Дрогобицький державний педагогічний університет імені Івана Франка

Олег Наум, Андрій Пелещишин, Дмитро Карпин

ВЕБ-ТЕХНОЛОГІЇ (ЧАСТИНА 1)

Лабораторний практикум

з дисципліни “Веб-технології”

для студентів напряму підготовки 6.050101 “Комп’ютерні науки”

Дрогобич

2014

УДК 004.4

ББК

Р

Рекомендовано до друку вченою радою Дрогобицького державного педагогічного університету імені Івана Франка

(протокол № ___ від _____ 20___ р.)

Рецензенти:

Дорошенко М.В., кандидат фізико-математичних наук, доцент кафедри інформатики та обчислювальної математики Дрогобицького державного педагогічного університету імені Івана Франка.

Литвин В.В., доктор технічних наук, професор кафедри інформаційних систем та мереж Національного університету “Львівська політехніка”.

Відповідальний за випуск:

Лучкевич М.М., кандидат фіз.-мат. наук, т.в.о. завідувача кафедри інформаційних систем і технологій Дрогобицького державного педагогічного університету імені Івана Франка.

Р

Наум О.М., Пелещишин А.М., Карпин Д.С. Веб-технології (Частина 1): лабораторний практикум: навчальний посібник [для студентів напрямку підготовки 6.050101 “Комп’ютерні науки”] / Наум Олег Миколайович, Пелещишин Андрій Миколайович, Карпин Дмитро Степанович. – Дрогобич : Редакційно-видавничий відділ Дрогобицького державного педагогічного університету імені Івана Франка, 2014. – 164 с.

Лабораторний практикум розроблено відповідно до програми навчальної дисципліни “Веб-технології ” для підготовки фахівців освітньо-кваліфікаційного рівня “Бакалавр” напряму підготовки 6.050101 “Комп’ютерні науки” галузі знань 0501 “Інформатика та обчислювальна техніка”, затвердженої вченою радою Дрогобицького державного педагогічного університету імені Івана Франка (протокол № ___ від _____ 2014 р).

Посібник містить покрокові інструкції до виконання поставлених завдань та завдання для самостійного виконання 11 лабораторних робіт засобами інтерпретатору PHP.

УДК 004.4

ББК

Зміст

Вступ

В умовах інтенсивного розвитку мережі Інтернету з'являються нові технології, витісняючи ті, що здавалось б, нещодавно ввійшли до повсякденної практики. Сьогодні веб-технології стали невід'ємною частиною сучасного світу, вони значною мірою впливають на подальший суспільний розвиток людства. Веб-технології, проникають практично у всі сфери людської діяльності. Інтернет став неодмінним атрибутом робочого місця працівників багатьох професій.

У наші дні роль розробки додатків для WWW в структурі глобальної мережі зростає, відповідно збільшується і середня оцінка складності сценаріїв. Багато систем (наприклад, пошукові) за обсягом коду наближаються до розміру вихідних кодів серйозних прикладних програм. Частка ж статичних сторінок у Веб постійно падає; на зміну їм приходять динамічні сторінки, згенеровані автоматично тим чи іншим сценарієм.

Мета цього практикуму – ознайомити студентів з мовою програмування PHP, інтерфейсом CGI, методами передачі даних у PHP-сценарії, технологіями сесій та cookie, використанням шаблонів для створення веб-сайтів.

У поданих лабораторних роботах в якості інструментів для створення скриптів використовується Notepad++, Dreamweaver, Netbean, для обробки http-запитів – веб-сервер Apache.

Матеріал лабораторного практикуму розподілений на 11 робіт, форма викладання яких методично продумана й забезпечує поступове послідовне засвоєння основних конструкцій мови PHP та технологій створення веб-сайту. Кожна лабораторна робота містить теоретичні відомості, у яких досить ґрунтовно описані основні аспекти з певної теми. Теми розташовані у такому порядку, щоб кожна наступна не лише надавала нові знання, але й закріплювала та доповнювала попередні. У кожній темі, крім теоретичного матеріалу, міститься коди розв'язку прикладів завдань, що допомагають зрозуміти спосіб виконання та правильно виконати індивідуальне завдання. Контрольні запитання подані в кінці кожної теми дають змогу перевірити, як засвоєні знання з певної тематики.

Простота й послідовність викладеного матеріалу, наявність прикладів із повним описом застосування роблять цей навчальний посібник загальнодоступним. Він успішно може бути використаний для самостійного вивчення основ мови PHP, взаємодії клієнта та веб-сервера, використання шаблонізатора для створення сайту. Особи, які у повному обсязі засвоять запропонований матеріал, будуть достатньо підготовлені для самостійного розв’язання завдань, що виникають у практичній діяльності.

Зміст практикуму відповідає типовим освітнім програмам і розрахований на студентів ВНЗ, слухачів курсів, а також самоосвіти.

Тема 1: Лінійні програми та розгалуження

Мета роботи: навчитись створювати php-сценарій з лінійними виразами та розгалуженнями.

Теоретичні відомості

Код сценарію на мові PHP починається після відкриваючого тегу <? і закінчується ?>. Між цими двома тегами текст інтерпретується як програма, і в HTML документ не потрапляє. Якщо ж програмі потрібно щось вивести, вона повинна скористатися оператором echo (це не функція, а конструкція мови). PHP влаштований так, що будь-який текст, який розташований поза програмними блоками, обмеженими <? і ?>, виводиться в браузер безпосередньо, і сприймається, як виклик оператора echo.

Частина рядка після // або # є коментарем і на програму ніяк не впливає. Багаторядковий коментар розміщується між /* та */, приклад:

```
/*
це коментар
...і ще один рядок
*/
```

Розглянемо рядок коду:

```
$dat = date("d.m.y");
```

Цей рядок робить наступне: змінній з ім'ям $dat (назва змінної в PHP повинна починатися із знаку $) привласнюється значення, яке повернула функція date(). В PHP, по-перше, немає необхідності явно описувати змінні, а по-друге, ніде не вказується їх тип (рядок, число і т д). Інтерпретатор сам вирішує, що, де і якого типу. У цьому прикладі функції date() передається один параметр, який визначає формат результату, і поверне сьогоднішню дату у заданому форматі

В кінці кожного оператора повинна стояти крапка з комою (;). Зокрема, необхідно ставити крапку з комою перед else в конструкції if else, але не потрібно після опису функції.

Обмежувальні теги <?...?>, <?php…?> і навіть <%...%>, як правило, позначають одне і те ж. Проте для працездатності «скорочених» тегів <?...?> необхідне включення опції short_open_tag в конфігураційному файлі PHP php.ini, а для конструкції <%...%> настройки asp_tags.

Змінні

Як у більшості мов програмування, в PHP існує таке поняття, як змінна. При програмуванні на PHP прийнято не скупитися на оголошення нових змінних, навіть якщо можна обійтися і без них. У такому випадку сценарій буде значно читабельніший. Це також пов'язано з тим, що створення нового ідентифікатора інтерпретатору обходиться досить "дешево".

Імена змінних чутливі до регістру букв: наприклад $my_variable - не те ж саме, що $My_variable або $my_Variable. Крім того, імена всіх змінних повинні починатися із знаку $ - так інтерпретатору значно легше "зрозуміти" і відрізнити їх, наприклад, в рядках.

У офіційній документації сказано, що ім'я змінної може складатися не тільки з латинських букв і цифр, але також і з будь-яких символів, код яких старший 127, - зокрема, і з "кириличних" букв! Проте застосовування кирилиці в іменах змінних не рекомендується - хоч би через те, що в різних кодуваннях її букви мають різні коди.

Дії із змінними

Незалежно від типу змінної, над нею можна виконувати три основні дії.

Привласнення значення

Можна привласнити деякій змінній значення іншої змінної (або значення, повернуте функцією), посилання на іншу змінну, або ж константний вираз (за винятком об'єктів, для яких натомість використовується оператор new). За перетворення типів відповідає сам інтерпретатор. Крім того, при привласненні старий вміст і, що найважливіше, тип змінної втрачаються, і вона стає абсолютно

точною копією свого "батька". Тобто, якщо масиву привласнити число, це спрацює, і масив при цьому буде загублений.

Перевірка існування

Можна перевірити, чи існує (тобто чи ініціалізована) вказана змінна. Здійснюється це за допомогою вбудованого в PHP оператора isset(). Наприклад:

```
if (isset($myVar))
echo "Така змінна є. Її значення $myVar";
```

Якщо змінної в даний момент не існує (тобто ніде раніше їй не привласнювалося значення, або ж вона була вручну видалена за допомогою unset), то isset() повертає хибність.

Знищення

Знищення змінної реалізується оператором unset(). Після цієї дії змінна видаляється з внутрішніх таблиць інтерпретатора, тобто програма починає виконуватися так, як ніби змінна ще не ініціалізована. Наприклад:

```
// Змінної $a ще не існує
$a = "Hello there!";
// Тепер $a ініціалізована
// ... якісь команди, що використовують $a
echo $a;
// А зараз видалимо змінну $a
unset($a);
// Тепер змінної $a знову не існує
echo $a;  // Помилка: немає такої змінної $a
```

Оператор привласнення

Оператор привласнення в мові PHP це - знак рівності (=):

```
$ім'я_змінної = значення;
```

Розробники PHP пішли по лінії мови C в питанні операторів привласнення (і перевірки рівності, яка позначається ==), чим, ймовірно, внесли свій внесок у розмноження численних помилок. Наприклад, якщо в C написати:

```
if (a = b) { ... }
```
замість
```
if (a == b) { ... }
```

(пропускаючи ненароком один символ рівності), то компілятор видасть, принаймні, попередження. Інакше йде справа в PHP, наприклад:

```
$a = 0; $b = 1;
if ($a = $b) echo "a і b однакові";
else echo "a і b різні";
```

Інтерпретатор навіть не "пікне", а програма захоплено заявить, що "a і b однакові", хоча очевидно, що це зовсім не так (річ у тому, що $a = $b так само, як і $a + $b, є виразом, значення якого є права частина оператора привласнення, рівна в нашому прикладі 1).

Константи

Зустрічаються випадки, коли змінні досить незручно використовувати для постійного зберігання яких-небудь певних величин, які не міняються протягом роботи програми. Такими величинами можуть бути математичні константи, шляхи до файлів, різноманітні паролі і т.д. Якраз для цих цілей в PHP передбачена така конструкція, як константа.

Константа відрізняється від змінної тим, що, по-перше, їй ніде в програмі не можна привласнити значення більше одного разу, а по-друге, перед її ім'ям не потрібно ставити знак $, як це робиться для змінних. Наприклад:

```
// Припустимо, визначена константа MYCONST, рівна 10.
$a = MYCONST * 1.2 + 5; // використання константи
echo "Це число MYCONST";   // виведе "Це число MYCONST"
echo "Це число ".MYCONST;  // виведе "Це число 10"
```

Як можна бачити з прикладу, є і недолік: вже не можна використовувати ім'я константи безпосередньо в текстовому рядку.

Зарезервовані константи

Константи бувають двох типів: одні - зарезервовані (тобто ті, що встановлюються самим інтерпретатором) та інші, які визначаються програмістом. Існує декілька зарезервованих констант.

- __FILE__ – містить ім’я файлу, в якому розташований запущений зараз код.
- __LINE__ – містить поточний номер рядка, який обробляє у даний момент інтерпретатор. Ця своєрідна "константа" кожного разу міняється по ходу використанні програми. (Втім __FILE__ також міняється, якщо ми передаємо управління в інший файл.)
- PHP_VERSION – версія інтерпретатора PHP.
- PHP_OS – ім'я операційної системи, під управлінням якої працює PHP.
- TRUE або true – ця константа містить значення "істина".
- FALSE або false – містить значення "хибність".
- NULL або null – містить значення null.

