zobrazí čas príkaz `echo date("@B")`?
Zadanie 2.4	Doplňte do vášho skriptu komentáre vysvetľujúce jeho jednotlivé časti. Jednotlivé príkazy môžete komentovať jednoriadkovým komentárom. Logické bloky skriptu, ktoré si vyžadujú dlhší komentár, môžete komentovať viacriadkovým komentárom.

Čo sme sa naučili

Naučili sme sa pomocou PHP skriptu generovať (X)HTML kód. Ukázali sme si ako prehľadne písať skripty, ako rozdeliť PHP kód do viacerých súborov tak, aby sme ich mohli použiť vo viacerých aplikáciách, a ako dokumentovať PHP kód komentármi. Naučili sme sa vytvárať validné (X)HTML dokumenty. Vieme pracovať s časom a dátumom a zobraziť ich vo zvolenom formáte.

Kapitola 3: Práca s premennými

Premenná a jej typ, príkaz priradenia

Premenné slúžia na uchovanie hodnôt, s ktorými v skriptoch pracujeme. Premenné

Chybne pomenované premenné sú napr. `$4časť`, `$2`, `$ x`, `$prva premenna`.

Správne pomenované premenné sú napr. `$_4časť`, `$_2`, `$_x`, `$prva_premena`, `$_`, `$premenna` a pod.

Aj keď v názve premennej môžeme použiť znaky s diakritikou, odporúčame tak nerobiť.

Pristupovať k hodnote neinicializovanej premennej predstavuje potenciálne bezpečnostné riziko. Viac sa dozvieme v Kapitole 9.

Uniform Server môže byť spustený v dvoch rôznych konfiguráciách PHP: Production a Development. V konfigurácii Development nás PHP upozorní na použitie neinicializovanej premennej. V konfigurácii Production je toto upozornenie potlačené. Konfiguráciu PHP môžeme zmeniť kliknutím na 1 | Advanced | Php.ini switch to ...
Po zmene konfigurácie je potrebné server reštartovať.
Počas tvorby skriptov odporúčame použiť konfiguráciu Development.

Ďalšie operátory jazyka PHP nájdeme v manuáli http://www.php.net/manual/en/language.operators.php.

URI (Uniform Resource Identifier) je reťazec znakov s definovanou štruktúrou použitý na identifikáciu alebo pomenovanie zdroja v sieti internet.

Skriptu môžeme poslať aj viac hodnôt. V jeho URI ich navzájom oddelíme znakom &: http://localhost/skript.php?p1=h1&p2=h2&p3=h3.

sú v PHP reprezentované začiatočným znakom „$“, za ktorým nasleduje meno premennej. Meno premennej začína písmenom alebo podčiarkovníkom. PHP v názvoch premenných rozlišuje veľké a malé písmená. Napr. `$pom` a `$Pom` sú dve, rozdielne premenné.

Premenné nie je potrebné deklarovať. Deklarácia prebehne v tom okamihu, keď premennú prvý krát použijeme. Z toho vyplýva aj ďalšia vlastnosť premenných. Typ premennej sa automaticky určí v okamihu priradenia hodnoty do premennej. Dokonca, typ premennej sa podľa potreby automaticky skonvertuje na vhodný typ. Pre potreby nášho kurzu budeme používať premenné typu celé číslo (Integer), reálne číslo (Floating point numbers), reťazec (String), pole (Array). Ak do premennej nič neuložíme (a pristupujeme k nej), PHP do nej automaticky vloží hodnotu prázdneho reťazca.

Hodnotu do premennej priradíme priraďovacím príkazom „=“ (`$premenná = výraz`). Výraz na pravej strane sa vyhodnotí a výsledok sa priradí do premennej na ľavej strane.

```
<?php
  $x = "1";          // $x je reťazec ("1")
  $x = $x + 2;       // $x je celé číslo (3)
  $x = $x + 1.3;     // $x reálne číslo (4.3)
  $x = 5 + "10 x"; // $x je celé číslo (15)
  $x[1] = $x;
  $x["a"] = 3.3;     // $x je pole (1 => 15, "a" => 3.3)
?>
```

Výrazy môžu obsahovať (okrem iného) aritmetické operátory a operátor zreťazenia:

+ - sčítanie	(`$x = 13 + 5; // 18`),
- - odčítanie	(`$x = 13 - 5; // 8`)
***** - násobenie	(`$x = 13 * 5; // 65`),
/ - delenie	(`$x = 13 / 5; // 2.6`),
% - zvyšok po delení	(`$x = 13 % 5; // 3`),
. - operátor zreťazenia	(`$x = 13 . 5; //"135"`).

Superglobálna premenná $_GET

PHP umožňuje spracovávať hodnoty, ktoré skriptu pošleme (napr. v URI skriptu, pomocou formulára a pod.). Na prístup k týmto hodnotám nám slúžia tzv. superglobálne premenné. Sú to premenné, ktoré sú automaticky prístupné vo všetkých častiach skriptu bez toho, aby sme ich museli definovať alebo inicializovať. Jednou z nich je premenná `$_GET`. Premenná `$_GET` je typu pole a sú v nej prístupné všetky hodnoty, ktoré sme skriptu poslali v URI.

Nasledujúci kód je ukážkou jednoduchého skriptu na premenu jednotiek pamäťovej kapacity. V URI pošleme skriptu počet bajtov. Výstupom skriptu je počet KiB.

http://localhost/premena_jednotiek.php?x=1024

```
<?php
  include "hlavicka.php";
  echo "<h1>Premena jednotiek:</h1>";
  $x = $_GET["x"];
  echo $x . " B = " . $x / 1024 . " KiB";
  include "pata.php";
?>
```

Hodnota premennej `$x` je automaticky prístupná v poli `$_GET` na pozícii s indexom `"x"`.

Spracovanie externých hodnôt a prístup k nim v PHP nie je komplikované. Je tu však potenciálne riziko, ktoré si nie vždy uvedomujeme. V skripte *premena_jednotiek.php* očakávame, že hodnotou premennej `$x` bude číslo (celé alebo reálne). Čo sa však stane, ak „záškodník“ zavolá náš skript s takýmto

parametrom:

http://localhost/premena_jednotiek.php?x=?

Všimnime si, že súčasťou výpisu nášho skriptu je hodnota premennej `$x`. Jej hodnotou je však (X)HTML reťazec pre vloženie obrázka do webovej stránky. Výsledkom je, že na webovej stránke sa tento obrázok skutočne zobrazí. Hocikto teda môže dosiahnuť to, aby súčasťou našej stránky bol ľubovoľný (X)HTML kód (alebo napr. aj JavaScript). A to už môže byť problém (naša stránka môže obsahovať nebezpečný kód). Aby sa tak nestalo, v skripte zabezpečíme, aby hodnotou premennej `$x` bolo vždy číslo. Náš kód upravíme nasledovne:

http://localhost/premena_jednotiek.php?x=1024

```
<?php
  include "hlavicka.php";
  echo "<h1>Premena jednotiek:</h1>";
  $x = $_GET["x"];
  $x = strip_tags($x);
  setType($x, "float");
  echo $x . " B = " . $x / 1024 . " KiB";
  include "pata.php";
?>
```

PHP ponúka množstvo funkcií pre prácu s reťazcami. Záujemcom odporúčame časť „Reťazcové funkcie“ v manuáli jazyka PHP: http://www.php.net/manual/en/ref.strings.php.

Funkcia `strip_tags()` odstráni z reťazca (uloženého v premennej `$x`) všetky (X)HTML a PHP značky. Funkcia `settype()` pretypuje premennú `$x` na reálne číslo. Nevhodne zadanú hodnotu náš skript upraví. Ak hodnotou premennej `$x` bolo číslo, táto úprava ho nezmení.

Radikálnejším, ale bezpečnejším, riešením je nevhodné hodnoty premenných neupravovať. Ak hodnotou premennej `$x` nie je reálne číslo, náš skript by ju mal ignorovať a nepracovať s ňou. Viac o bezpečnosti sa dozvieme v Kapitole 9.

Premenná typu reťazec

Ak sa pozrieme na niektoré z predchádzajúcich skriptov, zistíme, že najpoužívanejším typom je typ reťazec. Reťazec je postupnosť znakov uzatvorená v úvodzovkách (napr. `"reťazec"`) alebo v apostrofoch (napr. `'reťazec'`). Rozdiel je v tom, že reťazce uzatvorené v apostrofoch nie sú kontrolované na obsah premenných a escape sekvencií. Pre lepšie pochopenie si pozrime nasledujúci skript:

http://localhost/retazce.php

```
<?php
  include "hlavicka.php";
  $x = 2;
  echo "\$x = $x"; // $x = 2
  echo '\$x = $x'; // \$x = $x
  include "pata.php";
?>
```

Pri výpise reťazca uzatvoreného v úvodzovkách sa premenná `$x` nahradila jej hodnotou. Ak potrebujeme vypísať názov premennej ($x), použijeme tzv. escape sekvenciu `\$`. Znak `\` zabezpečí, že znak `$` bude interpretovaný ako obyčajný znak bez špeciálneho významu. V reťazcoch, ktoré sú uzatvorené v apostrofoch, sa escape sekvencie používať nedajú. Reťazce uzatvorené v apostrofoch interpretujú celý svoj obsah ako obyčajný text bez špeciálneho významu.

Ďalšie escape sekvencie:

\"	úvodzovka
\\	spätná lomka
\$	znak dolára
\r	znak návrat vozíka
\n	znak konca riadku
\t	znak tabulátora

Čo sme sa naučili

Naučili sme sa správne pomenovať premenné v jazyku PHP. Vieme, že premennú nie je potrebné vopred deklarovať a jej typ sa určí automaticky podľa kontextu, v akom ju použijeme. Vieme používať príkaz priradenia a základné operátory vo výrazoch. Naučili sme sa ako skriptu poslať hodnoty v URI a aké nebezpečenstvo hrozí, ak tieto hodnoty použijeme bez úpravy. Poznáme špecifiká premennej typu reťazec a vieme aké dôsledky má použitie úvodzoviek a apostrofov.

Zadanie 3.1

Spustite skript *premena_jednotiek.php* z tejto kapitoly v režime Production a v režime Development. Aký je

	výstup skriptu, ak v URI nepošleme premennú s danou hodnotou?
***Zadanie 3.2**	Vytvorte skript, ktorý vypíše dátum v tvare: Dnes je štvrtok, 24. decembra 2009. Návod: Názvy dní v týždni a mesiacov v roku uložte do premenných typu pole.
Zadanie 3.3	Upravte skript *premena_jednotiek.php* z tejto kapitoly tak, aby jeho výstup bol nasledovný (v URI sme zadali x=15): **Premena jednotiek:** Hodnota premennej "$x" je 15 15 KiB = 15360 B Upravený skript *premena_jednotiek.php* uložte pod iným menom. S pôvodnou verziou budeme ešte neskôr pracovať.

Kapitola 4: Podmienené príkazy, zložený príkaz

Od väčšiny algoritmov (skriptov), ktoré budeme programovať, očakávame, že budú reagovať na okolité, meniace sa podmienky. Podľa týchto podmienok sa výpočet môže vetviť tak, aby správne zareagoval na každú z možností. PHP ponúka niekoľko možností na podmienené vetvenie skriptov.

Podmienený príkaz IF

Podmienený príkaz IF má v PHP viacero tvarov:

if	if else	if elseif else

`if (podmienka) {`
`  zoznam príkazov;`
`}`
	`if (podmienka) {`
	`  zoznam príkazov;`
	`}`
	`else {`
	`  zoznam príkazov;`
	`}`
		`if (podmienka 1) {`
		`  zoznam príkazov;`
		`}`
		`elseIf (podmienka 2) {`
		`  zoznam príkazov;`
		`} // ďalšie elseIf bloky`
		`else {`
		`  zoznam príkazov;`
		`}`

Podmienkou je výraz, ktorý je po vyhodnotený pravdivý (`TRUE`) alebo nepravdivý (`FALSE`). Všimnime si aj páry zátvoriek `{}`. Zložené zátvorky, tzv. zložený príkaz, uzatvárajú blok príkazov. V prípade, že blok príkazov je tvorený len jedným príkazom, obaľujúce zátvorky použiť nemusíme. Napriek tomu je dobrým zvykom zátvorky používať vždy. Pri neskoršej úprave skriptov, najmä ak dopĺňame ďalšie príkazy, by sme na zátvorky mohli zabudnúť.

Skript *premena_jednotiek.php* z predchádzajúcej kapitoly mal jednu chybičku krásy. V prípade, ak sme skriptu neposlali žiadnu hodnotu v URI, skript priradil do premennej $x hodnotu neinicializovanej premennej (`$x = $_GET["x"]`). Tento problém vieme v PHP odstrániť pomerne ľahko. Pred samotným priradením otestujeme, či je premenná `$_GET["x"]` inicializovaná. Využijeme funkciu `isset()`.

V podmienkach môžeme používať **logické** operátory:
`and` - logické and výrazov,
`or` - logické or výrazov,
`xor` - logické exkluzívne or,
`!` - logická negácia,

operátory porovnania:
`==` - rovnosť,
`===` - rovnosť, aj typov,
`!=` - nerovnosť
(bez medzery medzi `!=`),
`<` - menší,
`>` - väčší,
`<=` - menší alebo rovný,
`>=` - väčší alebo rovný.

V PHP môžeme použiť aj tzv. podmienený operátor:
`výraz1?výraz2:výraz3`

Ak je `výraz1` pravdivý, výsledkom je `výraz2`.
V opačnom prípade je výsledkom `výraz3`.

Napr.:
`$max=$x>$y?$x:$y;`

premena_jednotiek.php

```php
<?php
  include "hlavicka.php";
  echo "<h1>Premena jednotiek:</h1>";
  if (isset($_GET["x"])) {
    $x = $_GET["x"];
    $x = strip_tags($x);
    settype($x, "float");
    echo $x . " B = " . $x / 1024 . " kiB";
  }
  else {
    echo "Hodnota nebola zadaná.";
  }
  include "pata.php";
?>
```

Ak je premenná `$_GET["x"]` inicializovaná (`isset($_GET["x"])` je `TRUE`), prevedieme jednotky a výsledok vypíšeme. V opačnom prípade vypíšeme upozornenie (`echo "Hodnota nebola zadaná."`).