Визначення констант

Можна визначати свої власні, нові константи. Робиться це за допомогою оператора define(), дуже схожого на функцію.

```
void define(string $name, string $value, bool $case_sen=true);
```

Визначає нову константу з ім'ям, переданим в $name, і значенням $value. Якщо необов'язковий параметр $case_sen рівний true, то надалі в програмі регістр букв константи враховується, інакше - не враховується (за умовчанням, регістр враховується). Створена константа не може бути знищена або перевизначена.

Наприклад:

```
define("pi", 3.14);
define("str", "Test string");
echo sin(pi/4);
echo str;
```

При визначенні константи, її ім’я повинне бути обрамлене лапками. Також не можна двічі визначати константу з одним і тим же ім'ям - це породить помилку під час виконання програми.

Перевірка існування константи

У PHP існує також функція, яка перевіряє, чи існує константа з вказаним ім'ям.

```
bool defined(string $name)
```

Повертає true, якщо константа з ім'ям $name була раніше визначена.

Вирази

Практично все, що пишеться в програмі, - це вирази. Найпростіший приклад виразу - змінна або константа, що стоїть, скажімо, в правій частині оператора привласнення. Наприклад, цифра 5 в операторі $a = 5;

5 є вираз, оскільки він має значення 5. Після такого привласнення ми маємо право чекати, що в $a опиниться 5.

Тепер, якщо написати $b = $a; то, очевидно, в $b опиниться також 5, адже вираз $a у правій частині оператора має значення 5.

Тоді $b = $a - теж вираз! Неважко здогадатися, яке воно має значення: 5. А це означає, що можна написати таке:

```
$a = ($b = 10); // або просто $a = $b = 10
```

При цьому змінним $a і $b привласнюється значення 10. А ось ще приклад, вже менш тривіальний:

```
$a = 3 * sin($b = $c + 10) + $d;
```

Що опиниться в змінних після виконання цих команд? Очевидно, те ж, що і в результаті роботи наступних операторів:

```
$b = $c + 10;
$a = 3 * sin($c + 10) + $d;
```

В PHP при обчисленні складного виразу можна задавати змінним значення частини виразу прямо усередині оператора привласнення. Такий прийом може

дійсно сильно спростити життя і скоротити код програми, "читабельність" якої збережеться на колишньому рівні.

Абсолютно точно можна сказати, що у будь-якого виразу є тип його значення. Наприклад:

```
$a = 10 * 20;
$a = "" . (10 * 20);
echo "$a:".gettype($a).", $b:".gettype($b);
// виведе "200:integer, 200:string"
```

Щоб перетворити одне значення в інше, використовуються оператори перетворення типів. Ці оператори доступні як у функціональній, так і в префіксній операторній формі. Наприклад, наступні дві інструкції еквівалентні:

```
$a = intval($b); $a = (int)$b;
```

Оператори перетворення типів:

```
$b = intval(вираз) або $b = (int) (вираз)
```

Переводить значення виразу в ціле число і привласнює його $b.

```
$b = doubleval(вираз) або $b = (double)(вираз)
```

Переводить значення в дійсне число і привласнює його $b.

```
$b = strval(вираз) або $b = (string) (вираз)
```

Переводить значення виразу в рядок.

```
$b = (bool) (вираз)
```

Перетворить значення виразу в логічний тип. Тобто, після виконання цього оператора в $b опиниться або true, або false.

Логічні вирази

Абсолютно будь-який вираз може розглядатися як логічний в "логічному" ж контексті (наприклад, як умова для конструкції if-else). Адже, як істина може виступати будь-яке ненульове число, непорожній рядок і т. д., а під хибністю мається на увазі все інше.

Для логічних виразів справедливі всі ті ж висновки, які ми зробили щодо логічних змінних. Ці вирази найчастіше виникають при застосуванні операторів

>, < і == (рівно), || (логічне АБО), && (логічне І), ! (логічне НЕ) та інших. Наприклад:

```
$less = 10 < 5;  //   $less - false
$equals = $b == 1;    //   $equals - true, якщо $b == 1
$between = $b>=1 && $b<=10  //  $between - true, якщо $b від 1 до 10
$x = !($b || $c) && $d;     //  true, якщо $b і $c хибні, а $d - істинно
```

Перевірка істинності тієї або іншої логічної змінної робиться точно так, як і будь-якого логічного виразу:

```
$between = $x>=1 && $x<=7; // значення виразу, яке привласнюється $between
if ($between) echo "x в потрібному діапазоні значень";
```

Операції

Арифметичні операції

До них відносяться:

- a + b - додавання;
- a - b - віднімання;
- a * b - множення;
- a / b - ділення;
- a % b - залишок від ділення a на b.

Операція ділення / повертає ціле число (тобто результат ділення без остачі), якщо обидва вирази a і b - цілого типу (або ж рядки, що виглядають як цілі числа), інакше результат буде дробом. Операція обчислення залишку від ділення % працює тільки з цілими числами, так що застосування її до дробів може привести, м'яко кажучи, до небажаного результату.

Рядкові операції

До них відносяться:

- a.b - злиття рядків a і b;
- a[n] - символ рядка у позиції n.

Власне, інших рядкових операцій і немає - все інше, що можна зробити з рядками в PHP, виконують стандартні функції (такі як: strlen(), substr() і т.д.).

Операції привласнення

Основним з групи операцій є оператор привласнення =. Він "говорить" інтерпретатору, що значення правого виразу повинне бути привласнене змінною зліва. Наприклад:

```
$a = ($b = 4) + 5;
```

Після цього $a рівне 9, а $b рівне 4.

Крім цього основного оператора, існує ще комбіновані – по одному на кожну арифметичну, рядкову і бітову операцію. Наприклад:

```
$n = 6;
$n += 1; // додати 1 до $n
$message = "Виконано";
$message .= " $n разів!"; // тепер в $message "Виконано 7 разів!"
```

Операції порівняння

Це у своєму роді унікальні операції, тому що незалежно від типів своїх аргументів вони завжди повертають одне з двох значень: false або true. Операції порівняння дозволяють порівнювати два значення між собою і, якщо умова виконана, повертають true, інакше - false.

Отже:

a == b - істина, якщо a рівне b;

a != b - істина, якщо a не рівне b;

a < b - істина, якщо a менше b;

a > b - істина, якщо a більше b;

a <= b - істина, якщо a менше або рівне b;

a >= b - істина, якщо a більше або рівне b.

Логічні операції

Ці операції призначені виключно для роботи з логічними виразами і також повертають false або true:

- !a - істина, якщо a хибне, і навпаки;
- a && b - істина, якщо істинні і a, і b;
- a || b - істина, якщо істинні або a, або b, або обидва операнди.

Слід відмітити, що обчислення логічних виразів, які містять такі операції, йде завжди зліва направо, при цьому, якщо результат вже очевидний (наприклад false && щось завжди дає false), то обчислення обриваються, навіть якщо у виразі присутні виклики функцій. Наприклад, в операторі

```
$logic = 0 && (time()>100);
```

стандартна функція time() ніколи не буде викликана.

Інструкції PHP

Інструкція if-else

Формат умовного оператора такий:

```
if (логічний_вираз)
інструкція_1; else
інструкція_ 2;
```

Дія інструкції наступна: якщо логічна умова істинна, то виконується інструкція_1, а інакше – инструкція_2. Як і в будь-якій іншій мові, конструкція else може опускатися, в цьому випадку при набутті належного значення просто нічого не робиться.

Приклад:

```
if ($salary>=100 && $salary<=5000)
 echo "Вам ще рости і рости";
else echo "Ну і правильно - не в грошах щастя.";
```

Якщо інструкція_1 або інструкція_2 повинні складатися з декількох команд, то вони, беруться у фігурні дужки. Наприклад:

```
if ($a > $b) { print "a більше b"; $c = $b; }
  elseif ($a == $b) { print "a рівне b"; $c = $a;}
    else { print "a менше b"; $c = $a; }
echo "<br>Мінімальне з чисел: $c";
```

Це не друкарська помилка: elseif пишеться разом, замість else if. Конструкція if-else має ще один альтернативний синтаксис:

```
if (логічна_умова) :
команди;
elseif (друга_ логічна_умова) :
інші_ команди;
else:
інакше_ команди;
endif;
```

Варто звернути увагу на розташування двокрапки (:)! Якщо його пропустити, згенерується повідомлення про помилку. І ще: блоки elseif і else можна опускати.

Використання альтернативного синтаксису

Часто буває потрібно робити не вставки HTML всередину програми, а вставки коду всередину HTML. Це набагато простіше для дизайнера, який, можливо, в майбутньому захоче переоформити сценарій, але не зможе розібратися, що йому міняти, а що не чіпати. Тому доцільно відокремлювати HTML-код від програми, наприклад, помістити його в окремий файл, який потім підключається до програми за допомогою інструкції include.

Приклад коду з відокремленням програми від HTML:

Лістинг 1.1.

```
<!-- Альтернативний синтаксис -->
<? if (isset($_REQUEST['go'])): ?>
  Привіт, <?= $_REQUEST['name']; ?>!
<? else: ?>
```

```
  <form action="<?= $_SERVER['REQUEST_URI'] ?>" method=post>
  Ваше ім’я: <input type=text name=name><br>
  <input type=submit name=go value="Відіслати!">
<? Endif; ?>
```

Конструкція switch-case

Часто замість декількох розташованих підряд інструкцій if-else доцільно скористатися спеціальною конструкцією switch-case:

```
switch (вираз) {
case  значення1:  команди1;  [break;]
case  значення2:  команди2;  [break;]
case  значенняN:  командиN;  [break;]
[default:  команди_за_замовчуванням; ] }
```

Робить вона наступне: обчислює значення виразу (нехай він рівний, наприклад, v), а потім намагається знайти рядок, що починається з case v:. Якщо такий рядок виявлений, виконуються команди, розташовані відразу після неї (причому на всі подальші оператори «case щось» уваги не звертається, неначе їх немає, а код після них залишається без змін). Якщо ж знайти такий рядок не вдалося, виконуються команди після default (якщо вони задані).

Варто звернути увагу на оператори break (які поміщені у квадратні дужки, щоб підкреслити їх необов'язковість), додані після кожного рядка команд, окрім останньої. Коли б не вони, то при рівності v=значення1 спрацювали б не тільки команди1, але і всі інші.

Ось альтернативний синтаксис для конструкції switch-case:

```
switch (вираз) :
case значення1:  команди1;  [break;]
case значенняN:  командиN,  [break;]
[default: команди_за_замовчуванням; [break]]
endswitch;
```

Інструкції require і include

Ці інструкції дозволяють розбити текст програми на декілька файлів. Формат require такий:

```
require ім'я_файла;
```

При запуску програми інтерпретатор просто замінить інструкцію на вміст файлу ім'я_файла (цей файл може також містити сценарій на PHP). Це буває досить зручно для включення у сценарій всяких "шапок" з HTML-кодом. Наприклад.

Лістинг 1.2. Файл require/head.html

```
<!-- "Шапка". -->
<html>
<head><title>Title!</title></head>
<body bgcolor=black text=#00FFOO>
<b><pre>
```

Лістинг 1.3. Файл require/script.php

```
<?php ## Тіло скрипта.
require "head.html";
print_r($GLOBALS);
require "foot.html"; ?>
```

Лістинг 1.4. Файл require/foot.html

```
<!-- "Підвал". -->
</pre></b>
&copy;ТзОВ “Рога і копита”, 2014.
</body></html>
```

Безумовно, це краще, ніж включати HTML-код в сам сценарій разом з інструкціями програми. Недоліком цього методу є розростання сценарію аж до трьох файлів.

Інструкція include практично ідентична require, за винятком того, що у разі неможливості включення файлу, робота сценарію не завершується негайно, а продовжується (з виведенням відповідного діагностичного повідомлення). В більшості випадків навряд чи її використання виявиться доцільним.

Інструкції одноразового включення

Великі і складні сценарії зазвичай складаються з не одного десятка файлів, що включають один одного. Тому в скриптах доводиться зустрічати неодноразове застосування інструкцій include і require. При цьому виникає проблема: стає досить складно контролювати, як би випадково не включити один і той же файл кілька разів (що найчастіше призводить до помилки).

Рішення: require_once

Інструкція require_once працює точно так, як і require, але за одним важливим виключенням. Якщо вона бачить, що потрібний файл, вже був раніше включений, то нічого не робить.

Інструкція include_once працює абсолютно аналогічно, але у разі неможливості знайти файл, що включається, робота скрипта продовжується, а не завершується негайно.

Завдання

Лінійні програми

0. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$z_1 = 2\sin^2(3\pi - 2\alpha) \times \cos^2(5\pi + 2\alpha);$

$z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right).$

1. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$z_1 = \cos\alpha + \sin\alpha + \cos 3\alpha + \sin 3\alpha;$

$z_2 = 2\sqrt{2}\cos\alpha \times \sin\left(\frac{\pi}{4} + 2\alpha\right).$

2. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$z_1 = \frac{\sin 2\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha + 1 - 2\sin^2 2\alpha};$

$z_2 = 2\sin\alpha.$

3. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = \frac{\sin\alpha + \sin 5\alpha - \sin 3\alpha}{\cos\alpha - \cos 3\alpha + \cos 5\alpha};$$

$$z_2 = tg\,3\alpha.$$

4. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = 1 - \frac{1}{4}\sin^2 2\alpha + \cos 2\alpha;$$

$$z_2 = \cos^2\alpha + \cos^4\alpha.$$

5. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = \cos\alpha + \cos 2\alpha + \cos 6\alpha + \cos 7\alpha;$$

$$z_2 = 4\cos\frac{\alpha}{2} \times \cos\frac{5}{2}\alpha \times \cos 4\alpha..$$

6. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = \cos^2\left(\frac{3}{8}\pi - \frac{\alpha}{4}\right) - \cos^2\left(\frac{11}{8}\pi + \frac{\alpha}{4}\right);$$

$$z_2 = \frac{\sqrt{2}}{2} \cdot \sin\frac{\alpha}{2}.$$

7. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = \cos^4 x + \sin^2 y + \frac{1}{4}\sin^2 2x - 1;$$

$$z_2 = \sin(y + x) \times \sin(y - x).$$

8. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = (\cos\alpha - \cos\beta)^2 - (\sin\alpha - \sin\beta)^2;$$

$$z_2 = -4\sin^2\frac{\alpha - \beta}{2} \times \cos(\alpha + \beta).$$

9. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = \frac{\sin\left(\frac{\pi}{2} + 3\alpha\right)}{1 - \sin(3\alpha - \pi)};$$

$$z_2 = ctg\left(\frac{5}{4}\pi + \frac{3}{2}\alpha\right).$$

10. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = \frac{1 - 2\sin^2\alpha}{1 + \sin 2\alpha};$$

$$z_2 = \frac{1 - tg\,\alpha}{1 + tg\,\alpha}.$$

11. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = \frac{\sin 4\alpha}{1 + \cos 4\alpha} \times \frac{\cos 2\alpha}{1 + \cos 2\alpha};$$

$$z_2 = ctg\left(\frac{3}{2}\pi - \alpha\right).$$

12. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = \frac{\sin\alpha + \cos(2\beta - \alpha)}{\cos\alpha - \sin(2\beta - \alpha)};$$

$$z_2 = \frac{1 + \sin 2\beta}{\cos 2\beta}.$$

13. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = \frac{\cos\alpha + \sin\alpha}{\cos\alpha - \sin\alpha};$$

$$z_2 = tg\,2\alpha + \sec 2\alpha.$$

14. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = \frac{\sqrt{2b + 2\sqrt{b^2 - 4}}}{\sqrt{b^2 - 4} + b + 2};$$

$$z_2 = \frac{1}{\sqrt{b + 2}}.$$

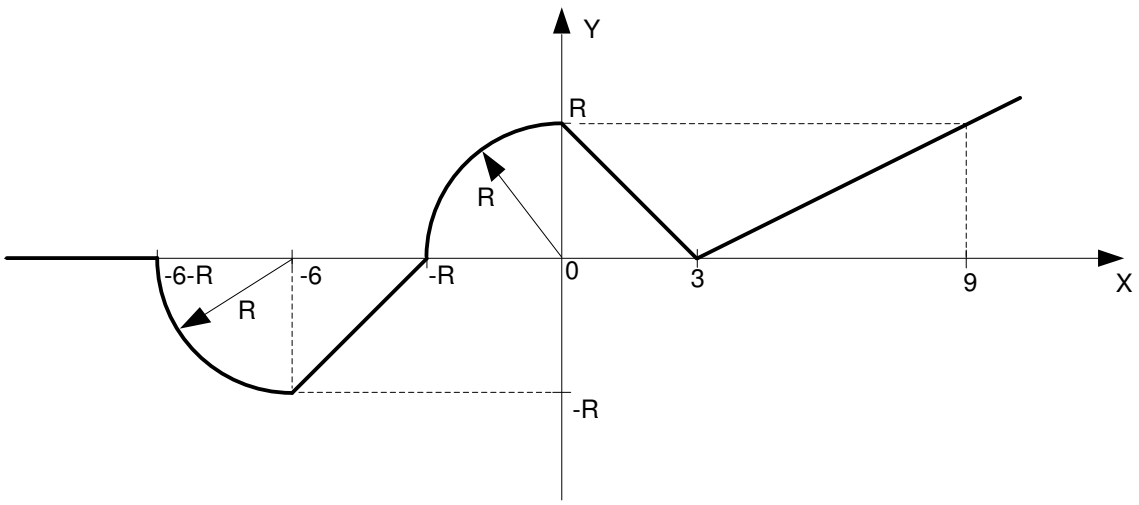
15. Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати).

$$z_1 = \frac{x^2 + 2x - 3 + (x+1)\sqrt{x^2 - 9}}{x^2 - 2x - 3 + (x-1)\sqrt{x^2 - 9}};$$

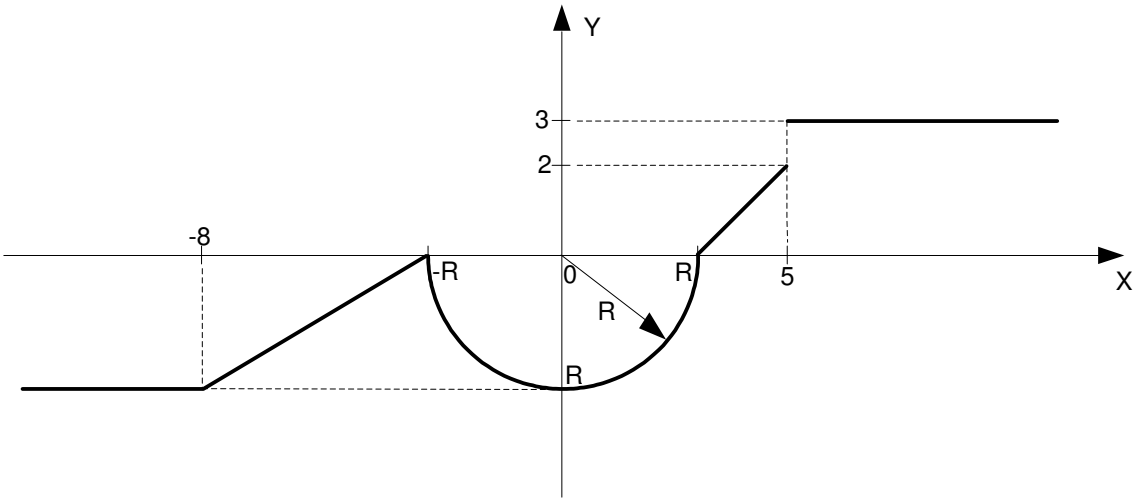
$$z_2 = \sqrt{\frac{x+3}{x-3}}.$$

Розгалужені програми

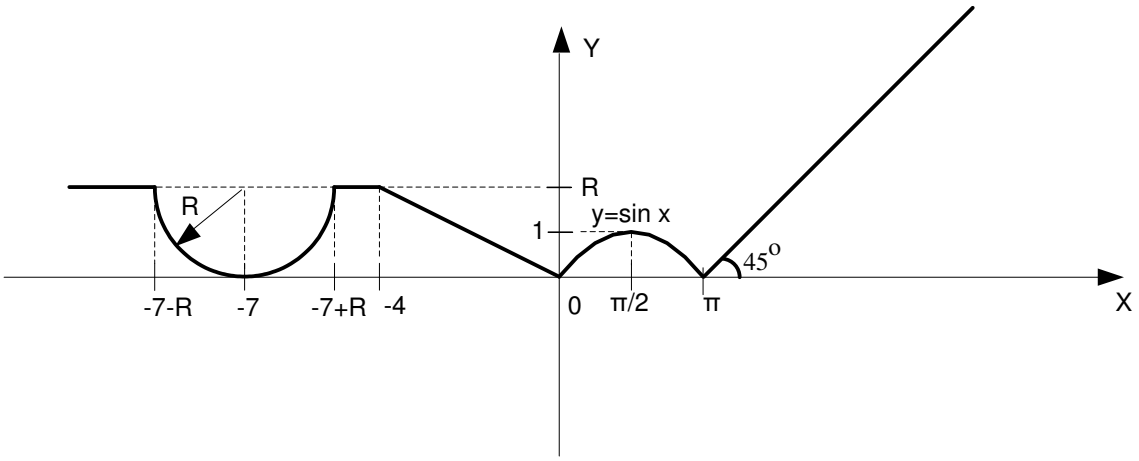
0. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



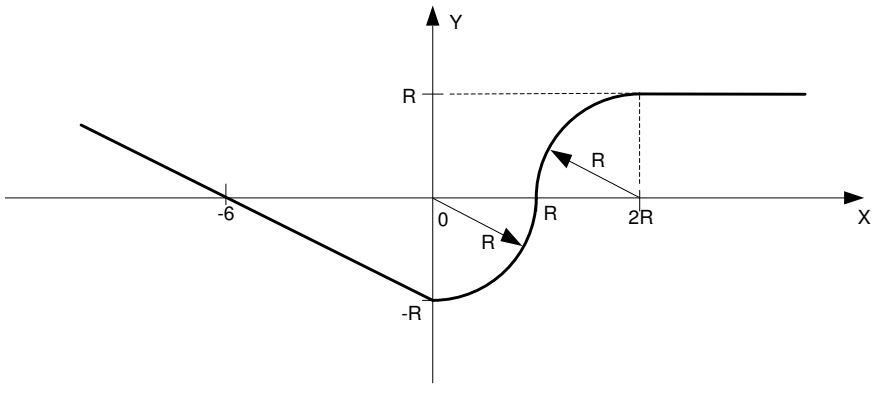
1. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