Programujeme jednoduchý web

V tomto okamihu už máme dostatok vedomostí na to, aby sme si ukázali, ako naprogramovať jednoduchý web. Náš ukážkový web obsahuje niekoľko stránok (uvod, prva, druha) a menu, pomocou ktorého si vyberieme stránku, ktorú chceme zobraziť. Identifikátor stránky, ktorú chceme zobraziť, budeme posielať v URI skriptu. Hodnota identifikátora bude v skripte prístupná v premennej `$_GET["id"]`, resp. `$id`.

jednoduchy_web.php

```php
<?php
  include "hlavicka.php";
  include "menu.php";
  if (isset($_GET["id"])) {
    $id = $_GET["id"];
    if ($id == 1) {
      include "prva.php";
    }
    elseif ($id == 2) {
      include "druha.php";
    }
    else {
      include "uvod.php";
    }
  }
  else {
    include "uvod.php";
  }
  include "pata.php";
?>
```

menu.php

```php
<p>Vyberte si:</p>
<a href="?id=0">úvod</a><br />
<a href="?id=1">prvá</a><br />
<a href="?id=2">druhá</a><br />
```

Pri prvom volaní (http://localhost/jednoduchy_web.php) premenná `$_GET["id"]` nie je inicializovaná. Skript zobrazí menu (`include "menu.php"`) a úvodnú stránku (`include "uvod.php"`). Ak klikneme na niektorú položku v menu, v URI sa skriptu pošle hodnota premennej `$id`. Podľa jej hodnoty skript zobrazí príslušnú stránku. Ak hodnota premennej `$id` nie je žiadna z očakávaných, zobrazí sa úvodná stránka.

Zadanie 4.1	Vytvorte skript, ktorý návštevníka stránky pozdraví podľa toho, aká je aktuálna hodina dňa. (dobré ráno, dobré dopoludnie, ..., dobrý večer, dobrú noc)
Zadanie 4.2	Vytvorte skript, ktorý na stránke vypíše príslušné heslo dňa v týždni (pondelok, utorok - štvrtok, piatok, víkend): Včera bolo voľno, ale dnes to opäť začalo :-| Sme v tom až po uši. Robota všade naokolo :-(

Pozornejším z vás určite napadlo, že je zbytočne komplikované posielať skriptu v premennej `$id` číslo stránky. Možno by bolo jednoduchšie poslať skriptu priamo názov dokumentu, v ktorom sa stránka nachádza: namiesto `<a href="?id=0">` použiť `<a href="?id=uvod.php">`

Správnu stránku by sme potom zobrazili príkazom: `include $id`.

Toto riešenie by fungovalo až do chvíle, kedy by prípadný útočník začal manipulovať s hodnotou v URI skriptu. Stačí, aby zadal názov súboru, ktorý obsahuje neverejné informácie (napr. prihlasovacie meno a heslo).

PHP ponúka ešte jeden podmienený príkaz, príkaz `switch`. Test hodnoty premennej $id by pomocou `switch` vyzeral nasledovne:

```php
switch ($id){
 case 1:
  include "prva.php";
  break;
 case 2:
  include "druha.php";
  break;
 default:
  include "uvod.php";
}.
```

Ďalšie informácie nájdeme v manuáli jazyka PHP.

	Dnes nepracujte, šetrite sa na zajtra! Zajtra bude voľno :-) Konečne voľno! Žiadna práca :-))

Čo sme sa naučili

Ukázali sme si význam a použitie podmienených príkazov vetvenia. Poznáme logické operátory a operátory porovnania v PHP. Vieme naprogramovať skript pre zobrazenie stránok jednoduchého webu.

Kapitola 5: Príkazy cyklu, premenná typu pole

Častokrát sa dostaneme do situácie, keď budeme potrebovať opakovane vykonávať skupinu príkazov. Rovnako ako iné programovacie jazyky, aj PHP ponúka niekoľko možností príkazov cyklu.

Príkaz cyklu for

Príkaz cyklu `for` je najvýkonnejším príkazom cyklu, ktorý PHP ponúka. Jeho všeobecný tvar je nasledovný:

for
```
for (výraz1; výraz2; výraz3) {
    príkaz1;
    príkaz2;
    ...
}
```

`výraz1` sa vyhodnotí ešte pred začiatkom vykonávania cyklu. Potom sa vyhodnotí `výraz2`. Ak je jeho hodnota `TRUE` vykoná sa telo cyklu. Na konci každého prechodu telom cyklu sa vyhodnotí `výraz3`. Telo cyklu sa vykonáva tak dlho, pokiaľ je hodnota `výraz2` rovná `TRUE`. Pre lepšie pochopenie si pozrime nasledujúci príklad, ktorý vypíše denný plánovací kalendár od 6^{00} do 23^{00} (kalendár vypisujeme po hodinách)

harmonogram.php
```
<?php
  include "hlavicka.php";
  echo "<h1>Program dňa</h1>";
  for ($i=6; $i<=22; $i++) {
      echo "$i:00 - ".($i+1).":00 .............<br />";
    }
  include "pata.php";
?>
```

Na začiatku sa vyhodnotí výraz `$i=6` (do premennej `$i` sa priradí hodnota 6). Potom sa vyhodnotí výraz `$i<=22`. Tento výraz je pravdivý, takže sa vykoná telo cyklu. Vyhodnotí sa výraz `$i++`, hodnota premennej `$i` sa zvýši na 7. Keďže výraz `$i<=22` je stále pravdivý, pokračuje vykonaním tela cyklu. Toto sa opakuje až do vtedy, kým hodnota premennej `$i` nedosiahne 23.

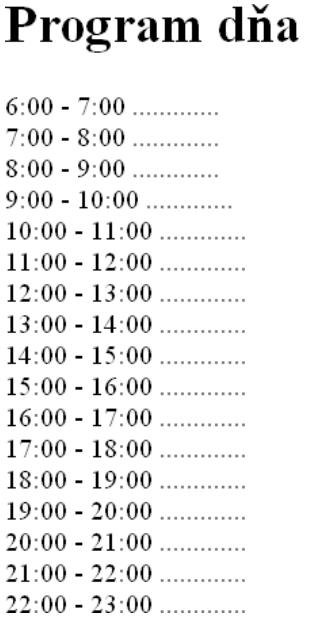
Príkazy cyklu while a do-while

Jednoduchšími príkazmi cyklu sú príkazy cyklu `while` a `do-while`.

while
```
while (podmienka) {
    príkaz1;
    príkaz2;
    ...
}
```

do-while
```
do {
    príkaz1;
    príkaz2;
    ...
} while (podmienka);
```

Príkazu `while` sa zvykne hovoriť aj príkaz s podmienkou na začiatku, pretože platnosť podmienky testuje ešte pred vykonaním samotného tela cyklu. Príkaz `do-while` testuje platnosť podmienky až po vykonaní tela cyklu a označuje sa aj ako príkaz s podmienkou na konci. Zatiaľ čo príkaz `do-while` vykoná telo vždy aspoň raz, príkaz `while` sa nemusí vykonať ani raz. Obidva cykly použijeme v situácii, v ktorej



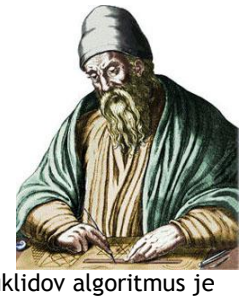
Výraz `$i++` je skráteným zápisom výrazu `$i=$i+1`. Podobne vieme výraz `$i=$i-1` nahradiť výrazom `$i--`.

dopredu nevieme presne určiť, koľkokrát sa majú príkazy vykonať, ale počet opakovaní závisí od platnosti podmienky.

Použitie príkazu cyklu `while` si ukážeme pri výpočte NSD pomocou Euklidovho algoritmu.

nsd.php
```
<?php
  include "hlavicka.php";
  echo "<h1>NSD dvoch čísiel</h1>";
  if (isset($_GET["x"], $_GET["y"])) {
      $x = strip_tags($_GET["x"]);
      settype($x, "integer");
      $y = strip_tags($_GET["y"]);
      settype($y, "integer");
      while ($x != $y) {
        if ($x>$y) {
            $x=$x-$y;
        }
        else {
            $y=$y-$x;
        }
      }
      echo "NSD je: $x";
    }
    else {
        echo "V URI skriptu zadajte prirodzené hodnoty x a y.";
    }
  include "pata.php";
?>
```

Pomocou funkcie `isset()` vieme súčasne testovať viacero premenných či boli inicializované, napr.: `isset($p1, $p2, $p3)`



Euklidov algoritmus je jeden z najstarších netriviálnych algoritmov ktoré ľudstvo pozná. Grécky matematik Euklides ho popísal približne okolo roku 300 p. n. l.

Premenná typu pole

S premennou typu pole sme sa už v krátkosti stretli v kapitole 3. Pole je špeciálna dátová štruktúra. Jediná premenná typu pole môže obsahovať niekoľko hodnôt rôznych typov. Hodnoty sú prístupné pomocou indexu, ktorý zapisujeme do hranatých zátvoriek „`[]`“.

Pokiaľ nedefinujeme inak, prvý index poľa má hodnotu 0 a každý ďalší je o 1 väčší. Premennú typu pole môžeme inicializovať niekoľkými spôsobmi, napr. postupným vkladaním prvkov do poľa: `$pole[0] = 0; $pole[1] = 1; ...`. Ukážeme si však použitie príkazu `array`, vďaka ktorému sa skráti PHP kód potrebný pre inicializáciu poľa.

den_v_tyzdni_1.php
```
<?php
  include "hlavicka.php";
  $dni = array("nedeľa", "pondelok", "utorok", "streda", "štvrtok",
               "piatok", "sobota");
  date_default_timezone_set("Europe/Bratislava");
  //date("w") vráti číslo dňa v týždni 0-nedeľa ... 6-sobota
  echo "Dnes je: " . $dni[date("w")];
  include "pata.php";
?>
```

PHP umožňuje vytvárať aj polia, ktorých indexy nie sú čísla, ale reťazce. Ide o tzv. asociatívne polia:
```
<?php
//vytvorenie
asociatívneho pola

$tel["Peter"]="0111";
$tel["Ivan"]="0222";
$tel["Kamil"]="0333";
$tel["Zdenka"]="0444";
echo "Kamilov tel. je:"
   . $tel["Kamil"];
?>
```

Uvedený príklad používa pre nás trochu nezvyčajné číslovanie dní v týždni. My sme skôr zvyknutí začínať pondelkom ako prvým dňom a končiť nedeľou ako siedmym dňom. Upraviť pole `$dni` tak, aby prvý prvok `"pondelok"` mal index 1, môžeme nasledovne.

den_v_tyzdni_2.php
```
<?php
  include "hlavicka.php";
  $dni = array(1 => "pondelok", "utorok", "streda", "štvrtok",
                    "piatok", "sobota", "nedeľa");
  date_default_timezone_set("Europe/Bratislava");
  //date("N") vráti číslo dňa v týždni 1-pondelok ... 7-nedeľa
```

```
  echo "Dnes je: " . $dni[date("N")];
  include "pata.php";
?>
```

Týmto spôsobom (`1 => "pondelok"`) sme inicializovali pole `$dni` od indexu 1. Každý nasledujúci prvok je o 1 väčší.

Aj pre výpis prvkov poľa nám PHP ponúka niekoľko možností. Univerzálnym spôsobom je použitie konštrukcie `foreach` (funguje aj vtedy, ak nepoznáme indexy prvkov poľa). Ak poznáme index prvého prvku poľa (a každý ďalší je vždy o 1 väčší), môžeme použiť niektorý z príkazov cyklu. Názvy dní z predchádzajúceho príkladu vypíšeme nasledovne.

vypis_prvkov_pola.php

```
<?php
  include "hlavicka.php";
  $dni = array(1 => "pondelok", "utorok", "streda", "štvrtok",
                    "piatok", "sobota", "nedeľa");
  echo "<h1>Výpis prvkov poľa:</h1>";

  //výpis prvkov poľa pomocou foreach
  foreach ($dni as $hodnota) {
      echo $hodnota . "<br />";
  }

  //výpis prvkov poľa pomocou for
  for ($i=1; $i <= count($dni); $i++) {
      echo $dni[$i] . "<br />";
  }
  include "pata.php";
?>
```

Všimnime si použitie príkazu `count($dni)`. Jeho výsledkom je počet prvkov poľa. Pri prechode poľom je výhodnejšie použiť konštrukciu `foreach`. Tento spôsob funguje rovnako dobre pre každé pole. O jeho indexy sa v podstate ani nemusíme zaujímať.

Ak by sme z nejakého dôvodu potrebovali pracovať aj s indexmi prvkov poľa, môžeme použiť alternatívnu syntax konštrukcie `foreach`:

```
foreach ($dni as $index
=>$hodnota) {
echo "$index: $hodnota<
br />";
 }
```

PHP dokáže pracovať aj s viacrozmernými poľami:

```
$pole =
  array(
    array(1, 2),
    array(3, 4)
  );
 echo $pole[0][1]; //2
```

```
Meniny má
 dnes: Hugo
 zajtra: Zita
 pozajtra: Richard
```

Zadanie 5.1	V súbore *meniny.php* je zoznam mien. Naprogramujte PHP skript, ktorý vypíše, kto má v daný deň meniny. Návod pre pokročilejších: Použite dvojrozmerné pole.
Zadanie 5.2	Upravte skript na výpočet NSD dvoch čísiel (*nsd.php*) tak, aby použil vylepšený Euklidov algoritmus. Návod: Namiesto postupného odpočítavania menšieho čísla od väčšieho budete počítať zvyšok po delení (*$x % $y*).
Zadanie 5.3	Čo je výsledkom nasledujúceho skriptu? `<?php` `  include "hlavicka.php";` `  echo "<h1>Záhada</h1>";` `  $x = 10; //ľubovoľné prirodzené číslo` `  $y = 1;` `  do` `  {` `  if ($x % $y == 0) echo $y . " ";` `  $y++;` `  }` `  while ($y <= $x);` `  include "pata.php";` `?>`
Zadanie 5.4	Upravte skript (*harmonogram.php*) na výpis časového harmonogramu dňa tak, aby bol formulár v tabuľke.

Zadanie 5.5	V súbore *meniny.php* je zoznam mien. Naprogramujte PHP skript, ktorý vypíše prehľadný menný kalendár pre zadaný rok. Návod pre pokročilejších: Použite dvojrozmerné pole.

Čo sme sa naučili

Naučili sme sa, aké príkazy cyklu ponúka PHP a podľa situácie sa vieme rozhodnúť, ktorý cyklus je najvhodnejší. Vieme vytvoriť premennú typu pole a pracovať s jej prvkami. Ukázali sme si, ako definovať vhodné indexy prvkov poľa a ako k týmto prvkom pristupovať.

Kapitola 6: PHP a formuláre

Webový formulár predstavuje pohodlný nástroj na vzájomnú komunikáciu medzi používateľom a webovou aplikáciou. Dáta z formulára sa PHP aplikácii môžu „doručiť“ dvoma spôsobmi: metódou GET alebo metódou POST.

V prípade metódy GET sú dáta posielané v URI skriptu (podobne ako sme v kapitole 3 posielali hodnotu premennej $x: premena_jednotiek.php?x=1024). Všetky dáta z formulára sa automaticky pripoja na koniec URI obslužného skriptu.

Metóda POST funguje tak, že dáta sú odoslané v tele HTTP požiadavky (zatiaľ sa nemusíme veľmi zamýšľať nad tým, čo to je HTTP požiadavka). Z praktického hľadiska sú tieto dáta pre používateľa neviditeľné, nikde sa nezobrazujú.

Dáta, ktoré „dorazili“ k obslužnému skriptu metódou GET, sú prístupné v superglobálnej premennej `$_GET`. Dáta odoslané metódou POST sú prístupné v superglobálnej premennej `$_POST`. Obidve premenné sú typu pole. Hodnota odoslanej premennej (presnejšie hodnota formulárového elementu) je uložená na pozícii s rovnomenným indexom.

metóda GET

```
<form method="post"
      action="skript.php">
<!-- formulárové elementy -->
</form>
```

metóda POST

```
<form method="get"
      action="skript.php">
<!-- formulárové elementy -->
</form>
```

Hodnota atribútu `action` definuje meno obslužného skriptu (v našom prípade `skript.php`), ktorý spracuje dáta z formulára. Metódu na odoslanie dát definujeme pomocou atribútu `method` elementu `form`. Pozrime sa bližšie na to, kedy ktorú metódu použiť.

Metódy GET a POST

Aj keď spracovanie hodnôt formulárových prvkov je takmer rovnaké bez ohľadu na použitú metódu ich odoslania, tieto metódy nie sú rovnocenné. Je dôležité sa vedieť správne rozhodnúť, ktorú z metód kedy použiť.

Metódu GET použijeme, ak:

- dáta z formulára **nemenia** stav servera (zápis do súboru, zmena v databáze a pod.),
- má zmysel uložiť si celú URI skriptu do záložiek (napr. výsledok vyhľadávania),
- formulár neobsahuje citlivé dáta (napr. heslo, číslo účtu a pod.),
- skripty vytvárame a ladíme, pretože hodnoty formulárových prvkov sú súčasťou URI skriptu, takže pomerne ľahko ich vieme skontrolovať.

Metódu POST použijeme, ak:

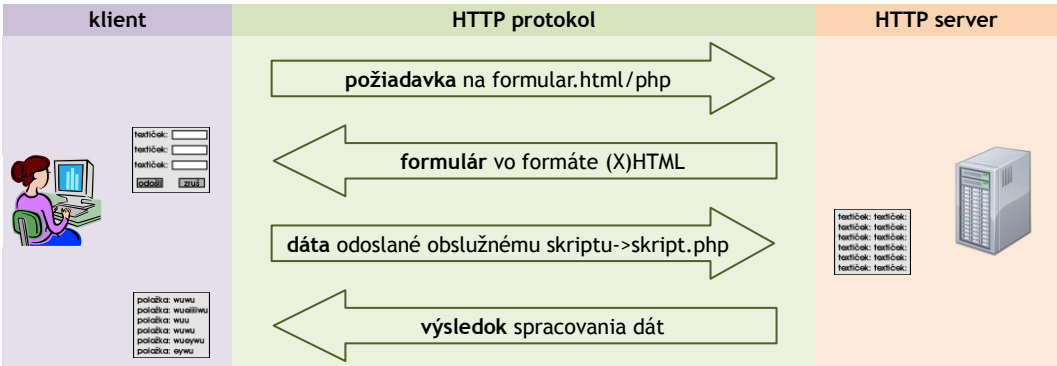
- dáta z formulára **menia** stav servera (zápis do súboru, zmena v databáze

a pod.),
- formulár odosiela veľké množstvo dát (množstvo dát prenášaných metódou GET je obmedzené),
- formulár obsahuje citlivé dáta (napr. heslo, číslo účtu a pod.),
- je aplikácia v ostrej prevádzke.

V skriptoch v tomto module budeme kvôli prehľadnosti používať metódu GET.

Formulár a obslužný skript

Pred tým, než vytvoríme prvý webový formulár, si ukážeme princíp vzájomnej komunikácie medzi formulárom a obslužným skriptom.



Obrázok 2: Princíp komunikácie používateľa s PHP skriptom pomocou formulára

Generovanie formulára a spracovanie jeho výsledkov môžeme obslúžiť jediným skriptom. Musíme však zabezpečiť, aby skript „vedel“ rozlíšiť stav, keď má zobraziť formulár, od stavu, kedy má dáta z formulára spracovať. Dosiahnuť to môžeme napr. skrytým políčkom formulára alebo testom inicializácie premenných z formulára.

Webový formulár môže byť uložený v „statickom“ X(HTML) dokumente alebo môže byť vygenerovaný dynamicky PHP skriptom. Záleží na tom, či formulár je zostavený napevno už vopred alebo sa generuje dynamicky (napr. z dát v súbore alebo v databáze). Súčasťou definície formulára je meno obslužného skriptu, ktorému budú dáta odoslané (`action="skript.php"`). Výsledok spracovania odoslaných dát sa opäť môže zobraziť v klientskom prehliadači webových stránok.

Prvý formulár

Vytvorenie webového formulára a obslužného skriptu si ukážeme na jednoduchom príklade. Vylepšíme skript *premena_jednotiek.php* z kapitoly 3. Používateľovi umožníme zadať počet vstupných jednotiek, vstupnú a výstupnú jednotku dátovej kapacity. Samotný skript, ktorý vygeneruje formulár, vyzerá nasledovne.

formular.php

```
<?php
  include "hlavicka.php";
?>
<h1>Zadajte hodnotu a jednotky:</h1>
<form method="get" action="premena_jednotiek_1.php">
  <label for="pj">Počet jednotiek:</label>
  <input type="text" name="pj" id="pj" /><br />
  <label for="vstupj">Vstupná jednotka:</label>
  <select name="vstupJ" id="vstupj">
    <option value="B">B</option>
    <option value="KiB">KiB</option>
    <option value="MiB">MiB</option>
    <option value="GiB">GiB</option>
    <option value="TiB">TiB</option>
  </select><br />
  <label for="vystupj">Výstupná jednotka:</label>
    <select name="vystupJ" id="vystupjj">
    <option value="B">B</option>
    <option value="KiB">KiB</option>
    <option value="MiB">MiB</option>
    <option value="GiB">GiB</option>
    <option value="TiB">TiB</option>
  </select><br/>
```

```
  <input type="submit" value="Vypočítaj" />
</form>
<?php
  include "pata.php";
?>
```

Všimnime si tri formulárové elementy: vstupné pole typu text, do ktorého používateľ zadá počet vstupných jednotiek a dva rozbaľovacie zoznamy, z ktorých si používateľ vyberie vstupnú a výstupnú jednotku. Dáta z formulára sa odošlú obslužnému skriptu *premena_jednotiek_1.php*. Samotný skript vyzerá nasledovne:

premena_jednotiek_1.php

```
<?php
  $j["B"]=0; $j["KiB"]=10; $j["MiB"]=20; $j["GiB"]=30; $j["TiB"]=40;
  include "hlavicka.php";
  echo "<h1>Prepočet jednotiek dátovej kapacity:</h1>";
  if (isset($_GET["pj"], $_GET["vstupJ"], $_GET["vystupJ"])){
      $pj = trim(strip_tags($_GET["pj"]));
      $vstupJ = strip_tags($_GET["vstupJ"]);
      $vystupJ = strip_tags($_GET["vystupJ"]);
      if (isset($j[$vystupJ], $j[$vstupJ]) and $pj != "") {
        $vysledok = $pj / pow(2, $j[$vystupJ] - $j[$vstupJ]);
        echo "$pj $vstupJ = $vysledok $vystupJ";
      }
      else {
          echo "Chybné alebo nezadané vstupné hodnoty.";
      }
  }
  else {
      echo "Nezadané vstupné hodnoty.";
  }
  echo "<br /><a href='formular.php' title='Zadať iné hodnoty'>znova</
a>";
  include "pata.php";
?>
```

Dáta z korektne vyplneného a odoslaného formulára sú prístupné v poli `$_GET`. Ak príslušné premenné nie sú inicializované (prvý `if`), skript vypíše chybovú správu. Ak premenné sú inicializované, ale ich hodnoty nie sú korektné (druhý `if`), skript opäť vypíše chybovú správu. Až keď obidva testy boli vyhodnotené ako pravdivé, skript prepočíta jednotky a výsledok vypíše.

Ak si nie sme istí, v akom formáte (type) dorazili hodnoty z formulára do skriptu, využijeme príkaz `print_r($premenna)`. Výsledkom je ľahko čitateľná informácia o premennej a jej hodnote.

Pre zjednodušenie prepočtu sme si do poľa `$j` uložili stupeň mocniny 2 príslušnej jednotky.

Funkcia `trim()` odstráni zo začiatku a z konca reťazca netlačiteľné znaky (napr. medzery).

Funkcia `pow(základ,exponent)` vráti hodnotu výrazu $základ^{exponent}$.

Ak by sme v skripte netestovali vstupné hodnoty, náš skript by pre korektné vstupy vypísal správny výsledok. Pre nekorektné alebo nezadané hodnoty by sa používateľovi zobrazilo chybové hlásenie PHP interpretera. Z takýchto hlásení sa však prípadný útočník môže dozvedieť informácie, vďaka ktorým môže náš skript zneužiť. Preto je vhodné ich nezobrazovať a nekorektný vstup programovo ošetriť. Viac o bezpečnosti sa dozvieme v Kapitole 9.

Zadanie 6.1	Upravte aplikáciu na prepočet jednotiek tak, aby sa dáta predávali obslužnému skriptu metódou POST.
Zadanie 6.2	Upravte aplikáciu na prepočet jednotiek tak, aby sa po odoslaní hodnôt z formulára zobrazil nielen samotný prepočet jednotiek, ale aj formulár. Používateľ tak bude môcť zadať ďalšie hodnoty a nebude musieť klikať na odkaz na formulár.

Čo sme sa naučili

Ukázali sme si, že existujú dve metódy ako poslať dáta z formulára obslužnému skriptu. Vieme aké majú výhody a nevýhody a vieme sa správne rozhodnúť, kedy ktorú použiť. Vieme, ako funguje komunikácia medzi formulárom a skriptom a naučili sme sa vytvárať jednoduché dynamické aplikácie, ktoré komunikujú s používateľom prostredníctvom formulárov. Vieme, že je dôležité overiť dáta z formulára a vieme ako to spraviť.

Poznámka: Ďalšia časť študijného materiálu je prístupná v elektronickej podobe. Príslušný elektronický dokument nájdete v LMS Moodle projektu ĎVUi alebo na priloženom CD.

Kapitola 7: Práca so súbormi

Táto kapitola nás posunie o veľký krok dopredu. Používanie súborov pri programovaní v PHP pridá nášmu programovaniu ďalší rozmer. Možnosti uchovávať dáta aj po tom, čo skript skončí, pamätať si reakcie návštevníkov našej stránky, či jednoducho len počítať počet návštevníkov nášho webu, sú veľmi lákavé. Ukážeme si základné príkazy pre prácu so súbormi a zopár užitočných príkladov, ako príkazy používať. Nezabudneme ani na bezpečnosť. Naprogramovať skript tak, aby umožňoval modifikovať obsah súboru a zároveň neumožnil nechcenú zmenu jeho obsahu nejakým záškodníkom nie je jednoduché.

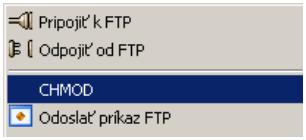
Väčšina internetových severov beží pod operačným systémom typu UNIX (LINUX). Aj keď si naše skripty odladíme na lokálnych počítačoch, nakoniec ich premiestnime na nejaký internetový server.

Keďže tieto systémy boli navrhnuté ako viacužívateľské (v jednom okamihu môže systém využívať viac ľudí), bolo potrebné zabezpečiť ochranu dát jedného používateľa pred činnosťou ostatných používateľov. Vznikol preto prepracovaný systém prístupových práv súborov a priečinkov.

Prístupové práva súborov a priečinkov

Každý súbor alebo priečinok má svojho vlastníka, t. j. používateľa, ktorý daný objekt vytvoril. Vlastník súboru alebo priečinku má právo meniť prístupové práva svojich objektov. Každý proces (úloha, program, ...) beží pod hlavičkou (menom) nejakého používateľa. Ak daný používateľ nemá právo vykonávať istú činnosť, tak ani proces bežiaci pod jeho menom toto právo nemá.


PSPad obsahuje integrovaného ftp klienta. Súbor tak môžeme otvoriť priamo zo servera. V kontextovej ponuke súboru (priečinka) nájdeme aj príkaz pre nastavenie prístupových práv.



Práva sa súborom a priečinkom prideľujú v troch úrovniach: práva vlastníka objektu, práva používateľov rovnakej skupiny ako vlastník a práva pre ostatných. Kto je vlastník už vieme. Každý používateľ patrí aj do nejakej (nejakých) skupiny. Skupina združuje niekoľko používateľov (niekedy aj všetkých). Práva pridelené skupine platia pre všetkých používateľov, ktorí sú v tejto skupine. No a posledná úroveň sú všetci ostatní. Aj im môžeme prideliť isté práva. Niektorí ftp klienti dovoľujú nastavovať a meniť práva súborov a priečinkov. Ak máme umožnený prístup k serveru cez telnet alebo SSH, môžeme využiť príkazy operačného systému (chmod, chown) a meniť prístupové práva a vlastníka súborov a priečinkov pomocou nich.

Pre našu prácu bude potrebné, aby sme so súborom, resp. priečinkom mohli pracovať my - vlastníci, naše skripty a prostredníctvom nich aj návštevníci našich stránok. Ako teda nastaviť prístupové práva a čo uvedené práva znamenajú?

Prístupové práva v operačnom systéme LINUX nastavíme linuxovým príkazom `chmod`, napr. `chmod 604 subor`.

```
gunis@rayman:~$ chmod -
v 604 subor

mode of `subor' changed
to 0604 (rw----r--)

gunis@rayman:~$
```

Poznámka: Voľbou `-v` *sme dosiahli výpis správy o zmene prístupových práv.*

Každý objekt má definované tri trojice práv, po jednej pre vlastníka, používateľov v danej skupine a všetkých ostatných. Každá trojica definuje tri práva. Ak práva boli pridelené, sú označené niektorým so znakov „rwx“. Ak práva nie sú pridelené, na príslušnom mieste je znak pomlčky. Prvá trojica sú práva vlastníka, druhá práva skupiny a tretia práva pre ostatných. Napríklad práva „`rw----r--`“ znamenajú práva „`rw-`“ pre vlastníka, práva „`---`“ pre skupinu a práva „`r--`“ pre všetkých ostatných.