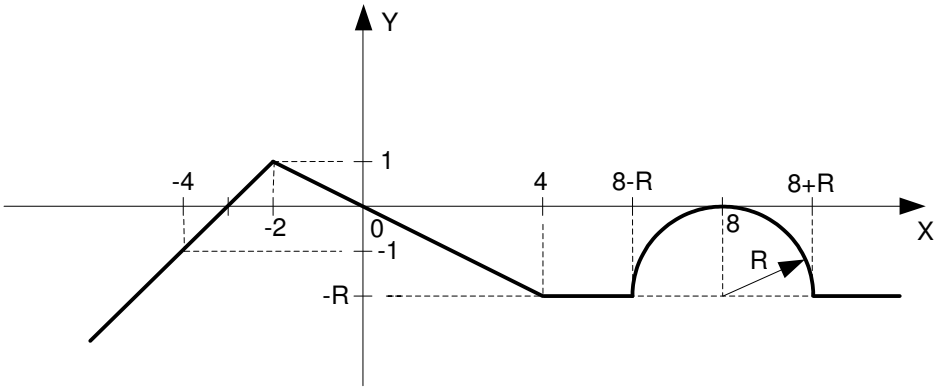
2. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



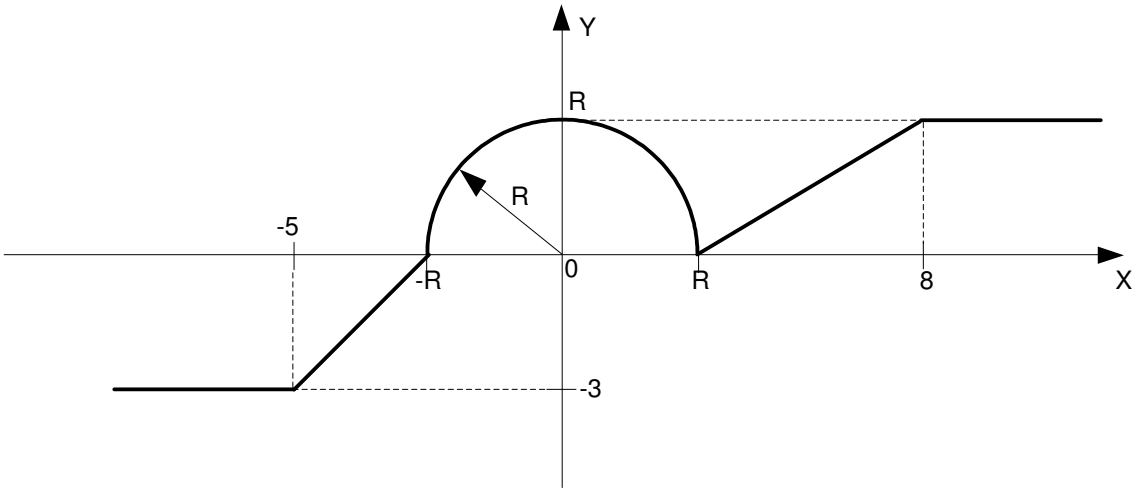
3. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



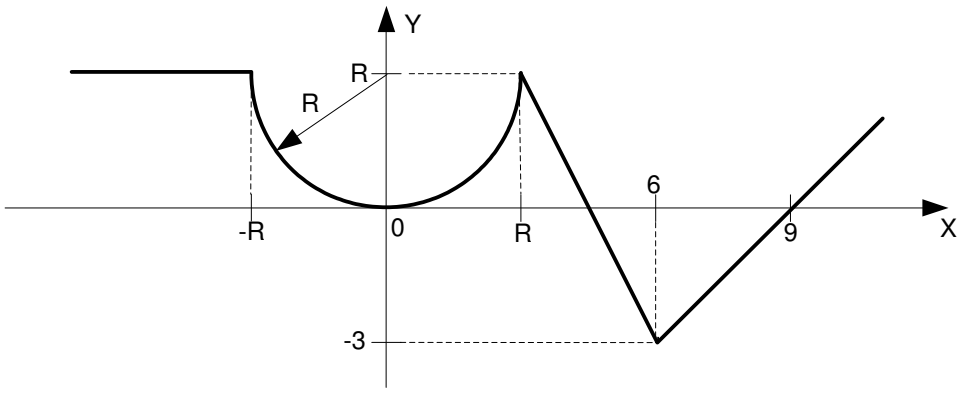
4. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



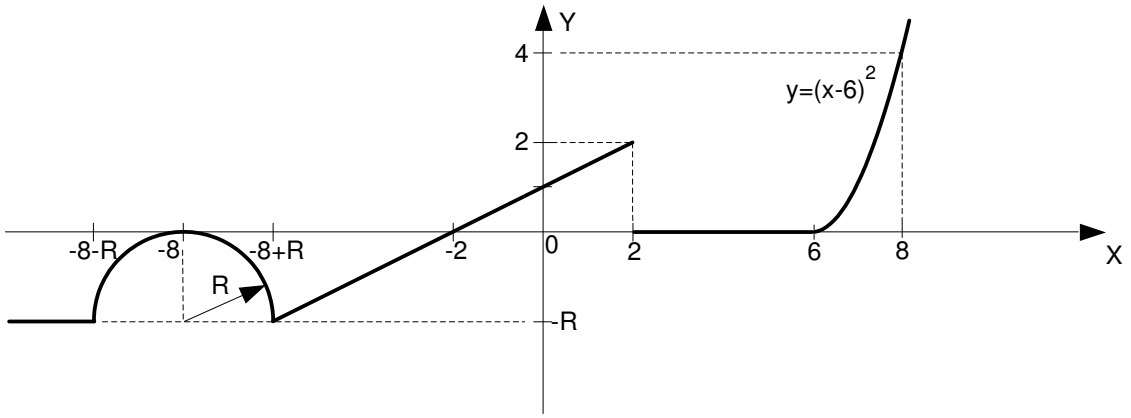
5. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



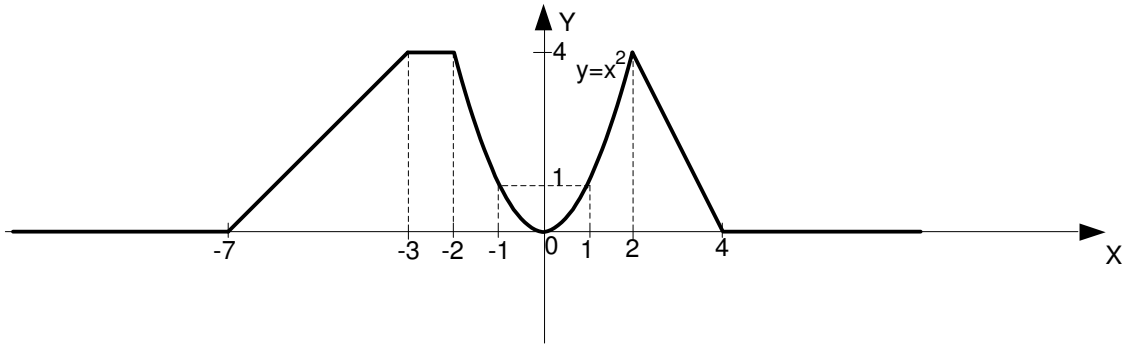
6. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



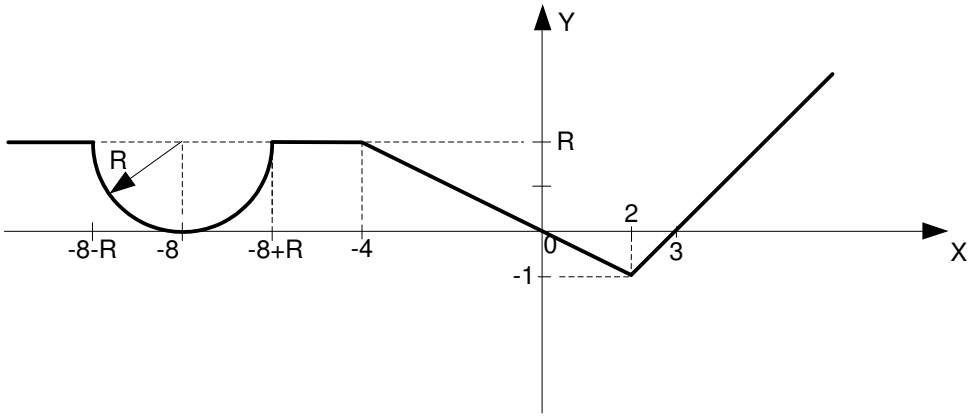
7. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



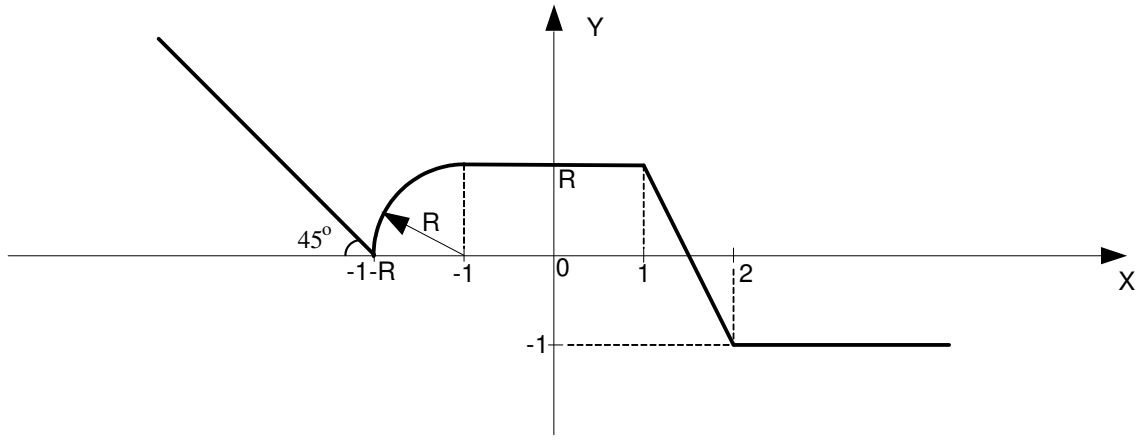
8. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



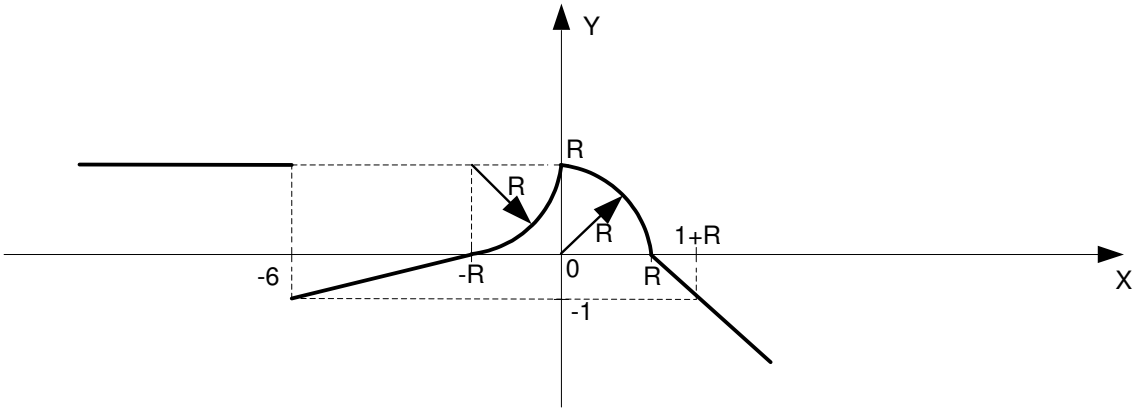
9. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