Ak ide o **súbor**, práva majú nasledovný význam:

- **r** právo čítať obsah súboru,
- **w** právo zapisovať do súboru (meniť jeho obsah),
- **x** právo spúšťať súbor, pre nás (programátorov v PHP) v tejto chvíli v podstate nemá zmysel, takže ho ani nikdy nebudeme prideľovať (uvedomme si, že skripty nie sú spúšťané v operačnom systéme, ale sú interpretované PHP interpreterom).

V prípade **priečinkov** majú pridelené práva odlišný význam:

- **r** obsah priečinka je povolené vypísať,

- **w** právo zapisovať do priečinka, (vytvárať a mazať súbory v ňom),
- **x** právo vstúpiť do priečinka.

Konkrétne oprávnenie pre danú skupinu je buď pridelené, alebo nepridelené. Na práva teda môžeme nazerať ako na postupnosť núl a jednotiek. Práva `rw----r--` môžeme vyjadriť ako postupnosť `110000100`. Práva sa však zadávajú nie v dvojkovej, ale v osmičkovej sústave. Platí teda: `rw----r--⇔110000100⇔604`.

Prístupové práva vieme nastaviť aj prostredníctvom skriptu. Využijeme funkciu jazyka PHP `chmod()`. Prístupové práva očakáva funkcia `chmod()` v osmičkovej sústave. Pred číselný výraz preto doplníme znak 0.

```
chmod("subor", 0604)
```

Z pohľadu bezpečnosti platí, že to, čo nemusí byť povolené, je zakázané. Súbory a priečinky, s ktorými budú naše skripty pracovať, by preto mali mať povolené len najmenšie možné práva zabezpečujúce ich funkčnosť. Zvoliť správne prístupové práva nám pomôže nasledujúca tabuľka:

skript.php	`rw----r--`
súbor, ktorý sme vytvorili my, ale číta z neho skript	`rw----r--`
súbor, ktorý sme vytvorili my, ale číta z a zapisuje do neho skript	`rw----rw-`
súbor, ktorý vytvoril skript a číta z a zapisuje do neho skript	`rw-------`
nami vytvorený priečinok, v ktorom sú naše skripty umiestnené	`rwx-----x`
nami vytvorený priečinok, v ktorom naše skripty vytvárajú súbory	`rwx----wx`

Poznámka: Uvedomme si, že je rozdiel medzi tým, ak sme súbor vytvorili my a tým, keď súbor vytvoril náš skript. V prvom prípade sme vlastníkom súboru my, v druhom je to používateľ, pod ktorým je spustený HTTP server. Môže sa stať, že súbor, ktorý vytvorí náš skript, nebudeme môcť zmazať (lebo nie sme jeho vlastníkmi). Vieme však vytvoriť skript, ktorý tento súbor zmaže.

Každý novovytvorený súbor alebo priečinok má už vopred prednastavené prístupové práva. Aké konkrétne práva to sú, záleží od bezpečnostnej politiky správcu servera. Vždy by sme si preto mali overiť, či naše súbory a priečinky majú také práva, aké majú mať (viď. tabuľka). Ak nie, práva zmeníme.

Anketa

Prácu so súbormi si ukážeme na jednoduchom príklade webovej ankety. Celú aplikáciu rozdelíme na dve časti. Dátovú a vykonávaciu.

V dátovej časti bude samotná anketová otázka, možné odpovede a počty hlasov pre jednotlivé odpovede. Dátový súbor môže vyzerať napr. takto:

data.php

```
<?php /*
Programujete v PHP?
áno, často
0
občas
0
vôbec nie
0
*/ ?>
```

Aj keď dátový súbor neobsahuje žiadne funkčné časti PHP kódu, je uložený v súbore s príponou php. Ak by niekto zadal jeho URI do prehliadača, namiesto jeho obsahu sa zobrazí prázdna stránka, pretože jeho obsah je interpretovaný ako PHP komentár. Takto zabezpečíme, že o štruktúre našich dát sa nikto nič nedozvie.

Vykonávacia časť bude obsahovať skripty, ktoré načítajú obsah dátového súboru, vygenerujú (X)HTML kód ankety a spracujú prípadné hlasovanie zvýšením počtu hlasov pre danú odpoveď.

Obrázok 3: Zobrazenie ankety v okne prehliadača

Takéto rozdelenie (na dátovú a vykonávaciu časť) má aj ďalšiu výhodu. Pri zmene anketovej otázky nemusíme preprogramovať skript, stačí zmeniť obsah dátového súboru. Samotný skript, ktorý vygeneruje anketu a spracuje odoslaný hlas vyzerá nasledovne.

anketa.php

```
<?php
  include "hlavicka.php";       //vložíme hlavičku dokumentu
                                //obsah data.php načítame do poľa $data
  $data = @file("data.php") or
          die("Chyba v aplikácii, kontaktujte správcu.");
  $pocet = (count($data)-3) / 2;
                                // ošetrenie vstupných hodnôt $id
if (isset($_GET["id"]) and is_numeric($_GET["id"]) and
    !strpos($_GET["id"], ".")){
    $id = $_GET["id"];
    settype($id, "integer");//$id sme pretypovali na integer
    if ($id >= 1 and $id <= $pocet){
      $data[2*$id+1] = $data[2*$id+1] + 1;
                                //upravene data zapíšeme do súboru
      $ukazovatel = @fopen("data.php","w") or
                    die("Chyba v aplikácii, kontaktujte správcu.");
      flock($ukazovatel,LOCK_EX);//súbor uzamkneme pre úpravu
      for ($i = 0; $i < count($data); $i++){
        $data[$i] = trim($data[$i]);
        fwrite($ukazovatel,"$data[$i]\n");
      }
      flock($ukazovatel,LOCK_UN);//zámok súboru uvoľníme
      fclose($ukazovatel);
    }
  }

  echo "<h1>$data[1]</h1>";
  for ($i=1; $i <= $pocet; $i++){
    echo "[".trim($data[$i*2+1])."] ";
    echo "<a href=\"anketa.php?id=".($i)."\">".($data[2*$i])."</a><br
/>";
  }
  include "pata.php";          //vložíme pätu dokumentu
?>
```

Znak `@` pred volaním funkcie `file()` zabezpečí potlačenie prípadného chybového hlásenia (napr. ak súbor neexistuje).

Ak nastane chyba, jazykový konštrukt `die()` vypíše upozornenie a ukončí vykonávanie skriptu. Uvedomme si, že skript sa ukončí presne na tom mieste, kde je jazykový konštrukt `die()` vložený. V prípade chyby je v tomto prípade výsledkom nesprávne zostavený (X)HTML dokument (chýba kód generovaný skriptom `pata.php`). Tento nedostatok môžeme vyriešiť tak, že okrem samotnej chybovej správy v jazykovom konštrukte `die()` vygenerujeme aj zvyšnú časť (X)HTML dokumentu.

Do celkového počtu riadkov v súbore `data.php` započítavame aj začiatočný a ukončovací reťazec bloku PHP kódu.

`strpos("r1","r2")` vracia pozíciu prvého výskytu reťazca `r2` v reťazci `r1`. Ak `r2` sa v `r1` nenachádza, výsledkom je `FALSE`.

Súbor pomocou príkazu `fopen()` môžeme otvoriť aj v iných režimoch, napr.:
`"r"` - na čítanie,
`"a"` - pre zápis na koniec.
Pre ďalšie možnosti odporúčame manuál k príkazu `fopen()`: http://sk.php.net/manual/en/function.fopen.php.

Vysvetlime si, ako náš skript funguje. Príkazom `file()` načítame obsah súboru `data.php` do premennej `$data`. Premenná `$data` je typu pole a každý riadok súboru je samostatným prvkom poľa. Do premennej `$pocet` si uložíme počet možných odpovedí v ankete (`$pocet = (count($data)-3) / 2`).

Zaujímavá je nasledujúca podmienka. Testuje, či niekto hlasoval (`isset($_GET["id"])`) a či je potrebné spracovať hlasovanie. Zároveň otestujeme, či hodnota premennej `$_GET["id"]` je vyjadrením čísla (`is_numeric()`) bez desatinnej bodky (`!strpos($_GET["id"], ".")`). Ak podmienka platí, vieme, že niekto zahlasoval a obsahom premennej `$_GET["id"]` je zápis celého čísla (typovo je táto hodnota však stále reťazec). Pre istotu ju preto pretypujeme na celé číslo (`settype($id, "integer")`).

Nasledujúca podmienka (`$id >= 1 and $id <= $pocet`) overí, že zadané číslo je z povoleného rozsahu. Ak niekto zahlasuje v ankete, skriptu sa pošle číslo voľby, za ktorú zahlasoval (v našom prípade hodnota od 1 do 3). Ak by toto číslo bolo mimo tento interval, bude skript hlasovanie ignorovať. Ak aj druhý test zbehol v poriadku, môžeme hlas spracovať. Zvýšime počet hlasovaní za túto možnosť o 1 (`$data[2*$id+1] = $data[2*$id+1] + 1`). Nové údaje zapíšeme späť do súboru `data.php`.

Príkazom `fopen()` otvoríme súbor `data.php` pre zápis (`"w"`). Funkcia `fopen()` vracia ukazovateľ, pomocou ktorého potom môžeme so súborom pracovať. Tu ale môže nastať závažný problém. Ak by v ankete hlasovali v rovnakom čase viacerí používatelia, v jednom čase by do súboru zapisovali viaceré inštancie skriptu

anketa.php. To by zrejme „poškodilo“ jeho štruktúru. Prvá inštancia skriptu, ktorá sa pokúša do súboru zapisovať, si ho teda uzamkne výhradným zámkom (`flock($ukazovatel,LOCK_EX)`). Ostatné inštancie skriptu ostanú čakať, až kým sa zámok neuvoľní (`flock($ukazovatel,LOCK_UN)`). Obsah premennej `$data` (teraz už upravenej o započítaný hlas) zapíšeme do súboru, každý prvok poľa do samostatného riadku. Nakoniec súbor uzatvoríme (`fclose($ukazovatel)`) a vygenerujeme stránku s anketou. Ak by v ankete nebolo zahlasované, skript by vykonal len túto časť. Zvyšná časť skriptu vygeneruje nasledujúci (X)HTML kód (v hranatých zátvorkách je počet hlasovaní za danú možnosť):

kód ankety

```
<h1>Programujete v PHP?</h1>
[71] <a href="anketa.php?id=1">áno, často</a><br />
[27] <a href="anketa.php?id=2">občas</a><br />
[53] <a href="anketa.php?id=3">vôbec nie</a><br />
```

V aplikácii anketa sme vytvorili skript, ktorý zapisoval dáta do súboru. Nie však dáta od používateľa (alebo z iného systému), ale vlastné dáta. Ako ošetriť dáta z iného systému pre bezpečný zápis a opätovné zobrazenie na webovej stránke si ukážeme na nasledujúcom príklade. Naprogramujeme jednoduchú knihu návštev.

Kniha návštev

Aj aplikáciu Kniha návštev rozdelíme na dve časti, dátovú a vykonávaciu. V dátovej časti budú uložené príspevky používateľov. O obsah tejto časti sa však bude „starať“ niektorý zo skriptov z vykonávacej časti. Nevytvárame ju teda ručne my, ale náš skript.

Skôr, ako začneme vytvárať samotné skripty, mali by sme definovať aké údaje budeme od používateľa požadovať a aké údaje a v akej forme budeme ukladať do dátovej časti. Pre naše potreby si vystačíme s jednoduchou knihou návštev.

Zle navrhnutá štruktúra údajov môže značne skomplikovať naprogramovanie vykonávacej časti. Štruktúra dát závisí od toho, aké funkcie s nimi potrebujeme vykonávať. Jej voľbe by sme preto mali venovať dostatočnú pozornosť.

Používateľ zadá svoje meno (prezývku) a samotný príspevok. Meno nie je povinný údaj (umožníme aj anonymné príspevky). Príspevok je však povinný údaj, bez neho by kniha návštev zrejme nemala zmysel. Okrem mena a samotného príspevku je vhodné pamätať si aj dátum a čas príspevku. Toto samozrejme používateľ nebude zadávať, tento údaj si vie skript zistiť aj sám. Do dátového súboru budeme teda ukladať trojicu údajov, dátum a čas príspevku, meno prispievateľa a samotný príspevok. Najjednoduchšie bude, ak každý údaj uložíme do samostatného riadka dátového súboru:

kniha_dat.php *

```
<?php die("Chyba, kontaktujte správcu."); ?>
1262697940
Jano Guniš
Prvý príspevok.
1262697977
Jano Guniš
Druhý príspevok.<br />Druhý riadok.
```

Časový údaj ukladáme vo formáte UNIX timestamp.

** Význam prvého riadku skriptu si vysvetlíme neskôr.*

Hlavný skript (*kniha.php*), ktorý bude riadiť celú aplikáciu, vyzerá nasledovne.

kniha.php

```
<?php
  include "hlavicka.php";
  //ak je táto konštanta definovaná, bol volaný skript kniha.php
  define("kniha", "ok");
  //test, či bol odoslaný príspevok
  if (isset($_GET["meno"], $_GET["prispevok"])){
    //spracovanie a uloženie príspevku do súboru
    include "uloz_prispevok.php";
  }
  include "formular.php";          //zobrazenie formulára
  include "zobraz_prispevky.php"; //zobrazenie príspevkov
```

```
  include "pata.php";
?>
```

Logika skriptu je v podstate jednoduchá. Ak sú inicializované premenné `$_GET["meno"]` a `$_GET["prispevok"]`, vieme, že používateľ odoslal príspevok do knihy návštev. Skript *uloz_prispevok.php* upraví príspevok a meno a v požadovanom tvare (spolu s časovou pečiatkou) ich uloží do súboru *kniha_dat.php*. Zvyšná časť vygeneruje formulár (`include "formular.php"`) pre odosielanie príspevkov do knihy návštev a zobrazí príspevky (`include "zobraz_prispevky.php"`) uložené v súbore *kniha_dat.php*. Pozrime sa na jednotlivé vykonávacie časti podrobnejšie.

Vygenerovanie formulára zabezpečí skript *formular.php*.

formular.php

```
<h1>Kniha návštev</h1>
<form method="get" action="kniha.php">
  <label for="meno">Meno: </label>
  <input type="text" name="meno" id="meno" /><br />
  <label for="prispevok" style="color: Red;">* Príspevok: </label>
  <textarea name="prispevok" id="prispevok"></textarea><br />
  <input type="submit" value="Odošli" />
</form>
```

Všimnime si, že aj samotný formulár ukazuje, že pole pre príspevok je povinné a používateľ ho musí vyplniť.

Naša kniha návštev pôsobí veľmi stroho a nezaujímavo. V tomto prípade nám však ide skôr o funkčnosť a bezpečnosť aplikácie.

Obrázok 4: Zobrazenie knihy návštev v okne prehliadača

Spracovanie a uloženie príspevku zabezpečí skript *uloz_prispevok.php*.

uloz_prispevok.php

```
<?php
  //zabránime priamemu volaniu tohto skriptu
  if (!defined("kniha")) die("Chyba, kontaktujte správcu.");

  //úprava obsahu príspevku
  $prispevok = trim(strip_tags($_GET["prispevok"]));
  if ($prispevok != "") {
    $prispevok = str_replace("\n", "",str_replace("\r", "", nl2br($pri
spevok)));
    $meno = trim(strip_tags($_GET["meno"]));
    $meno = str_replace("\n", "",str_replace("\r", "", $meno));
      if ($meno == "") {
          $meno = "anonym";
        }
      $ukazovatel = @fopen("kniha_dat.php","a") or
                    die("Chyba v aplikácii, kontaktujte správcu.");
      flock($ukazovatel,LOCK_EX);//súbor uzamkneme pre úpravu
      fwrite($ukazovatel,time() . "\n"); //zapíšeme čas príspevku
      fwrite($ukazovatel,$meno . "\n");  //zapíšeme meno prispievateľa
      fwrite($ukazovatel,$prispevok . "\n");//zapíšeme obsah príspevku
```

```
      flock($ukazovatel,LOCK_UN); //zámok súboru uvoľníme
    fclose($ukazovatel);
  }
?>
```

Najskôr upravíme samotný príspevok. Odstránime z neho prípadné HTML a PHP značky (`strip_tags($_GET["prispevok"])`). Potom zo začiatku a z konca výsledného reťazca odstránime netlačiteľné znaky (`trim()`). Týmto zabezpečíme odstránenie medzier a znakov konca riadka zo začiatku a z konca príspevku. Ak to, čo z príspevku po tejto úprave zostalo, nie je prázdny reťazec (`$prispevok != ""`), má zmysel používateľov príspevok spracovať celý.

Rôzne operačné systémy používajú pre ukončenie riadka rôzne znaky. Napr.
MS Windows: „\r\n“,
LINUX: „\n“,
Mac OS: „\r“.

Používateľ mohol zadať viacriadkový príspevok. Ak by sme takýto príspevok uložili do súboru *kniha_dat.php*, pokazili by sme si jeho štruktúru (ukladáme obsah celého príspevku do jedného riadka). Znaky konca riadka z príspevku odstránime (`str_replace()`) a nahradíme ich (X)HTML značkou `<br />`(`nl2br()`). Rovnako upravíme aj reťazec v premennej `$_GET["meno"]`. Ak by obsah premennej `$meno` zostal prázdny reťazec, uložíme do nej reťazec `"anonym"` (`$meno = "anonym"`).

Funkcia `str_replace("čo","čím","v čom")` nahradí reťazec `"čo"` reťazcom `"čím"` v reťazci `"v čom"`.

Súbor *kniha_dat.php* otvoríme pre zápis na koniec (`@fopen("kniha_dat.php","a")`) a všetky tri hodnoty do súboru zapíšeme. Každú do samostatného riadku (pridaním reťazca `"\n"`).

Skript *zobraz_prispevky.php* prečíta dáta zo súboru a vygeneruje (X)HTML kód pre zobrazenie príspevkov.

zobraz_prispevky.php

```
<?php
  //zabránime priamemu volaniu tohto skriptu
  if (!defined("kniha")) die("Chyba, kontaktujte správcu.");

  if (!file_exists("kniha_dat.php")){ //kniha je volaná prvý krát
    $ukazovatel = @fopen("kniha_dat.php","w") or
                  die("Chyba v aplikácii, kontaktujte správcu.");
    flock($ukazovatel,LOCK_EX);//súbor uzamkneme pre úpravu
    fwrite($ukazovatel,"<?php die(\"Chyba, kontaktujte správcu.\"); ?>
 \n");
    flock($ukazovatel,LOCK_UN);//zámok súboru uvoľníme
    fclose($ukazovatel);
  }
  else {
    date_default_timezone_set("Europe/Bratislava");
    $prispevky = file("kniha_dat.php");
    for ($i=1; $i <= (count($prispevky)-1) / 3; $i++){
      echo "<p>" . date("d.n.Y H:i",trim($prispevky[$i*3-2]));
      echo " " . $prispevky[$i*3-1] . "<br />";
      echo $prispevky[$i*3] . "</p>";
    }
  }
?>
```

V prípade, že skript *kniha.php* je zobrazený prvý krát, súbor *kniha_dat.php* ešte neexistuje (`!file_exists("kniha_dat.php")`). V tomto prípade ho skript vytvorí. Ak súbor už existoval, tak zvyšná časť skriptu načíta obsah dátového súboru do poľa (`$prispevky = file("kniha_dat.php")`) a postupne vypíše jeho obsah. Počet príspevkov v súbore je `(count($prispevky)-1) / 3`, každý príspevok je uložený v troch riadkoch – časová pečiatka, meno a obsah samotného príspevku.

Ak by prípadný záškodník zadal URI skriptu *uloz_prispevok.php* ešte predtým ako sa dátový súbor vytvorí, pokazil by tak jeho štruktúru.
V súbore by chýbal prvý riadok, ktorý tam zapíše skript *zobraz_prispevky.php* pri prvom zavolaní knihy návštev.

Na prvý pohľad by sa mohlo zdať, že bezpečnosť skriptu sme dostatočne ošetrili. Vstupy sme upravili na bezpečný tvar a až potom ich zapísali do dátového súboru. Čo by sa však stalo, ak by si niekto vyžiadal v URI meno niektorého zo skriptov? Keďže tieto skripty sú vkladané do hlavného skriptu len pri splnení určitých podmienok a v určitej situácii, mohli by nastať problémy (či už s funkčnosťou alebo s bezpečnosťou). Aby sa tak nestalo, zabránime tomu. V hlavnom skripte sme

definovali konštantu `kniha` (`define("kniha", "ok")`). Každý so skriptov si najskôr overí, či táto konštanta existuje (`defined("kniha")`). Ak nie, vieme, že skript bol zavolaný priamo, a jeho činnosť sa ukončí `die()`. Všimnime si, že aj samotný dátový dokument nedovolí, aby si niekto priamo cez jeho URI zobrazil jeho obsah.

Ukladať údaje o návštevách do samostatných riadkov je dosť nepraktické. Výhodnejšie by bolo, ak by všetky údaje o jednej návšteve boli uložené v jedinom riadku. Všetky údaje môžeme spojiť do jedného reťazca, s príslušným oddeľovacím reťazcom, a ten potom uložiť do súboru.
Napr.
```
$ret = "čas|||IP";
```
Takýto reťaze potom pomocou príkazu `explode()` rozdelíme do poľa:
```
$pole = explode("|||", $ret);
echo $pole[0]; //cas
echo $pole[1]; //IP
```

Zadanie 7.1

Vytvorte jednoduché počítadlo návštev vašej webovej stránky. Pri každej návšteve skript zapíše do súboru čas návštevy.

Evidovať môžete aj iné informácie, napr.:

stránku odkiaľ používateľ prišiel: `$_SERVER['HTTP_REFERER']`,

jeho IP adresu: `$_SERVER['REMOTE_ADDR']`,

informácie o prehliadači: `$_SERVER['HTTP_USER_AGENT']` a pod.

Viac informácií nájdete vo výpise príkazu `phpinfo()`.

Vytvorte skript *statistika.php*, ktorý dáta so súboru vypíše v prehľadnej štatistike.

Zadanie 7.2

Aplikáciu Anketa by sme radi umiestnili na viaceré webové stránky, samozrejme vždy s inou otázkou a inými možnosťami odpovede. Nakopírovať kód skriptu pre každú webovú stránku je síce možné, ale nie veľmi praktické (ak by sme chceli v kóde niečo zmeniť, museli by sme upraviť každú kópiu skriptu). Upravte skript tak, aby sme jeden a ten istý kód (v jednom skripte) vedeli využiť pre rôzne ankety.

Návod: Pri volaní skriptu ankety posielajme aj identifikátor súboru, v ktorom je dátová časť ankety uložená.

Zadanie 7.3

Upravte aplikáciu Kniha návštev tak, aby výstup bol „krajšie“ formátovaný. Využite CSS.

Zadanie 7.4

Knihu návštev sme naprogramovali tak, že najnovšie príspevky sú zobrazené na konci stránky. Upravte knihu návštev tak, aby najnovšie príspevky boli vždy zaradené na začiatku.

Čo sme sa naučili

Naučili sme sa pracovať so súbormi ako efektívnym nástrojom na trvalé ukladanie dát na serveri. Vieme ako nastaviť správne prístupové práva skriptov, súborov, ktoré vytvárajú a priečinkov, v ktorých sú skripty umiestnené. Ukázali sme si ako vytvoriť jednoduché aplikácie využívajúce súbory na serveri. Vieme ako bezpečne ošetriť dáta od používateľa a ako ich upraviť tak, aby sme zachovali požadovanú štruktúru dátového súboru. Poznáme spôsoby ako zabrániť požiadavke na priame spustenie skriptu, ktorý je určený pre vloženie do hlavného skriptu.

Kapitola 8: Vlastné funkcie

Ak na viacerých miestach našich skriptov vykonávame rovnakú alebo veľmi podobnú postupnosť príkazov, je výhodné túto postupnosť „osamostatniť“. Definovať ju samostatne ako vlastnú funkciu. Takéto usporiadanie kódu má aj ďalšiu výhodu, prispieva k sprehľadneniu kódu. Ľahšie sa v ňom hľadajú chyby a robia neskoršie úpravy. Je pravdepodobné, že jednu a tú istú funkciu budeme používať vo viacerých skriptoch. V tomto prípade je vhodné funkcie uložiť do samostatného dokumentu a ten potom na príslušné miesto (zväčša na začiatok skriptu) vložiť pomocou `include()`.

Definícia funkcie obsahuje kľúčové slovo `function`, za ktorým nasleduje meno funkcie a okrúhle zátvorky. Samotné telo funkcie je uzatvorené v zložených zátvorkách `{}`.

Názov funkcie je ľubovoľná postupnosť písmen, číslic a podčiarkovníka začínajúca písmenom alebo podčiarkovníkom.

Aj keď PHP umožňuje aj obrátený postup (najskôr v skripte zavolať funkciu a až potom ju zadefinovať), neodporúčame to.

```
//volanie funkcie
menoFunkcie();

//..., príkazy skriptu
function menoFunkcie(){
  //telo funkcie
}
```

Kód skriptu by sa stal neprehľadným. Zvyknime si preto funkcie definovať vždy na začiatku skriptu, prípadne v samostatnom súbore a do skriptu ich vložiť pomocou `include()`.

definícia funkcie

```
function meno () {
  //telo funkcie
}
```

```
function pisNadpis () {
  echo "<h1>Nadpis</h1>";
}
//... príkazy skriptu
//volanie funkcie
pisNadpis();
```

Vykonanie tela funkcie zabezpečíme volaním jej názvu. (napr. `pisNadpis()`).

Sila funkcií však spočíva najmä v tom, že im môžeme posielať hodnoty, ktoré majú spracovať. Súčasťou definície funkcie môže byť zoznam parametrov, pomocou ktorých funkcii predávame hodnoty, ktoré má spracovať.

definícia funkcie s parametrami

```
function meno ($par1, $par2, ...) {
  //telo funkcie
}
```

```
function pisNadpis ($text) {
  echo "<h1>$text</h1>";
}
//..., príkazy skriptu
//volanie funkcie
pisNadpis("Nadpis");
```

V našich predchádzajúcich skriptoch sme často testovali, či zadaná hodnota predstavuje celé číslo (prípadne iný typ hodnoty, resp. hodnotu z daného rozsahu). Oplatí sa preto vytvoriť si funkciu, ktorá danú hodnotu otestuje. Od takejto funkcie však očakávame nie len to, že zadanú hodnotu otestuje, ale aj to, že o výsledku testu nás bude spätne informovať. V tele funkcie môžeme definovať návratovú hodnotu (`return hodnota`). Ako náhle PHP interpreter narazí na `return` v tele funkcie, vykonávanie tela funkcie sa ukončí a hlavnému skriptu je vrátená `hodnota`.

Funkcia testujúca, či zadaná hodnota je vyjadrením celého čísla, vyzerá nasledovne:

funkcia jeCele()

```
function jeCele($data){
  if (is_int($data)){//$data je typu celé číslo
    return TRUE;
  }                  //$data je číselný reťazec
  elseif (is_string($data) and is_numeric($data)){
    return (strpos($data, '.') == FALSE); //test na výskyt des. bodky
  }
  return FALSE;
}
```

Od PHP verzie 5.2.0 môžeme využívať filter pre kontrolu a úpravu dát. Test celého čísla by vyzeral nasledovne:

```
function jeCele($data){
  if(!filter_var($data,
 FILTER_VALIDATE_INT)){
    return FALSE;
  }
  else {
    return TRUE;
  }
}
```

Najskôr otestujeme, či zadaná hodnota je typu celé číslo (`is_int($data)`). Ak táto podmienka neplatí, testujeme ďalšiu z možností. Zadaná hodnota môže byť reťazec vyjadrujúci zápis čísla (`is_string($data) and is_numeric($data)`) Tejto podmienke však vyhovujú aj reálne čísla, otestujeme preto výskyt desatinnej bodky v reťazci (`strpos($data, '.') == FALSE`).

Volanie funkcie `jeCele()` v hlavnom skripte by mohlo vyzerať napríklad takto:

volanie funkcie jeCele()

```
//... kód skriptu
if (jeCele($hodnota)) {
  echo "celé číslo";
}
else {
  echo "NIE celé číslo";
}
```

Použitie funkcie `jeCele()` predstavuje bezpečnostný mechanizmus nazývaný whitelist (biely zoznam). Ak dáta spĺňajú vstupné požiadavky (napr. je celé číslo) spracujú sa. V opačnom prípade sa „zahodia“ a skript ich neakceptuje.