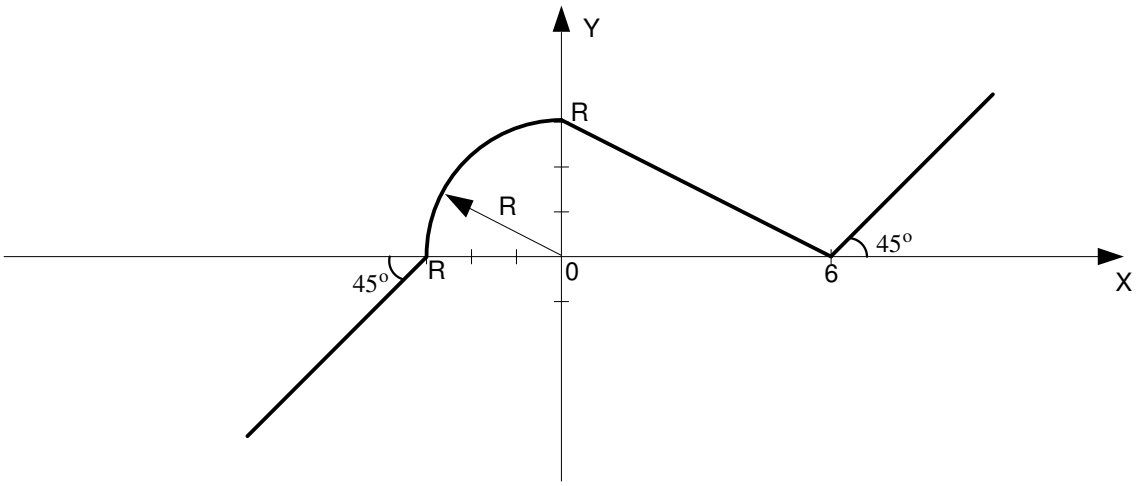
10. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



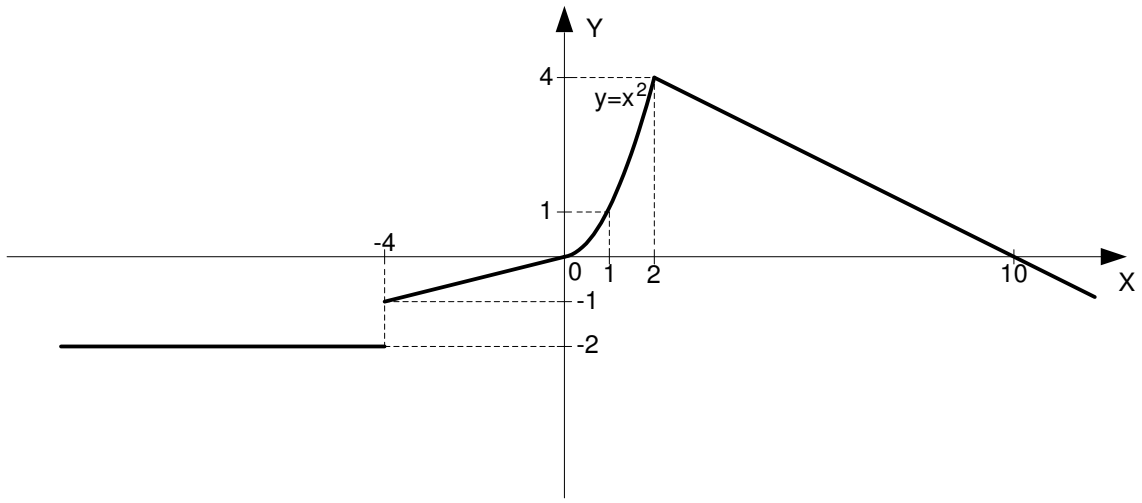
11. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



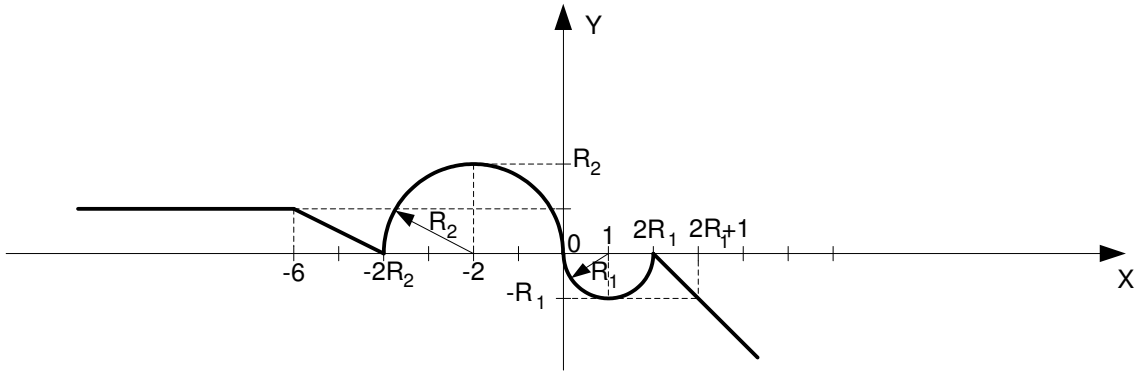
12. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



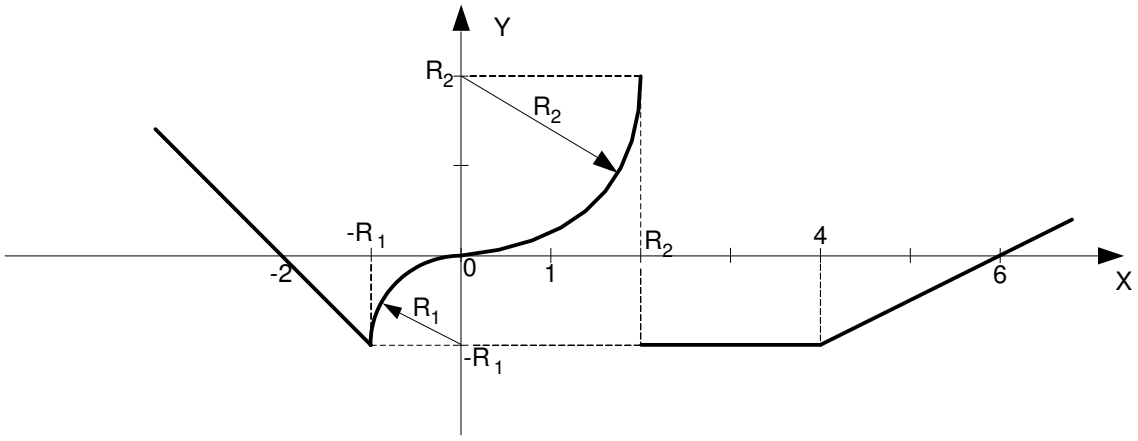
13. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



14. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



15. Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R задається користувачем.



Хід роботи

1. Створити новий файл у текстовому редакторі (Dreamweaver, Notepad або іншому).
2. Написати розв’язок завдань.
3. Зберегти програму на диск під новим ім`ям 1.php у папку <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB01
4. Перевірити виконання скрипта у браузері.
5. При виникненні помилки – відкоригувати програму та виконати її.
6. Оформити звіт по виконаній роботі.

Методичні рекомендації

Для введення даних використовуються параметри в URL, для виконання програми потрібно у браузері ввести <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB01\1.php?a=<зн1>&r=<зн2>&x=<зн3>.

Розв’язок завдання (текст програми):

```
<?php
/* Завдання 1 */
  $a = $_GET['a'];
  $z1 = 2*pow(sin(3*PI-2*$a),2)*pow(cos(5*PI+2*$a),2);
  $z2 = 1/4-1/4*sin(5/2*PI-8*$a);
    print("Завдання 1: a = $a, z1 = $z1, z2 = $z2<br/>");

/* Завдання 2 */
  $x = $_GET['x'];
  $R = $_GET['r'];
  if ($x < -6-$R) $y = 0;
    if ($x < -6) $y = -sqrt($R*$R-($x+6)*($x+6));
      elseif ($x < -$R) $y = ($x+6)*$R/(6-$R)-$R;
        elseif ($x < 0) $y = sqrt($R*$R-$x*$x);
          elseif ($x < 3) $y = ($x-$R)/3+$R;
            else $y = ($x-3)*$R/6;
    print("Завдання 2: R = $R, x = $x, y = $y");
?>
```

В цьому коді pow(x, n) – це функція піднесення x до степеня з показником n.

Контрольні запитання

1. Які теги позначають PHP-сценарій?
2. Як позначаються коментарі в PHP?
3. Які є форми умовного оператора?
4. Яка повна схема умовного оператора?
5. Що таке оператор switch?

Тема 2: Цикли

Мета роботи: навчитись використовувати різні форми операторів циклу на мові PHP.

Теоретичні відомості

Цикл з передумовою while

Ця конструкція успадкована безпосередньо від мови C. Її призначення - циклічне виконання команд в тілі циклу, що включає попередню перевірку, чи потрібно це робити (чи істинний логічний вираз в заголовку). Якщо не потрібно (вираз хибний), то конструкція закінчує свою роботу, інакше виконує чергову ітерацію і починає все спочатку. Виглядає цикл так:

```
while (логічний_вираз) інструкція;
```

де, логічна умова - логічний вираз, а інструкція - проста або складена інструкція тіла циклу. (Очевидно, що усередині інструкції повинні проводитися якісь дії, які іноді змінюватимуть значення логічного виразу, інакше оператор зациклиться. Це може бути, наприклад, просте збільшення деякого лічильника, що бере участь у виразі, на одиницю.) Якщо вираз з самого початку хибний, то цикл не виконається жодного разу. Приклад:

Лістинг 2.1.

```
<?php ## Виведення всіх ступенів двійки до 2^31 включно.
$i = 1; $p = 1;
while ($i < 32) {
  echo $p," ";
  $p = $p * 2; // можна було б написати $p *= 2
  $i = $i + 1; // можна було б написати $i += 1 або навіть $i++
}
?>
```

Аналогічно інструкції if, цикл while має альтернативний синтаксис, що спрощує його застосування разом з HTML-кодом:

```
while (логічна_умова):
команди;
```

```
endwhile;
```

Цикл з післяумовою do-while

На відміну від циклу while, цей цикл перевіряє значення виразу не до, а після кожного проходу. Таким чином, тіло циклу виконується хоч би один раз. Виглядає оператор так:

```
do {команди; } while (логічна_умова) ;
```

Після чергової ітерації перевіряється, чи істинна логічна_умова, і, якщо це так, управління передається знову на початок циклу, інакше цикл обривається.

Альтернативного синтаксису для do-while розробники PHP не передбачили.

Цикл for

```
for (ініціалізуючі_команди; умова_циклу; команди_після_проходу)
тіло_циклу ;
```

Працює він таким чином. Як тільки управління доходить до циклу, насамперед виконуються оператори, включені в команди, що ініціалізуються (зліва направо). Ці команди перераховуються через кому, наприклад:

```
for (i=0, j=10, $k="Test!; ...)
```

Потім починається ітерація. Спочатку перевіряється, чи виконується умова_циклу. Якщо так, то все гаразд, і цикл продовжується. Інакше здійснюється вихід з конструкції. Наприклад:

```
// додаємо по одній крапці
for ($i=0, $j=0, $k="Test"; $i<10; ...) $k .= ".";
```

Припустимо, що тіло циклу пропрацювало одну ітерацію. Після цього вступають в дію команди_після_проходу. Приклад:

Лістинг 2.2.

```
<?php ## Демонстрація циклу for
for ($i=0, $j=0, $k="Points"; $i<100; $j++, $i+=$j) $k = $k.".";
echo $k;
?>
```

Приведений приклад (та і взагалі будь-який цикл for) можна реалізувати і через while, тільки це виглядатиме не так лаконічно.