Rozsah platnosti premenných

V skriptoch, ktoré sme doteraz vytvárali, je premenná s rovnakým názvom stále jedna a tá istá premenná s tou istou hodnotou. Toto ale neplatí v tele funkcie.

platnosť pemenných

```
//... kód skriptu
  function pis (){
    echo $premenna;
  }
  $premenna = "hodnota";
  pis();
//... kód skriptu
```

V iných jazykoch je potrebné premenné deklarovať. Takže aj keď deklarujeme vo funkcii premennú s rovnakým názvom ako má premenná v hlavnom programe, zmena hodnoty lokálnej premennej neovplyvní hodnotu rovnomennej premennej v hlavnom programe.

Výsledkom behu uvedenej časti kódu bude hlásenie PHP interpretera o použití neinicializovanej premennej `$premenna`. Na rozdiel od iných jazykov (Pascal, C a pod.) globálne premenné nie sú automaticky prístupné vo funkciách. Na prvý pohľad to vyzerá ako nevýhoda jazyka PHP. Opak je však pravdou. Ak by tomu tak nebolo, veľmi ľahko by sme mohli v tele funkcie nechtiac prepísať hodnotu nejakej globálnej premennej. Uvedomme si, ako sa PHP skripty pomocou niekoľkých `include()` „skladajú“ dohromady. Ustriehnuť názvy všetkých lokálnych premenných by bolo prakticky nemožné. Ak si to však situácia vyžaduje, globálne premenné vieme sprístupniť aj v tele funkcie.

platnosť pemenných

```
//... kód skriptu
  function pis (){
    global $premenna;
    echo $premenna;
  }
  $premenna = "hodnota";
  pis();
//... kód skriptu
```

Kľúčové slovo `global` sa nepoužíva na sprístupnenie premenných definovaných v samostatných externých skriptoch, ktoré do hlavného skriptu vložíme pomocou `include()`. Kód vložený pomocou `include()` PHP interpretuje rovnako, ako keby sme tento kód do hlavného skriptu priamo napísali (namiesto jeho vloženia).

Pomocou kľúčového slova `global` sme v tele funkcie sprístupnili globálnu premennú `$premenna`. Aj keď nám to PHP umožňuje, mali by sme byť pri používaní globálnych premenných vo funkciách opatrní. Najlepšie je však sa tomuto kroku vyhnúť.

Použitie funkcií si ukážeme na príklade skriptu, ktorý vygeneruje kalendár pre zadaný mesiac a rok.

Kalendár

Aplikáciu rozdelíme do niekoľkých skriptov: skript pre vygenerovanie formulára, skript s funkciami potrebnými pre kontrolu vstupov a samotný riadiaci skript.

Skript s formulárom môže vyzerať nasledovne.

formular_kalendar.php

```
<h1>Vyberte si mesiac a zadajte rok</h1>
<form action="" method="post">
<label for="mesiac">Mesiac: </label>
<select name="mesiac" id="mesiac">
<?php
```

```
  for ($i=1; $i<=12; $i++){
      echo "<option value=\"$i\">$i</option>";
    }
?>
</select>
<label for="rok">Rok: </label>
<input type="text" name="rok" id="rok" />

<input type="submit" value="zobraz kalendár">
</form>
```

Všimnime si, že generovanie formulára sme si uľahčili skriptom. Nemusíme tak písať dvanásť krát časť `<option>` pre každý mesiac.

Formulár odošle číslo mesiaca a rok. V skripte by sme mali skontrolovať, či zaslané hodnoty zodpovedajú tomu, čo očakávame. Mesiac je celé číslo z intervalu <1, 12>, rok je tiež celé číslo a jeho rozsah môžeme obmedziť intervalom <1900, 2100>. Využijeme to, že funkciu (`jeCele()`) pre testovanie hodnoty na celé číslo už máme. Doprogramujeme si funkciu (`jeCeleVRozsahu()`), ktorá okrem hodnoty na celé číslo overí, či číslo je zo zadaného intervalu. Funkcie si uložíme do súboru *funkcie_kalendar.php*.

funkcie_kalendar.php

```
<?php
function jeCele($data){
  if (is_int($data)){//$data je typu celé číslo
    return TRUE;
  }                   //$data je číselný reťazec
  elseif (is_string($data) and is_numeric($data)){
    return (strpos($data, '.') == FALSE); //test na výskyt des. bodky
  }
  return FALSE;
}

function jeCeleVRozsahu($data, $min, $max){
  if (jeCele($data)) {
    return $min <= $data and $data <= $max;
  }
  else {
    return FALSE;
  }
}
?>
```

Funkcia `jeCeleVRozsahu()` využíva funkciu `jeCele()`. Takýmto „skladaním“ jednoduchých funkcií vieme naprogramovať aj pomerne náročné funkcie. Navyše, viac „menších“ funkcií je flexibilnejších, než jedna veľká funkcia.

Ak si nájdeme trochu času a pozrieme sa do manuálu jazyka PHP, zistíme, že väčšina funkcií (príkazov) PHP robí len jednu vec. Ale robí ju PORIADNE.

Hlavný riadiaci skript *kalendar.php* môže vyzerať nasledovne.

kalendar.php

```
<?php
  include "funkcie_kalendar.php";
  include "hlavicka.php";
  if (isset($_POST["mesiac"], $_POST["rok"]) and
      jeCeleVRozsahu($_POST["mesiac"], 1, 12) and
      jeCeleVRozsahu($_POST["rok"], 1900, 2100)) {
     $mesiac = $_POST["mesiac"];
     $rok = $_POST["rok"];
     $dni = @cal_days_in_month(CAL_GREGORIAN, $mesiac, $rok);
     echo "<h1>Mesačný kalendár: $mesiac/$rok</h1>";
     for ($i=1; $i<=$dni; $i++) {
       echo "[$i. $mesiac. $rok] : ......................<br />";
     }
     echo "Kalendár pre <a href=\"kalendar.php\" title=\"kalendár pre
iný mesiac\">iný mesiac</a>.";
  }
  else {
```

Funkcia `cal_days_in_month(CAL_GREGORIAN, $mesiac, $rok)` vráti počet dní zadaného mesiaca a roku podľa Gregoriánskeho kalendára. Tento údaj by sme mohli zistiť aj ručne, takto je to ale pohodlnejšie.

```
      include "formular_kalendar.php";
    }
  include "pata.php";
?>
```

Skript *kalendar.php* funguje v dvoch režimoch: v režime, keď zobrazí formulár a v režime, keď spracuje dáta z formulára a vygeneruje príslušný kalendár. Vďaka častiam kódu, ktoré sme umiestnili do iných skriptov, a funkciám je jeho kód ľahko čitateľný a pochopiteľný.

Zadanie 8.1	V aplikácii Kniha návštev sme ošetrili používateľské vstupy (`$meno`, `$prispevok`) pre bezpečný zápis do súboru. Vytvorte funkciu, ktorá v aplikácii Kniha návštev zadaný reťazec ošetrí rovnakým spôsobom. Funkcia vráti ako svoju návratovú hodnotu upravený reťazec.
Zadanie 8.2	V našich skriptoch sme používali príkaz `include "hlavicka.php";` na vloženie hlavičky (X)HTML dokumentu. Tento spôsob má však jednu nevýhodu, všetky hlavičky sú rovnaké (dokumenty majú rovnaký titulok). Vytvorte funkciu hlavicka($titulok), ktorá do dokumentu vloží (X)HTML hlavičku s príslušným titulkom, napr. hlavicka("Webová anketa").
Zadanie 8.3	V kapitole 4 sme naprogramovali jednoduchý web. Riešením predchádzajúceho zadania vieme pre každú stránku nášho webu vygenerovať jej unikátny titulok. Súčasťou „dobrých“ webov býva aj dátum ostatnej zmeny dokumentu. Upravte dokument *pata.php* tak, aby okrem korektného ukončenia (X)HTML dokumentu vygeneroval aj dátum zmeny aktuálneho dokumentu. Návod: Vytvorte funkciu pata($dokument), ktorá do dokumentu, okrem korektného ukončenia (X)HTML dokumentu, vloží aj dátum zmeny príslušného dokumentu. Časovú pečiatku modifikácie súboru vráti funkcia *filemtime*("nazov suboru").

Čo sme sa naučili

Naučili sme sa definovať vlastné funkcie. Vieme aké výhody prinášajú a ako ich používať. Vieme vytvárať funkcie bez parametrov, ale aj s parametrami. Vieme vzájomne kombinovať viacero funkcií. Poznáme obmedzenia použitia globálnych premenných v tele funkcie, ale vieme ho obísť, vedomí si rizika s tým súvisiacim.

Kapitola 9: Bezpečnosť aplikácií založených na PHP

Otázkam bezpečnosti sme sa čiastočne venovali už priebežne. V tejto kapitole sa na bezpečnosť aplikácií založených na PHP pozrieme bližšie. Oboznámime sa s najčastejšími bezpečnostnými problémami a naučíme sa ako ich eliminovať.

Dynamické webové stránky napriek svojim nesporným výhodám prinášajú aj množstvo problémov súvisiacich s bezpečnosťou. Naše aplikácie budú na základe požiadaviek používateľov modifikovať dáta na serveri (či už v súbore alebo v databáze), spracovávať používateľské vstupy, spracovávať dáta z viacerých subsystémov alebo generovať výstupy na základe požiadaviek používateľov. Vstup od jedného používateľa do nami vytvorenej aplikácie môže byť výstupom pre iného používateľa (napr. kniha návštev). Na serveri môže mať svoje konto viacero používateľov. „Nedbalo“ napísaná aplikácia tak môže nepriamo ohroziť návštevníkov našich stránok alebo majiteľov stránok na rovnakom serveri. Otázka bezpečnosti je tu teda podstatná.

Neinicializované premenné

Používanie neinicializovaných premenných nepatrí k najlepším programátorským zvykom. Aj keď PHP do takejto premennej priradí prázdny reťazec, nie je dobré sa na to spoliehať. V konfigurácii starších verzií PHP (<=4.2.3) bola direktíva register_globals štandardne nastavená na hodnotu On. Dôsledkom tohto nastavenia bolo, že neinicializovanej premennej sme mohli podstrčiť ľubovoľnú hodnotu v URI skriptu.

http://localhost/premenna.php?pom=hodnota (register_globals = On)

```
<?php
  include "hlavicka.php";
  echo $pom; // hodnota
  include "pata.php";
?>
```

Aj keď takéto nastavenie PHP umožňuje jednoduchý prístup k premenným posielaným metódou GET, je s ním spojené pomerne veľké riziko podstrčenia hodnoty premennej. Uvedomili si to aj tvorcovia PHP a direktíva register_globals je v nových verziách PHP štandardne nastavená na Off. Stále však nájdeme množstvo serverov, ktoré bežia na starších verziách PHP alebo ktorých správcovia povolili register_globals na On.

Vstupy z iného systému

Neošetrenie vstupu z iného systému (vstupy používateľa, z nie vlastného súboru a pod.) je najčastejším rizikom, s ktorým sa tvorcovia dynamických webových stránok môžu stretnúť. Naprogramovať aplikáciu, ktorá bude fungovať pre korektné vstupy, je pomerne jednoduché, pretože vieme, aké vstupy očakávame. Ošetriť aplikáciu pred nekorektným vstupom už môže byť značne problematické, lebo len pomerne ťažko dokážeme odhadnúť aké vstupy prípadný útočník môže zadávať. PHP našťastie ponúka na výber množstvo funkcií, ktoré môžeme využiť.

Funkcia `strip_tags()` odstráni z reťazca (X)HTML a PHP značky. Využijeme ju, ak potrebujeme reťazec zobraziť ako textovú súčasť webovej stránky.

strip_tags()

```
<?php
  include "hlavicka.php";
  $text = "<p>Odsek textu.</p><!-- Komentár -->
           <a href='URI'>Odkaz</a> <?php include 'subor.php' ?>";
  echo strip_tags($text); //Odsek textu. Odkaz
  include "pata.php";
?>
```

V PHP nájdeme aj funkciu `htmlentities()` ktorá prevedie všetky možné znaky na HTML entity.

Niekedy je však takáto úprava nevhodná (napr. v diskusnom fóre o tvorbe webu). Vtedy môžeme využiť funkciu `htmlspecialchars()`, ktorá prevedie špeciálne znaky (&, <, >, ", ') na HTML entity. Prehliadač ich bude interpretovať ako obyčajné znaky

bez špeciálneho významu.

V tomto prípade môžeme použiť aj funkciu `strip_tags()`. Záleží na tom, či chceme kód skriptu zobraziť alebo ho odstrániť.

htmlspecialchars ()
```
<?php
  include "hlavicka.php";
  $retazec="<img src=\"http://stranka.sk/obr.gif\" />";
  echo htmlspecialchars($retazec);
  //&lt;img src=&quot;http://stranka.sk/obr.gif&quot; /&gt;
  include "pata.php";
?>
```

Ošetrenie vstupov (a nedôvera k nim) je základom ochrany voči väčšine typov útokov. Presvedčíme sa o tom v nasledujúcich častiach.

Cross-site scripting (XSS)

Cross-site scripting predstavuje jeden z najrozšírenejších, ale zároveň najviac podceňovaných útokov. Prípadný útočník pri ňom zneužíva nedostatočné ošetrenie používateľských vstupov na strane servera (viac v predchádzajúcej kapitole). Pomerne ľahko potom dokáže serveru podstrčiť kus „nebezpečného“ kódu, ktorý ho potom posiela nič netušiacim používateľom. Pomocou XSS vie útočník prepašovať na cudzie stránky nie len nevhodný (X)HTML kód, ale aj funkčné skripty.

„Podstrčený“ kus kódu môže obsahovať nielen časti (X)HTML, ale aj funkčné časti skriptov (napr. JavaScript).

Cross-site scripting
```
<?php
  include "hlavicka.php";
  $retazec="<script type=\"text/javascript\">
            //<![CDATA[
            document.location.replace (\"http://www.podstrcena.sk\");
            //]]>
            </script>";
  echo htmlspecialchars($retazec);
  /* &lt;script type=&quot;text/javascript&quot;&gt;
     //&lt;![CDATA[
     document.location.replace (&quot;http://www.podstrcena.sk&quot;);
     //]]&gt;
     &lt;/script&gt; */
  include "pata.php";
?>
```

Ak by sa vyššie uvedený kód zobrazil bez úpravy na webovej stránke, spôsobil by automatické presmerovanie zobrazenej stránky na podstrčenú, podvodnú stránku (`http://www.podstrcena.sk`).

Z hľadiska bezpečnosti je lepšie nekorektné dáta „zahodiť“, než sa pokúšať dostať ich do prijateľnej (korektnej) podoby. Prípadný (šikovný) útočník nám totiž môže podstrčiť dáta v takom tvare, že pri pokuse o ich úpravu do korektného tvaru vytvoríme potenciálne nebezpečný kus kódu.

V konfiguračnom súbore PHP je direktíva `allow_url_include` štandardne nastavená na `Off`. Vložiť súbor zo zadanej URI tak nie je umožnené. Spoliehať sa na nastavenie servera sa však neodporúča. Rovnako nebezpečné môže byť aj nesprávne použitie príkazu `require()`.

Problém s príkazom include()

S príkazom `include()` sme sa už stretli. Vďaka nemu vieme výslednú podobu webovej stránky „vyskladať“ z jednotlivých častí uložených v samostatných dokumentoch. Jednoduchým riešením by teda bolo, aby všetky odkazy v rámci nášho webu mali tvar www.domena.sk/index.php?subor=strankaxy.php. Samotná časť kódu by potom vyzerala nasledovne:

include(), nesprávne použitie: www.domena.sk/index.php?subor=strankaxy.php
```
<?php
  include "hlavicka.php";
  include $_GET["subor"];
  include "pata.php";
?>
```

Na prvý pohľad sa to javí ako elegantné a jednoduché riešenie. Prichádza však útočník a našu stránku si zobrazí v tvare www.domena.sk/index.php?subor=http://www.mojadomena.sk/skript.txt. Náš skript však dôverčivo vloží na príslušné miesto obsah súboru skript.txt. Jeho kód (ak ho obsahuje) sa navyše aj vykoná. Takto útočník v podstate dosiahol to, že na našom serveri, na ktorý nemá prístup, môže spúšťať vlastné skripty. Ak by sa útočníkovi podarilo odhaliť názvy niektorých našich dátových súborov (napr. hesla.dat), nič mu nebráni v tom, aby si nechal zobraziť obsah týchto súborov.

Aj keď súbor skript.txt obsahuje funkčný PHP skript, má príponu, ktorá nie je asociovaná s PHP interpreterom. Takto útočník docieli to, že jeho server odošle samotné telo skriptu a nie výsledok jeho behu. Samotný skript sa potom vykoná na „napadnutom“ serveri.

Riešením tohto bezpečnostného rizika je neposielať názov súboru, ale jeho identifikátor.

include(), správne použitie: www.domena.sk/index.php?id=xy

```
<?php
  include "hlavicka.php";
  switch ($id){
    case 1: include "prva.php";
            break;
    case 2: include "druha.php";
            break;
    default:include "uvod.php";
  }
  include "pata.php";
?>
```

Takto sme dosiahli to, že príkazom `include()` nevložíme žiaden cudzí súbor do našej stránky (resp. skriptu). Toto riešenie má aj ďalšiu výhodu - útočník nevie nič o názve našich súborov. Čím menej o vnútornej štruktúre nášho webu útočník vie, tým lepšie pre nás.

Rozdeliť výkonovú časť do niekoľkých samostatných skriptov skrýva aj ďalšie nebezpečenstvo. Pri ich vytváraní totiž predpokladáme, že budú volané z hlavného skriptu. Ak by sa však útočník dozvedel názov niektorého zo skriptov, mohol by ho spustiť priamo cez jeho URI. To by však mohlo spôsobiť problémy (napr. zápis neošetrených dát do súboru a pod.). Skripty, ktoré nie sú určené na samostatné spustenie, by sme mali voči tejto možnosti ošetriť. Riešením je napr. kontrola existencie konkrétnej konštanty (`defined("konstanta")`), ktorú sme vytvorili len v hlavnom skripte (`define("konstanta", "ok")`).

Chybové správy PHP interpretera

Na začiatku tohto modulu sme prepli konfiguráciu PHP interpretera do režimu Development. Jedným z dôsledkov tohto kroku bolo, že interpreter začal vypisovať aj menej závažné chyby (varovania, poznámky). Informácie o týchto chybách sú užitočné počas ladenia skriptov. V ostrej prevádzke by sme ich mali potlačiť (ideálne je naprogramovať skript tak, aby žiadne chyby, varovania a poznámky negeneroval :-). Z týchto hlásení interpretera sa totiž prípadný útočník môže dozvedieť informácie, ktoré mu môžu pomôcť napadnúť našu aplikáciu.

Nie vždy však máme možnosť zmeniť konfiguráciu servera. Pomôcť si môžeme príkazom `error_reporting()`, ktorý dočasne (počas behu skriptu) zmení úroveň chýb, ktoré bude PHP interpreter hlásiť. Funkcia `ini_set("display_errors","On")` nastaví, že chybové hlásenia sa budú vypisovať ako súčasť výstupu skriptu.

Úrovne 0 a -1 sú dve krajné možnosti. Pre nastavenie ostatných úrovní odporúčame nahliadnuť do manuálu jazyka PHP http://sk.php.net/manual/en/function.error-reporting.php.

error_reporting()

```
<?php
  include "hlavicka.php";
  ini_set("display_errors","On"); // povolené zobrazovanie chýb
  error_reporting(0);  // žiadne chyby nie sú hlásené
  echo $a;
  error_reporting(-1); // všetky chyby sú hlásené
  echo $b;
  include "pata.php";
?>
```

Žiadna z premenných (`$a`, `$b`) nie je inicializovaná. V prípade premennej `$a` sa však táto poznámka vo výstupe skriptu nezobrazí.

Keďže `file("subor")` vracia `FALSE`, pokračuje PHP interpreter vyhodnotením druhej časti logického výrazu. Tá však obsahuje jazykový konštrukt `die()`, ktorý spôsobí výpis hlásenia a ukončenie vykonávania skriptu.

Ak v skripte použijeme funkciu, pri volaní ktorej môže dôjsť k chybe (napr. otvorenie neexistujúceho súboru), môžeme toto chybové hlásenie potlačiť a programovo správne zareagovať. Stačí, ak pred volanie takejto funkcie napíšeme znak `@`. Využiť môžeme fakt, že neúspešne vykonaná funkcia vracia hodnotu `FALSE`.

@, die()
```
<?php
  include "hlavicka.php";

  //potlačené chybové hlásenie, vykonávanie skriptu pokračuje
  $f = @file("subor");
  if (!$f) {
      echo "Chyba v aplikácii, kontaktujte správcu.";
  }

  //potlačené chybové hlásenie, vykonávanie skriptu je ukončené
  $f = @file("subor") or
       die("Chyba v aplikácii, kontaktujte správcu.");
  include "pata.php";
?>
```

V predchádzajúcom skripte sme sa dvakrát pokúsili otvoriť neexistujúci súbor. V obidvoch prípadoch sme chybové hlásenie funkcie `file()` potlačili pridaním znaku `@` pred jej názov. V prvom prípade sme sa obmedzili len na výpis chybového hlásenia. V druhom prípade sme využili jazykový konštrukt `die()`. Jeho výsledkom je výpis chybového hlásenia a ukončenie vykonávania skriptu. Ktorú z možností použijeme záleží od závažnosti vzniknutej chyby a schopnosti našej aplikácie pokračovať ďalej. V obidvoch prípadoch sme sa obmedzili len na jednoduché chybové hlásenie bez podrobností (ktoré by mohol prípadný útočník zneužiť).

Chybové správy PHP interpretera sú užitočné počas ladenia skriptov. V ostrej prevádzke by sa nemali objavovať. Používateľ si však „zaslúži“ byť informovaný, ak nastal nejaký problém počas behu skriptu. Informácia, ktorú mu skript vypíše, by však nemala prezrádzať zbytočne veľa o fungovaní nášho webu.

Ak je to možné, tak bezpečnejší spôsob uloženia mena a hesla je do databázy. Už samotný prístup do databázy je chránený prístupovým menom a heslom. Navyše prístup k údajom v databáze môže byť obmedzený len na konkrétny počítač.

PHP si najlepšie rozumie so systémom riadenia bázy dát MySQL. MySQL je súčasťou Uniform Servera. Prevziať si ho môžeme aj so stránky http://www.mysql.com/.

Šifrovanie a heslá

Ak vytvárame rozsiahlejšie aplikácie a potrebujeme nejakým spôsobom kontrolovať kto má k akým dátam povolený prístup, asi sa nevyhneme autentifikácii pomocou mena a hesla. Meno a heslo bude zrejme uložené v súbore (ktorý nie je možné priamo zobraziť). Ak používateľom zadané meno a heslo zodpovedajú nejakému riadku súboru, môžeme zobraziť príslušnú časť neverejných dát. Za normálnych okolností sa k súboru s heslami nedostane nikto nepovolaný, takže heslá sú v bezpečí. Nikdy si však nemôžeme byť istí, že k takejto situácii nedôjde (napr. XSS, chybné použitie include() a pod.). Mali by sme preto spraviť všetko preto, aby prípadný útočník mal čo najmenej možností ako tieto informácie zneužiť. Jednou z možností je heslo šifrovať.

PHP nám ponúka tzv. hashovacie funkcie. Hashovacia funkcia je funkcia, ktorá pre ľubovoľne dlhý znakový vstup vygeneruje iný reťazec pevnej dĺžky, tzv. odtlačok. Tento odtlačok spĺňa nasledovné vlastnosti:

- pri rovnakom vstupe dostávame rovnaký výstup,
- z odtlačku nie je možné matematicky vytvoriť pôvodný reťazec,
- nie je možné matematicky zostrojiť dva rôzne reťazce, ktorých odtlačky by boli zhodné.

Hashovacie funkcie sú navyše vymyslené tak šikovne, že aj malá zmena v pôvodnom reťazci spôsobí veľkú zmenu v odtlačku reťazca. Pozrime sa na výsledok hashovacej funkcie `sha1()`.

sha1()

```php
<?php
  include "hlavicka.php";
  echo sha1("heslo1"); //21ad1ea473a1fbfbb4f5a192f4faaf9637143c6f
  echo sha1("heslo2"); //0d2203fc419914e07dbe2389b3f9fe4b4307aeca
  include "pata.php";
?>
```

Riešením teda je neukladať heslá, ale ich odtlačky. Pri autentifikácii potom zisťujeme, či odtlačok zadaného hesla sa zhoduje s odtlačkom, ktorý je uložený v súbore.

Bohužiaľ, ani toto nestačí. Predstavme si, že útočník sa zaregistruje (zadá do systému svoje meno a heslo). Keďže hashovacia funkcia vracia pre rovnaké vstupy rovnaké výstupy, stačí nájsť v súbore hesiel rovnaký odtlačok ako má čerstvo zaregistrovaný útočník. Takto získa zoznam používateľov, ktoré majú rovnaké heslo ako on. Takže je potrebné zabezpečiť, aby aj pre rovnaké heslá bol výsledok hashovania rôzny (hovorí sa tomu salting). Môžeme to urobiť napr. tak, že do súboru nebudeme ukladať len odtlačok hesla, ale napr. odtlačok hesla spojeného s prihlasovacím menom. Usporiadaná dvojica [meno, heslo] je zrejme jedinečná v celom systéme.

Je to silné, ale stále to má trhliny. Predpokladajme, že útočník sa dostane k súboru s takto vytvorenými odtlačkami. Navyše prenikol aj na iný server, ktorý používa rovnakú metódu ako my a tiež sa dostal k súboru s odtlačkami. Ak v tomto súbore nájde rovnaký odtlačok ako má uložený na našom serveri, pozná meno a heslo používateľa na inom serveri. Riešením je preto vložiť do reťazca s menom a heslom reťazec, ktorý je jedinečný pre náš server. Napr.:

Uvedený príklad je v skutočnosti málo pravdepodobný. Ukazuje nám však, že zrejme žiadna ochrana nie je 100 % a mali by sme byť preto stále v strehu a nenechať sa upokojiť falošným pocitom istoty.

```php
$hash_hesla = sha1($meno . $heslo . "d?_f54PJ\kLk54RE");
```

Potom sa nemôže (i keď teoreticky tá možnosť tu stále je) stať, aby dvaja rôzni používatelia s rovnakým prihlasovacím menom a heslom, na dvoch rôznych serveroch, ktoré používajú rovnakú metódu pri vytváraní odtlačkov, mali navzájom rovnaké odtlačky hesiel.

Problematická metóda GET

V súvislosti s formulármi sme spomínali spôsob odosielania dát z formulárových polí. Použiť môžeme dve metódy, metódu GET a metódu POST. Z hľadiska spracovania dát v skriptoch je to takmer jedno. Ak sa však pozrieme na bezpečnosť našich aplikácií, nájdeme podstatný rozdiel.

Ak posielame dáta metódou GET, tieto sa pripoja na koniec URI príslušného dokumentu, ktorý ich má spracovať. URI sa však môže uložiť (a zväčša aj ukladá) do histórie prejdených stránok alebo do pamäte typu cache na proxy serveroch a pod. Ktokoľvek, kto príde po nás k počítaču, môže stránku z histórie opäť použiť. Predstavme si, čo by sa stalo, keby sme sa takto prihlasovali do nejakého systému. Čo viac treba útočníkovi ak pozná URI v tvare:

Čiastočným riešením je použitie protokolu HTTPS namiesto protokolu HTTP. V prípade protokolu HTTPS nebude premenná `$_SERVER['HTTP_REFERER']` inicializovaná. Neplatí to však v prípade, ak útočníkova stránka je umiestnená na rovnakom serveri ako stránka, z ktorej sme prišli. Inicializácia premennej `$_SERVER['HTTP_REFERER']` je v kompetencii prehliadača, ktorý jej hodnotu odošle cieľovému serveru. To je ďalší dôvod, prečo nie je vhodné sa na toto spoliehať.

http://domena.sk/login.php?meno=janog&heslo=TatrY

V našich úvahách môžeme zájsť ďalej. Predpokladajme, že návštevník si chce pozrieť obsah diskusie na nejakom diskusnom serveri. Diskusia však nie je verejná a preto ešte pred samotným zobrazením príspevkov je návštevník presmerovaný na prihlasovací formulár. Po odoslaní dát z prihlasovacieho formulára sa vygeneruje napr. takéto URI:

http://domena.sk/login.php?meno=janog&heslo=TatrY&akcia=prihlas&id=2154

Obslužný skript vie, že má overiť totožnosť používateľa (akcia=prihlas) a v prípade úspechu zobraziť príslušný dokument (id=2154). Všetko prebehne v poriadku, používateľ sa prihlásil a zobrazila sa mu príslušná diskusia. V tejto diskusii ho však

zaujal odkaz www.akoneplatitdane.sk a preto naň klikol. Stránka nemusí byť ničím zaujímavá, stačí, aby si príslušný skript ukladal URI, odkiaľ prišli jeho návštevníci (napr. z premennej `$_SERVER['HTTP_REFERER']`). Autor stránky o daniach má teda všetky podstatné informácie. Ďalej snáď ani netreba pokračovať.