Наприклад:

```
$i = 0;  $j = 1;  $k = "";
while ($i<100) {
  $k .= ".";
  $i+ = $j;
}
echo $k;
```

Як завжди, є і альтернативний синтаксис конструкції:

```
for (ініціалізуючі_команди; умови_циклу; команди_після_проходу):
  оператори;
endfor;
```

Інструкції break і continue

Дуже часто, для того, щоб спростити логіку якого-небудь складного циклу, зручно мати можливість його перервати в ході чергової ітерації (наприклад, при виконанні якої-небудь особливої умови). Для цього і існує інструкція break, яка здійснює негайний вихід з циклу. Вона може задаватися з одним необов'язковим параметром - числом, яке вказує, з якого вкладеного циклу повинен бути проведений вихід. За умовчанням використовується 1, тобто вихід з поточного циклу, але іноді застосовуються і інші значення:

```
for ($i=0; $i<count($matrix); $i++) {
  for ($j=0; $j<count($matrix[$i]); $j++) {
    if ($matrix[$i][$j] == 0) break(2);
}}
if ($i < 10) echo 'Знайдений нульовий елемент в матриці!';
```

Інструкцію break зручно використовувати для циклів пошуків: як тільки чергова ітерація циклу задовольняє пошуковій умові, пошук обривається. У попередньому прикладі break використовувався для пошуку нульового елементу в деякому двовимірному масиві (прямокутній матриці). Стандартна функція count(), просто повертає кількість елементів в масиві $a.

Інструкція continue так само, як і break, працює тільки "в парі" з циклічними конструкціями. Вона негайно завершує поточну ітерацію циклу і переходить до

нової (звичайно, якщо виконується умова циклу для циклу з передумовою). Точно так, як і для break, для continue можна вказати рівень вкладеності циклу, який буде продовжений після повернення управління.

В основному continue дозволяє заощадити кількість фігурних дужок в коді і збільшити його легкість для читання. Це найчастіше буває потрібно в циклах-фільтрах, коли потрібно перебрати деяку кількість об'єктів і вибрати з них тільки ті, які задовольняють певним умовам. Наприклад, нижче представлений цикл, який друкує тільки ті елементи масиву $files (імена файлів і каталогів), які є файлами:

```
for ($i=0; count($files); $i++) {
  if ($files[$i] == ".") continue;
  if ($fiies [$i] == "..") continue;
  if (is_dir($fiies[$i] )) continue;
  echo "Знайдений файл: $files[$i] <br>";
}
```

Завдання

Напишіть програму для обчислення виразу. Невідомі змінні вводиться користувачем.

0. $\sum_{n=1}^{10} \frac{\sin \prod_{k=1}^{n} \frac{1}{k^2}}{n + \sqrt[n]{\prod_{k=1}^{n} \frac{1}{k^2}}}$

1. $\prod_{i=1}^{15} \frac{\sin^2 i + \cos^2 \sum_{k=1}^{i} \frac{1}{k}}{i^2}$

2. $\sum_{i=1}^{15} \frac{\sin 10i + \cos 10i}{\prod_{k=1}^{i} \sqrt{k}}$

3. $\prod_{j=1}^{15} \sqrt{\sum_{i=j}^{j^2} i^2}$

4. $\prod_{i=1}^{10} \frac{i + \sum_{k=1}^{i} \frac{1}{k}}{\sqrt{\sum_{k=1}^{i} \frac{1}{k}}}$

5. $\prod_{k=1}^{15}\left(1+\sum_{i=1}^{k}\cos^{i} k\right)^{k}$

6. $\prod_{k=1}^{20}\frac{\sum_{i=1}^{k} i^{2}}{1+\sum_{i=1}^{k} i^{2}}$

7. $\sum_{k=1}^{10}\frac{\sum_{n=1}^{k}\sin kn}{k}$

8. $\sum_{j=2}^{20}\frac{j}{j^{2}+\prod_{i=j^{2}}^{400} i}$

9. $\sum_{n=1}^{10}\sqrt[n]{1+\cos^{2} n+\prod_{k=1}^{n}\sin^{k} n}$

10. $\sum_{n=1}^{25}\left(\cos n+\prod_{k=1}^{n}\sqrt[k]{\cos^{2} n}\right)$

11. $\prod_{n=1}^{20}\frac{n^{2}+\left(\sum_{k=n}^{20} k\right)^{2}}{n+1}$

12. $\sum_{k=1}^{20}\frac{1+\sqrt{\sum_{i=k}^{40-k} i^{2}}}{k^{2}}$

13. $\sum_{i=1}^{10}\frac{1+\prod_{k=1}^{i}\frac{k}{i}}{i^{2}}$

14. $\sum_{n=1}^{18}\sum_{k=n}^{20}\frac{\sqrt{\left|1-\frac{k}{n}\right|}}{2n^{2}+k^{2}}$

15. $\prod_{i=1}^{20}\prod_{j=1}^{40-i}\frac{\sqrt[3]{|tg\, i|}}{i+j^{2}}$

Хід роботи

1. Створити новий файл у текстовому редакторі (Dreamweaver, Notepad або іншому).
2. Написати розв’язок завдання.
3. Зберегти програму на диск під новим ім`ям 1.php у папку <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB02
4. Перевірити виконання скрипта у браузері.

5. При виникненні помилки – відкоригувати програму та виконати її.
6. Оформити звіт по виконаній роботі.

Методичні рекомендації

Розв’язок завдання (текст програми):

```
<?php
  $s = 0;
  for ($n=1; $n<=10; $n++){
    $d = 1;
    $k = 1;
    while ($k <= $n) {
      $d *= 1/($k*$k);
      $k++;
    }
    $s += sin($d)/($n+pow($d,1/$n));
  }
    print("Результат обчислення виразу: = $s");
?>
```

Для виконання програми потрібно у браузері ввести <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB02\1.php.

Контрольні запитання

1. Яка загальна схема оператора while?
2. Яка загальна схема оператора do…while?
3. Яка загальна схема оператора for?
4. Яка дія оператора break?
5. Яка дія оператора continue?

Тема 3: Користувацькі функції

Мета роботи: навчитись створювати власні функції та використовувати їх.

Теоретичні відомості

Синтаксис опису функцій PHP простий і витончений:

- можна використовувати параметри за умовчанням (а значить, функції із змінним числом параметрів);
- кожна функція має свій власний контекст (або область видимості) змінних, яка знищується при виході з неї;
- існує зручна інструкція return;
- тип значення, що повертається, може бути будь-яким;
- при описі методів класів для параметрів функцій можливо вказати їх тип з примусовою перевіркою при виклику.

На жаль, розробники PHP не передбачили можливість створення локальних функцій (тобто одної усередині іншої). Проте деяка емуляція локальних функцій все ж таки є.

У системі визначення функцій в PHP є одна особливість, яка вельми неприємна тим, хто до цього програмував на інших мовах. Річ у тому, що всі змінні, які оголошуються і використовуються у функції, за умовчанням локальні для цієї функції. При цьому існує тільки один (і при тому досить непривабливий) спосіб оголошення глобальних змінних - інструкція global. З одного боку, це підвищує надійність функцій в сенсі їх незалежності від основної програми, а також гарантує, що вони випадково не змінять і не створять глобальних змінних. З іншого боку, розробники PHP цілком могли б передбачити необхідність інструкції, по якій всі змінні функції ставали б за умовчанням глобальними, - це істотно спростило б програмування складних сценаріїв.

Загальний синтаксис визначення функції

У загальному вигляді синтаксис визначення функції такий:

```
function ім'я_функції(арг1 [=зн1], арг2[=зн2] ..., аргN[=знN]) {
операторитіла_функції; }
```

Ім'я функції повинне бути унікальним з точністю до регістру букв. Це означає, що, по-перше, імена MyFunction, myfunction і myFunction вважатимуться однаковими, і, по-друге, не можна перевизначити вже існуючу функцію (стандартну чи ні - не важливо), та зате дозволено давати функціям такі ж імена, як і змінним в програмі (звичайно, без знаку $ на початку). Список аргументів, як легко побачити, складається з декількох перерахованих через кому змінних, кожну з яких потрібно буде задати при виклику функції (втім, тих параметрів для яких привласнено через знак рівності значення за умовчанням (позначення =знN), можна буде опустити). Звичайно, якщо у функції не повинно бути аргументів зовсім (як це зроблено, наприклад, у вбудованої функції time(), що видає поточний час), то слід залишити порожні дужки після її імені, наприклад:

```
function simpleFunction() { ... }
```

У фігурних дужках міститься тіло функції. У нім можуть бути будь-які інструкції, включаючи навіть інструкції визначення інших функцій. Якщо функція повинна повертати в програму, що її викликала, якесь значення, необхідно використовувати оператор return. Якщо ж вона повинна відпрацювати без повернення значень, то оператора return можна і не вказувати або вказувати без задання значення, що повертається.

Інструкція return

Синтаксис інструкції return абсолютно той же, що і в мові C, за винятком однієї дуже важливої деталі. Якщо в C функції дуже рідко повертають великі об'єкти (наприклад, структури), а масиви вони не можуть повернути зовсім, то в PHP можна використовувати return абсолютно для будь-яких об'єктів (якими б великими вони не були), причому без помітної втрати швидкодії. Ось приклад простої функції, що повертає квадрат свого аргументу:

```
function mySqrt($n){ return $n*$n; }
```

```
$value = mySqrt(4);
echo $value; // виводить 16
echo mySqrt(l0); // виводить 100
```

Відразу декілька значень функції, зрозуміло, повернути не можуть. Проте якщо це все ж таки дуже потрібно, то можна повернути асоціативний масив або ж список, наприклад, так, як представлено у наступному лістингу.

Лістинг 3.1.

```
<?php ## Повернення масиву.
function silly() { return array (1,2,3); }
// привласнює масиву значення array(1,2,3)
$arr = Silly() ;
var_dump ($arr);// виводить масив
// привласнює змінним $a, $b, $c перші значення із списку
list($a, $b, $c) = Silly();
?>
```

В даному прикладі використаний оператор list() для розподілу значень масиву по змінним.

Якщо функція не повертає ніякого значення, тобто інструкції return в ній немає, то вважається, що функція повернула null (лістинг 3.2).

Лістинг 3.2.

```
<?php ## Неявне повернення NULL.
function f() { }
var_dump (f()); // друкує NULL
?>
```

Оголошення і виклик функції

Функцію можна створювати не тільки у визначеному місці програми, але і прямо серед інших інструкцій. Фактично оголошення функції саме є інструкцією. Наприклад, цілком можна було б помістити функцію selectItems(), прямо в середину коду, скажімо, так:

```
echo "Програма...";
```

```
function selectItems($items, $selected=0){ /* ... тіло функції
... */ }
echo "Програма продовжується!";
```

При такому підході транслятор, дійшовши до визначення функції, просто перевірить її коректність і транслюватиме у внутрішнє представлення, але не генеруватиме код для виконання. Далі він перейде до наступної за тілом функції команди. Тільки потім, при виклику функції, інтерпретатор почне виконувати її команди...

Оскільки фази трансляції і виконання в PHP розділені, можна застосовувати виклики функції ще до того, як вона була описана.

Визначення функції нижче за її виклик, звичайно ж, працює тільки в тому випадку, якщо у момент виклику функції її код вже відтранслювався. Наприклад, виклик і опис функції відбуваються в одному і тому ж файлі. Зокрема, якщо спробувати помістити функцію в зовнішній файл, який потім включити по інструкції require_once, Викликати функцію можна лише після даної інструкції.

Краще завжди викликати функції тільки після того, як вони будуть визначені.

Параметри по замовчуванні

Часто трапляється, що у функції повинно бути передано багато параметрів, причому деякі з них задаватимуться абсолютно одноманітно. Наприклад, при розробці selectItems(), другий параметр (поточне вибране значення) задаватиметься не завжди, а може бути і опущений. Використовуючи синтаксис оголошення значень по замовчуванні, вказується інтерпретатору, щоб у разі опущеного параметра він підставив замість нього вказане значення (в даному випадку - 0):

```
function selectItems($items, $selected=0) { ... }
```

Тепер, маючи таку функцію, можна написати в програмі:

```
echo selectItems($names, "Goddard"); // вибраний елемент -
"Goddard"
```

```
echo selectItems($names);  // жоден елемент не вибраний по
замовчуванні
```

Тобто, можна опустити другий параметр у функції selectItem, що виглядатиме так, як ніби його задали рівним 0.

Як видно, значення по замовчуванні для деякого аргументу указується праворуч від його імені через знак рівності. Значення аргументів по замовчувані повинні визначатися справа наліво, причому неприпустимо, щоб після будь-якого з таких аргументів йшов звичайний аргумент. Ось, наприклад, невірний опис:

```
// Помилка! Опускати параметри можна тільки справа наліво!
function selectItems($selected=0, $items) { ... }
// Помилка! Це. не оператор list(), у якому такі речі допустимі,
echo selectItems(, $names);
```

Передача параметрів по посиланню

Механізм, за допомогою якого функції передаються її аргументи наведено у лістингу 3.3.

Лістинг 3.3.

```
<?php ## Передача параметрів по значенню.
function increment($a){
  echo "Поточне значення: $a<br>";
  $a++;
  echo "Після збільшення: $a<br>";
}
# ...
$num = 10;
echo "Початкове значення: $num<br>";
increment($num);
echo "Після виклику функції: $num<br>";
?>
```

Перед початком роботи функції increment() все починається з того, що створюється змінна $a, локальна для даної функції (тобто вона існує тільки усередині неї), і їй привласнюється значення 10 (те, що було у змінній $num). Після цього значення 10 виводиться на екран, величина $a інкременується

(збільшується на 1), і нове значення (11) знову друкується. Оскільки тіло функції закінчилося, відбувається повернення в програму, що її викликала.

При подальшому виведенні змінної $num буде надруковане 10 - і це не дивлячись на те, що в змінній $a до повернення з функції було 11 Це відбувається тому що $a - лише копія $num, а зміна копії, звичайно, ніяк не відбивається на оригіналі.

Якщо потрібно, щоб функція мала доступ не до величини, а саме до самої змінної (переданою їй в параметрах), достатньо при передачі аргументу функції перед його ім'ям поставити & (лістинг 3.4).

Лістинг 3.4.

```
<?php ## Передача параметрів по посиланню (застарілий спосіб).
function increment($a){
  echo "Поточне значення: $a<br>";
  $a++;
  echo "Після збільшення: $a<br>";
}
# ...
$num = 10;
echo "Початкове значення: $num<br>";
increment(&$num); // явна передача по посиланню
echo "Після виклику функції: $num<br>";  // виводить 11!
?>
```

Такий спосіб передачі параметрів історично називається "Передачею по посиланню", в цьому випадку аргумент не є копією змінної, а "посилається" на неї.

Щоб не забувати кожного разу писати & перед змінними, передаючи її функції, існує зручніший синтаксис передачі параметрів по посиланню. А саме, можна символ & перенести прямо в заголовок функції (лістинг 3.5).

Лістинг 3.5.

```
<?php ## Передача параметрів по посиланню (правильний спосіб).
function increment(&$a){ // $a - посилальна
```

```
  echo "Поточне значення: $a<br>";
  $a++;
  echo "Після збільшення: $a<br>";
}
# ...
$num = 10;
echo "Початкове значення: $num<br>";
increment($num); // передача по посиланню
echo "Після виклику функції: $num<br>";  // виводить 11!
?>
```

Змінне число параметрів

Функція може мати декілька параметрів, заданих по замовчуванні. Вони перераховуються справа наліво, і їх завжди фіксована кількість. Проте така схема може влаштувати не завжди.

Наприклад, нехай потрібно написати функцію в стилі echo, тобто функцію, яка приймає один або більше параметрів (скільки саме - невідомо на етапі визначення функції) і виводить їх в окремих рядках (лістинг 3.6).

Лістинг 3.6.

```
<?php ## Змінне число параметрів функції.
function myecho() {
  for ($i=0; $i<func_num_args(); $i++) {
    echo func_get_arg($i)."<br>\n"; // виводиться елемент
  }
}
// відображаємо рядки один під одним
myecho("Меркурій", "Венера", "Земля", "Марс");
?>
```

При описі myecho() вказано порожні дужки як список параметрів, немов функція не отримує жодного параметра. Насправді в PHP при виклику функції можна вказувати параметрів більше, ніж задано в списку аргументів - в цьому випадку ніякі попередження не виводяться. "Зайві" параметри як би ігноруються,

в результаті порожні дужки в myecho() дозволяють передати їй скільки завгодно параметрів.

Для того, щоб все ж таки мати доступ до "проігнорованих" параметрів, існують три вбудовані в PHP функції:

```
int  func_num_args ()
```

Повертає загальне число аргументів, переданих функції при виклику.

```
mixed  func_get_arg (int  $num)
```

Повертає значення аргументу з номером $num, заданого при виклику функції. Нумерація, як завжди, починається з нуля.

```
list  func_get_args ()
```

Повертає список всіх аргументів, вказаних при виклику функції. Найчастіше застосування цієї функції виявляється практично зручніше, ніж перших два.

Переписавши приклад із застосуванням останньої функції отримаємо (лістинг 3.7).

Лістинг 3.7.

```
<?php ## Використання fuct_get_args().
function myecho() {
  foreach (func_get_args() as $v) {
    echo "$v<br>\n"; // виводиться елемент
  }
}
// відображаємо рядки однин під інший
myecho("Меркурій", "Венера", "Земля", "Марс");
?>
```

Тут використовується цикл foreach для перебору аргументів, внаслідок чого програма спрощується.

Локальні змінні

В багатьох прикладах, що наводяться вище, розглядались аргументи функції (які передаються по значенню, а не по посиланню) як якісь тимчасові

об'єкти, які створюються у момент виклику і зникають після закінчення функції. Наприклад:

```
$a = 100;   // глобальна змінна, рівна 100
function test($a){
  echo $a; // виводиться значення параметра $a
  // Параметр $a не має до глобальної змінної $a ніякого
відношення!
  $a++;    // змінюється локальна копія значення, переданого в $a
}
test(l); // виводить 1
echo $a; // виводить 100 - глобальна змінна $a не змінилася
```

Насправді такими ж властивостями володітимуть не тільки аргументи, але і всі інші змінні, ініціалізовані або використовувані всередині функції. Сукупність таких змінних називають контекстом функції (або областю видимості усередині функції). Розглянемо приклад з лістингу 3.8.

Лістинг 3.8.

```
<?php ## Локальні змінні,
function silly() {
  $i = rand(); // записує в $i випадкове число
  echo "$i<br>"; // виводить його на екран
  // Ця $i не має до глобальної $i ніякого відношення!
}
// Виводить в циклі 10 випадкових чисел,
for ($i=0; $i!=10; $i++) silly();
?>
```

Тут змінна $i у функції буде не тією змінною $i, яка використовується в програмі для організації циклу. Тому, власне, цикл і пропрацює тільки 10 "витків", надрукувавши 10 випадкових чисел (а не крутитиметься довго і наполегливо, поки "в рулетці" функції rand() не випаде 10).

Глобальні змінні

В PHP є спосіб, за допомогою якого функції можуть добратися і до будь-якої глобальної змінної в програмі (не рахуючи, звичайно, передачі параметра по

посиланню). Для цього вони повинні виконати певні дії, а саме: до першого використання в своєму тілі зовнішньої змінної оголосити її "глобальною" за допомогою інструкції global:

```
global $variable;
```

У лістингу 3.9 приведений приклад, який показує зручність використання глобальних змінних усередині функції.

Лістинг 3.9.

```
<?php ## Глобальні змінні у функції.
$monthes = array(
  1  => "Січень",
  2  => "Лютий",
  // ...
  12 => "Грудень"
);
// Повертає назву місяця по його номеру. Нумерація починається з
1!
function getMonthName($n){
  global $monthes;
  return $monthes[$n];
}
// Застосування.
echo getMonthName(2); // виводить "Лютий"
?>
```

Масив $monthes, що містить назви місяців, досить об'ємистий. Тому описувати його прямо у функції було б, м'яко кажучи, незручно - він би тоді створювався при кожному виклику функції. В той же час функція getMonthName() є непоганим засобом для приведення номера місяця до його словесного еквівалента (що може бути потрібно в багатьох програмах). Вона має єдиний і зрозумілий параметр: це номер місяця.

Масив $GLOBALS

В принципі, є і другий спосіб дістатися до глобальних змінних. Це використання вбудованого в мову масиву $GLOBALS. Останній є хешом, ключі якого є імена глобальних змінних, а значення - їх величини.

Даний масив доступний з будь-якого місця в програмі - у тому числі і з тіла функції, і його не потрібно ніяк додатково оголошувати. Отже, приведений вище приклад можна переписати лаконічніше:

```
// Повертає назву місяця по його номеру. Нумерація починається з
1!
function getMonthName ($n) { return $GLOBALS["monthes"][$n]; }
```

Тут знову стикаємося з тим, що не тільки змінні, але навіть і масиви можуть мати абсолютно будь-яку структуру, якою б складною вона не була.

Припустимо, що в програмі є асоціативний масив $A, елементи якого - двовимірні масиви чисел. Тоді доступ до якого-небудь осередку цього масиву з використанням $GLOBALS міг би виглядати так:

```
$GLOBALS["A"]["First"][10][20];
```

Тобто вийшов чотиривимірний масив!

Щодо $GLOBALS слід додати ще декілька корисних відомостей.

Як вже було помічено, цей масив спочатку є глобальним для будь-якої функції, а також для самої програми. Так, цілком допустимо його використовувати не тільки в тілі функції, але і в будь-якому іншому місці.

З масивом $GLOBALS допустимі не всі операції, дозволені із звичайними масивами. А саме, не можна:

- привласнити цей масив якій-небудь змінній цілком;
- використовуючи оператор =; як наслідок, передати його функції "по значенню" - можна передавати тільки по посиланню.

Проте решта операцій допустима. Можна при бажанні, наприклад, поодинці перебрати у нього всі елементи і, скажімо, вивести їх значення на екран (за допомогою циклу foreach).

Додавання нового елементу в $GLOBALS рівнозначно створенню нової глобальної змінної, а виконання операції unset про для нього рівносильне знищенню відповідної змінної.

Як працює інструкція global

Конструкція global $a говорить про те, що змінна $a є глобальною, тобто є синонімом глобальної $a. Синонім в термінах PHP - це посилання. Виходить, що global створює посилання? Так, саме так. А ось як це сприймається транслятором:

```
function test () { global $a; $a = 10; }
```

Приведене визначення функції test() повністю еквівалентно наступному опису:

```
function test () (
  $a = &$GLOBALS['a'];
  $a = 10;
}
```

З другого фрагменту виходить, що оператор unset($a) в тілі функції (лістинг 3.10) не знищить глобальну змінну $a, а лише "відв'яже" від неї посилання $a. Точно те ж саме відбувається і в першому випадку.

Лістинг 3.10.

```
<?php ## Особливості global.
$a = 100;
function test() {
  global $a;
  unset($a);
}
test();
echo $a; // виводить 100, тобто справжня $a не була видалена в
test()!
?>
```

Як же видалити глобальну $a з функції? Існує тільки один спосіб: використовувати для цієї мети $GLOBALS[‘a‘]. От як це робиться:

```
function deleter() { unset($GLOBALS['a']); }
$a = 100;
deleter ();
echo $a; // Попередження: змінна $a не визначена!
```

Завдання

Напишіть програму для обчислення виразу. Невідомі змінні вводиться користувачем.