Záver teda je, nepoužívajme metódu GET v súvislosti s tajnými informáciami alebo naprogramujme skript tak, aby sa tajné informácie v URI skriptu nevygenerovali.

Problematické názvy súborov

Pri tvorbe aplikácií kvôli prehľadnosti rozdelíme väčšie skripty na niekoľko menších častí a každú z nich uložíme do samostatného súboru. Keďže tieto súbory sa nikdy nebudú volať priamo (cez ich URI), mnoho programátorov má zvyk týmto súborom dať inú príponu, napr. inc. V takomto súbore môžu byť uložené prístupové mená a heslá k databáze a pod. Keby však niekto zistil meno tohto súboru, vie si ho priamo zavolať cez jeho URI. Keďže prípona inc je pre server neznáma, ponúkne nám server obsah súboru, ktorý sa následne zobrazí v prehľadávači. Citlivé dáta alebo časť skriptu tak môže byť cennou trofejou pre prípadného útočníka.

Pri ladení skriptov programátori často pred väčšou úpravou zálohujú rôzne verzie svojich skriptov. Takže po serveri sa môžu „ponevierať“ súbory ako index.old, index.tmp, index.php~ a pod. Ak útočník uhádne názov tohto súboru, vie si ho priamo zavolať cez jeho URI. Dostávajú sa tak k nemu obsahy zdrojových súborov našich skriptov, z ktorých sa dá vyčítať až prekvapujúco veľa informácií o našej aplikácii.

Záver: Nikdy nepoužívajme neregistrované prípony na uloženie citlivých dát alebo skriptov a zálohu skriptov si robme radšej niekde inde ako na verejnom mieste servera.

Defence in Depth alebo čo ak niečo zlyhá

Nikoho z nás asi neprekvapí antivírusové riešenie v nejakej firme, ktoré vyzerá nasledovne. Firemný server, cez ktorý prebieha akákoľvek komunikácia medzi internetom a intranetom, obsahuje antivírusový skener, ktorý kontroluje všetky prechádzajúce dáta. Každý zo zamestnancov má na svojom stroji inštalovaný lokálny antivírusový systém. No a na pravidelných bezpečnostných školeniach zamestnancov im bezpečnostný technik neustále opakuje, ako sa správať k podozrivým e-mailom. Načo takáto trojnásobná kontrola? Z jednoduchého dôvodu. Ak jeden mechanizmus zlyhá (napr. antivírusový program na serveri nezachytí vírus) je tu šanca, že si s ním poradí antivírusový program na lokálnej stanici. Ak vírus unikne aj pozornosti lokálneho antivírusový program stále je šanca, že ho používateľ odhalí.

Podobný mechanizmus by sme mali aplikovať aj v našich aplikáciách. Ak jeden kontrolný mechanizmus zlyhá, ďalší v poradí by ho mal vedieť nejakým spôsobom nahradiť a problém eliminovať.

Na prvý pohľad sa môže zdať, že je to málo pravdepodobné. Opak je však pravdou. Pri vytvorení aplikácie si môžeme byť istí, že subsystém A neprepustí isté typy dát (spoľahneme sa napr. na konkrétne nastavenie servera alebo sme to jednoducho tak navrhli). Takže subsystém B, ktorý spracováva dáta z A, sa o túto kontrolu už nemusí starať. Neskôr sa vyskytne problém s tým, že isté typy dát by mali byť pre subsystém A povolené. Tak sa filtrovanie na úrovni subsystému A zvoľní. Lenže to, čo je teraz pre jeden systém akceptovateľné, pre druhý už byť nemusí. Keby si subsystém B prevádzal svoju vlastnú kontrolu, nič by sa nemuselo stať.

Takže ak je to možné, prevádzajme kontrolu na úrovni každého subsystému a snažme sa zabezpečiť kontrolu prostredníctvom niekoľkých nezávislých mechanizmov.

Biela, čierna, dobrá, zlá

Pri filtrovaní dát môžeme postupovať v zásade dvoma spôsobmi:

- identifikujeme neakceptovateľné (zlé) dáta a zahodíme ich, zvyšok prepustíme, zoznam „zlých“ dát sa nazýva čierna listina (blacklisting),
- identifikujeme akceptovateľné (dobré) dáta a pustíme ich do systému, zvyšok zahodíme, zoznam „dobrých“ dát sa nazýva biela listina (whitelisting).

Prvý z prístupov sa zdá byť celkom prirodzený a logický. Má však vážnu trhlinu. Ak by sme vedeli všetky dáta rozdeliť do dvoch skupín, dobré a zlé, bolo by v poriadku. Problém je ale v tom, že u niektorých dát nevieme povedať, či sú zlé alebo dobré. V podstate o nich nevieme nič.

Metóda bielej listiny ich zahodí (nie sú na bielej listine). Metóda čiernej listiny ich prepustí (lebo nie sú na čiernej listine). Teraz už vidno, prečo metóda čiernej listiny nie je správna. Dáta, o ktorých nič nevieme, a ktoré môžu byť nebezpečné, jednoducho pustíme ďalej. Pri kontrole vstupov identifikujme preto dobré dáta. Ak ich kontrola akceptuje, môžu vojsť do systému. V opačnom prípade nie.

Záznamy (logovanie)

Zrejme ste už vytvorili nejakú aplikáciu a dovolili ste, aby ju používali aj ostatní. A zrejme nie raz vás prekvapilo, ako zvláštne sa aplikácia správa v cudzích rukách. Nič nové pod slnkom. Používatelia sú naozaj vynaliezaví, takže náš „dokonalý“ program dokážu hravo dostať do neštandardnej situácie. Keď sa však neskôr snažíme zistiť, ako k tomu došlo, používatelia nie sú schopní nám presne popísať postupnosť krokov, ktorú vykonali.

Podobný problém nastane vtedy, ak niekto prenikne do našej aplikácie. Sme si istí, že všetko máme 100 % zabezpečené, ale aj tak sa to stalo. Teraz je to však horšie, útočník nám nepopíše postup, ako to spravil.

Práve v týchto situáciách by bolo dobré vedieť, akú postupnosť krokov používateľ vykonal. Jednou z možností je zaznamenávať (logovať) si činnosť používateľov. V podstate ide o to, že náš skript si bude zaznamenávať do nejakého neverejného priestoru všetko (alebo aspoň to podozrivé), čo sa vyskytlo počas jeho chodu, príp. aj s podmienkami, za ktorých sa to stalo. Zaznamenávať môžeme takmer čokoľvek - čas a dátum, IP adresu používateľa (resp. ďalšie identifikačné údaje), jeho login, z akej stránky prišiel, ktorú časť webu chcel zobraziť, hodnoty kľúčových premenných (ale nielen ich) a podobne. Okrem zaznamenávania činnosti používateľa je výhodné zaznamenávať aj chybové stavy skriptu (chyba pri include súborov, zápise do súborov a pod.) Z týchto záznamov môžeme neskôr vyčítať veľa zaujímavých informácií.

A ešte jedno upozornenie na záver. Záznamy vytvárajme pravidelne, ukladajme ich na bezpečné (neverejné) miesto, dlhodobo ich zálohujme, samozrejme, na inom mieste. Nie je nič horšie, ako keď sa nám niekto „nabúra“ do systému a ešte nám zmení aj záznamy.

Väčšina serverových aplikácií podporuje zaznamenávanie chýb počas svojej činnosti. Apache a PHP nie sú výnimkou (v PHP je táto možnosť v prednastavenej konfigurácii vypnutá). Direktívou log_errors (v konfiguračnom súbore PHP) nastavíme, či sa majú zaznamenávať chyby behu skriptov. Direktíva error_log určuje meno súboru, kam sa záznamy ukladajú. Záznam chýb do vlastného súboru nastavíme príkazmi:

```
ini_set("log_errors",
        "On");
ini_set("error_log",
        "meno_suboru");
```

Systémové záznamy chýb sú síce užitočné, pre podrobnú kontrolu činnosti skriptu sú však málo podrobné. Nič nám však nebráni v tom, aby sme si vytvárali vlastné záznamy činnosti našich skriptov.

Záznam chýb generovaných PHP interpreterom si môžeme pozrieť v časti „Error Log Viewer“ v administratívnej časti Uniform Servera na adrese http://localhost/apanel/.

„Security through obscurity“, alebo „... o tomto nikto nevie“

Jedna z častých chýb programátorov spočíva v tom, že očividný bezpečnostný problém ignorujú len preto, že si myslia, že nikto nemá šancu ho odhaliť. Spoliehať sa napr. na to, že heslá uložené v súbore nie je potrebné šifrovať len preto, že názov súboru „alseh.enjat“ nikto neuhádne, je naivné. Bezpečnosť systému založená na tom, že nikto o ňom nič nevie, nemá dlhú životnosť.

Slovo na záver kapitoly o bezpečnosti

Možno sa táto kapitola zdá zbytočne nafúknutá a rozťahaná. Možno naše skripty nebude nikto napádať. Možno si nikto nevšimne chybu, ktorú sme spravili. Možno sa

naozaj nič nestane. Spoliehať sa na to je však chybou. Je to podobné, ako keby sme riadenie auta zredukovali na ovládanie volantu a troch pedálov. Je to síce nutná podmienka, ale na bezpečné jazdenie v premávke absolútne nepostačujúca. Rovnako to platí aj pre naše PHP aplikácie. Buďme opatrní, predvídajme, učme sa z chýb. Byť trochu paranoickí nie je v tomto prípade vôbec na škodu.

Zadanie 9.1	Táto kapitola nemá žiadne zadanie. Dúfame ale, že sa k nej budete vracať vždy, keď budete programovať svoje vlastné skripty. Bezpečnosť sa neoplatí podceňovať.

Čo sme sa naučili

V tejto kapitole sme sa zamerali na bezpečnosť PHP aplikácií. Ukázali sme si najčastejšie bezpečnostné riziká hroziace PHP aplikáciám. Vieme, že je dôležité kontrolovať vstupy z iných systémov, najmä od používateľov našich stránok. Ukázali sme si ako správne používať príkaz pre vkladanie iných skriptov (častí našej stránky). Vieme ako správne využívať chyby generované PHP interpreterom. Dozvedeli sme sa ako bezpečne uchovávať heslá našich používateľov. Vieme aké skryté riziká prináša použitie metódy GET v našich aplikáciách. Ukázali sme si prečo nevytvárať súbory s neregistrovanou PHP príponou. Dozvedeli sme sa, že viacnásobná ochrana na viacerých úrovniach nie je chybou, ale práve naopak. Poznáme spôsoby ako posudzovať, či vstupné dáta sú dobré alebo zlé. Ukázali sme si, že zaznamenávanie činnosti našich skriptov môže byť užitočné nie len pri ich ladení, ale aj pri odhaľovaní nami nepredvídanej činnosti.

Čo sme sa naučili v tomto module

Zhrnutie

- získali sme prehľad o možnostiach využitia skriptovacieho jazyka PHP,
- rozumieme princípom technológií na strane servera a na strane klienta,
- poznáme výhody a nevýhody týchto technológií,
- vieme inštalovať a čiastočne konfigurovať HTTP server s podporou PHP,
- vieme vytvárať jednoduché webové dynamické aplikácie v jazyku PHP,
- poznáme a vieme používať formulárové elementy jazyka (X)HTML,
- vieme uložiť dáta do súboru na serveri a prečítať dáta zo súboru na serveri prostredníctvom PHP skriptu,
- rozumieme významu vlastných funkcií a vieme definovať vlastné funkcie,
- poznáme a vieme eliminovať základné bezpečnostné riziká dynamických webových aplikácií založených na PHP.

Preverenie výstupných vedomostí

Záverečné zadanie	Naprogramujte jednoduchú, zmysluplnú a bezpečnú webovú aplikáciu: 1. Podrobne špecifikujte problém, ktorý bude vaša aplikácia riešiť. 2. Urobte analýzu problému, snažte sa odhaliť všetky

situácie, do ktorých sa môžete pri riešení problému dostať. Skúste si uvedomiť všetko, čo vám môže skomplikovať vašu prácu a na čo budete musieť brať ohľad.

3. Navrhnite riešenie problému.
4. Naprogramujte toto riešenie v jazyku PHP.
5. Overte a otestujte funkčnosť a bezpečnosť vášho riešenia. V prípade potreby svoje skripty upravte.

Snažte sa, aby vaše zadanie a jeho riešenie neboli triviálne. Vo svojom zadaní by ste mali využiť formuláre a prácu so súbormi. Použitie štandardných konštrukcií jazyka PHP (funkcie, cykly, práca s premennou, podmienené vetvenie skriptu atď.) je samozrejmé. Vaše skripty by mali byť nielen správne a bezpečné, ale aj prehľadne napísané. Ich výsledkom by mali byť validované, prístupné a použiteľné webové stránky.

Vaše zadanie môže vychádzať z riešenia praktického problému, s ktorým sa možno denne stretávate. Môže to byť skript, ktorý vám alebo vašim kolegom uľahčí či zjednoduší prácu.

Literatúra a použité zdroje

[1] PHP v príkladoch, http://di.ics.upjs.sk/informatika_na_zs_ss/studijny_material/programovanie_internet/kui_php/index.htm
[2] Úvod do PHP, http://di.ics.upjs.sk/informatika_na_zs_ss/studijny_material/programovanie_internet/kui_php/php_kui.pps
[3] Formuláre: JavaScript a PHP, http://di.ics.upjs.sk/vyucba/pomocne_materialy/formulare/formulare.html
[4] Živě.cz – o počítačích a internetu, séria článkov o PHP, http://zive.cz/h/Programovani/default.asp?CAI=2038
[5] o webu, e zine pro webmastery, séria článkov o PHP, http://www.owebu.cz/php/
[6] PHP: Hypertext Preprocessor, http://www.php.net
[7] PHP: Download documentation, http://www.php.net/download-docs.php
[8] PHP Designer 2007 - Personal 5.0.2, http://www.mpsoftware.dk/phpdesigner_personal.php
[9] Welcome! - The Apache HTTP Server Project, http://httpd.apache.org
[10] The Uniform Server, http://www.uniformserver.com
[11] WampServer Apache, PHP, MySQL on Windows, http://www.wampserver.com/en/
[12] HUSEBY, Sverre. Zraniteľný kód. Computer Press, 2006. ISBN 80-251-1180-6
[13] KOSEK, Jiří. PHP -- tvorba interaktivních internetových aplikací, http://www.kosek.cz/php/php-tvorba-interaktivnich-internetovych-aplikaci.pdf
[14] The Open Web Application Security Project, http://www.owasp.org/