0. Дано дійсне k. Обчислити

$$z(2k+1) - z^2(2k-1) + \sqrt{z(k)} \quad , \quad \partial e$$

$$z(x) = \begin{cases} \dfrac{\cos x + 1}{\sin^2 x + e^x} & , \quad |x| \geq 1 \\ \dfrac{1}{e^x} \displaystyle\sum_{j=0}^{7} (-1)^j \frac{x^j}{j!} & , \quad |x| < 1 \end{cases}$$

1. Дано дійсне t. Обчислити

$$s(2t+1) + 2s(t^2) + \sqrt{s(1)} \quad , \quad \partial e$$

$$s(x) = \begin{cases} \dfrac{\cos^2 x + 1}{e^x} & , \quad |x| \geq 1,\ x = 0 \\ \dfrac{1}{\sin 2x} \displaystyle\sum_{k=0}^{4} \frac{2^{2k+1} x^{2k+1}}{(2k+1)!} & , \quad |x| < 1,\ x \neq 0 \end{cases}$$

2. Дано дійсне q. Обчислити

$$h(q+1) + h(q+1) + h^2(q^2)) \quad , \quad \partial e$$

$$h(x) = \begin{cases} \dfrac{\cos x + 1}{\cos^2 x + 1} & , \quad |x| \geq 1 \\ \dfrac{1}{\cos x} \displaystyle\sum_{i=0}^{6} \frac{x^{2i}}{(2i)!} & , \quad |x| < 1 \end{cases}$$

3. Дано дійсне p. Обчислити

$$z(p^2+1) - z(p^2-1) + 2z(p) \quad , \quad \partial e$$

$$z(x) = \begin{cases} \dfrac{\sin x + 1}{\cos^2 x + e^x} & , \quad |x| \geq 1 \\ \dfrac{1}{e^{x^2}} \displaystyle\sum_{k=0}^{6} \frac{2^k x^k}{k!} & , \quad |x| < 1 \end{cases}$$

4. Дано дійсне r. Обчислити

$$h(r+1) + h^2(r^2+1) + 1 \quad , \quad \partial e$$

$$h(x) = \begin{cases} \dfrac{\cos x + 1}{e^x} & , \quad |x| \geq 1,\ x = 0 \\ \dfrac{1}{\sin x^2} \displaystyle\sum_{n=0}^{6} \frac{x^{4n+2}}{(2n+1)!} & , \quad |x| < 1,\ x \neq 0 \end{cases}$$

5. Дано дійсне k. Обчислити

$$j(k)+j^2(k-1)+2j(1) \quad , \quad \partial e$$

$$j(x)=\begin{cases}\dfrac{\sin x+1}{\cos^2 x+e^x} & , \quad |x|\ge 1\\ \dfrac{1}{\cos 2x}\displaystyle\sum_{k=0}^{7}\frac{2^{2k}x^{2k}}{(2k)!} & , \quad |x|<1\end{cases}$$

6. Дано дійсне z. Обчислити

$$k(z^2+1)-k(z^2-1)+2k(z) \quad , \quad \partial e$$

$$k(x)=\begin{cases}\dfrac{e^x+\sin x}{\cos^2 x+1} & , \quad |x|\ge 1\\ \dfrac{1}{e^x}\displaystyle\sum_{i=0}^{5}\frac{x^i}{i!} & , \quad |x|<1\end{cases}$$

7. Дано дійсне r. Обчислити

$$s(\sqrt{r}+1)-s^2(\sqrt{r}-1)+1 \quad , \quad \partial e$$

$$s(x)=\begin{cases}\dfrac{1+x^2+\sqrt{|\sin x|}}{\sin^2 2x^2+1} & , \quad |x|\ge 1\\ \displaystyle\sum_{i=0}^{5}\frac{x^i}{i!}+\sum_{k=1}^{6}\frac{x^k}{k!} & , \quad |x|<1\end{cases}$$

8. Дано дійсне q. Обчислити

$$p(2q+1)+p^2(q^2-1)+\sqrt{p(1)} \quad , \quad \partial e$$

$$p(x)=\begin{cases}\dfrac{\sin x+\cos x}{\sin^2 x+e^x} & , \quad |x|\ge 1\\ \dfrac{1}{\cos x}\displaystyle\sum_{j=0}^{4}(-1)^j\frac{x^{2j}}{(2j)!} & , \quad |x|<1\end{cases}$$

9. Дано дійсне f. Обчислити

$$y(\frac{f}{2}) + y^2(f+1) + y(2f) \quad , \quad \partial e$$

$$y(x) = \begin{cases} \dfrac{e^x}{1+e^x+\sin x} & , \quad |x| \geq 1 \\ \displaystyle\sum_{j=0}^{8} (-1)^j \frac{x^{2j}}{(2j)!} & , \quad |x| < 1 \end{cases}$$

10.Дано дійсне t. Обчислити

$$f(2t) + f^2(t^2+1) \quad , \quad \partial e$$

$$f(x) = \begin{cases} \dfrac{\sin 2x + 1}{\sin x + \cos^2 x} & , \quad |x| \geq 1 \\ \displaystyle\sum_{n=0}^{7} \frac{x^{4n+2}}{(2n+1)!} & , \quad |x| < 1 \end{cases}$$

11.Дано дійсне t. Обчислити

$$f(t^2) + 2f(2t+1) \quad , \quad \partial e$$

$$f(x) = \begin{cases} \dfrac{\sin x + 1}{\sin x + \cos x} & , \quad |x| \geq 1 \\ \displaystyle\sum_{i=0}^{8} \frac{x^{2i+1}}{(2i+1)!} & , \quad |x| < 1 \end{cases}$$

12.Дано дійсне z. Обчислити

$$g(2z) + g^2(2z+z^2) + g(1{,}5) \quad , \quad \partial e$$

$$g(x) = \begin{cases} \dfrac{\sin x}{\cos x + 2} & , \quad |x| \geq 1 \\ \displaystyle\sum_{n=0}^{6} \frac{x^{2n}}{n!} & , \quad |x| < 1 \end{cases}$$

13.Дано дійсне f. Обчислити

$$t(f^2) + 2t(2f+1) + \sqrt{t(1)} \quad , \quad \partial e$$

$$t(x) = \begin{cases} \dfrac{\cos x + 1}{e^x + \sin^2 x} & , \quad |x| \geq 1 \\ \displaystyle\sum_{j=0}^{5} (-1)^j \frac{x^{2j+1}}{(2j+1)!} & , \quad |x| < 1 \end{cases}$$

14.Дано дійсне p. Обчислити

$$z(p^2)+2z(2p+p^2)+1 \quad , \quad \partial e$$

$$z(x)=\begin{cases}\dfrac{\cos 2x+1}{\cos x+\sin^2 x} & , \quad |x|\ge 1\\ \displaystyle\sum_{k=0}^{6}\frac{2^{2k+1}x^{2k+1}}{(2k+1)!} & , \quad |x|<1\end{cases}$$

15.Дано дійсне t. Обчислити

$$s(t^2+1)+2s^2(1-t)+s(1) \quad , \quad \partial e$$

$$s(x)=\begin{cases}\dfrac{\cos^2 x+1}{e^x} & , \quad |x|>1\\ \displaystyle\sum_{k=0}^{4}\frac{x^{2k+1}}{(2k+1)!} & , \quad |x|\le 1\end{cases}$$

Хід роботи

1. Створити новий файл у текстовому редакторі (Dreamweaver, Notepad або іншому).
2. Написати розв’язок завдання.
3. Зберегти програму на диск під новим ім`ям 1.php у папку <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB03
4. Перевірити виконання скрипта у браузері.
5. При виникненні помилки – відкоригувати програму та виконати її.
6. Оформити звіт по виконаній роботі.

Методичні рекомендації

Для введення даних використовуються параметри в URL, для виконання програми потрібно у браузері ввести <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB03\1.php?k=<зн1>.

Розв’язок завдання (текст програми):

```
<?php
function fuct($n){
  if ($n < 2) return 1;
  return $n*fuct($n-1);
}
```

```
function z($x){
  if (abs($x)>=1) return (cos($x)+1)/(pow(sin($x),2)+exp($x));
  else {
    $s = 0;
    for($j=0;$j<=7;$j++){
      $s += pow(-1, $j)*pow($x, $j)/fuct($j);
    }
    return 1/exp($x)*$d;
  }
}

$k = $_GET['k'];
echo "Результат обчислення: ", z(2*$k+1)- pow(z(2*$k-
1),2)+sqrt(z($k));
?>
```

Контрольні запитання

1. Як всередині функції звертатись до глобальних змінних?
2. Які є способи передачі параметрів?
3. Як реалізується вихід із функції?
4. Яким чином можна передати дані з функції у програму через параметри?
5. Як реалізувати параметри по замовчуванні та які є обмеження?

Тема 4: Робота із рядками

Мета роботи: навчитись працювати із символьними рядками у PHP.

Теоретичні відомості

Рядки в PHP - одні з основних об'єктів. Вони можуть містити текст разом з символами форматування або навіть бінарні дані. Визначення рядка в лапках або апострофах може починатися на одному рядку, а завершуватися - на іншому. Ось приклад, який синтаксично абсолютно коректний:

```
$multiline = "Це текст, що починається на одному рядку і
продовжується на іншому, третьому і т. д.";
```

Рядок в апострофах

Якщо рядок поміщений в апострофи (наприклад, 'рядок'), то він трактується майже в точності так само, як записаний, за винятком двох спеціальних послідовностей символів:

- послідовність \' трактується PHP як апостроф і призначена для вставки апострофа в рядок, взятий в апострофи: 'д\'Артаньян';
- послідовність \\ трактується як один зворотній слеш і дозволяє вставляти у рядок цей символ: 'C:\\m2transcript.txt'.

Решта символів позначають самі себе, зокрема, символ $ не має ніякого спеціального значення (звідси витікає, що змінні усередині рядка, взятого в апострофи, не інтерполюються, тобто їх значення не підставляються).

Рядок в лапках

В порівнянні з апострофами лапки "ліберальніші". Тобто, набір спеціальних метасимволів, які, будучи поміщені в лапки, визначають той або інший спеціальний символ, набагато багатший.

Ось деякі з них:

- \n позначає символ нового рядка;

- \r позначає символ повернення каретки;
- \t позначає символ табуляції;
- \$ позначає символ $, щоб наступний за ним текст випадково не був інтерпольований, як змінна;
- \" позначає лапку;
- \\ позначає зворотній слеш;
- \xNN позначає символ з шістнадцятковим кодом NN.

Змінні в рядках інтерполюються. Наприклад:

```
$hello = "Привіт"; echo "$hello світ!"
```

Цей фрагмент виведе

```
Привіт світ!
```

тобто символи `$hello` у рядку були замінена на значення змінної `$hello` (цьому посприяв знак долара, що передує будь-якій змінній).

Ще один приклад.

```
$SOME = "Приві";
echo "$SOMEт світ!";
```

Ми очікуємо, що виведеться знову той же самий рядок. Але як PHP дізнається, чи мали на увазі змінну $SOME або ж змінну $SOMEт? Очевидно, ніяк. Запустивши фрагмент, переконуємося, що він генерує повідомлення про те, що змінна `$someт` не визначена. Для вказання інтерпретатору яку саме змінну потрібно використовувати потрібно скористатись фігурними дужками:

```
$SOME = " Приві ";
echo $SOME."т світ!";  // один спосіб
echo "{$SOME}т світ!"; // інший спосіб
echo "${SOME}т світ!"; // третій спосіб!
```

Найбільш перспективний, варіант - це {$SOME}, бо таким методом можна вставляти в рядок не тільки значення змінних, але також і елементи масивів і властивості об'єктів:

```
$action = array("first"=>"Перший", "second"=>"Другий");
echo "Вибраний елемент: {$action['first']}";
```

Варто звернути увагу на апострофи, які використовуються для обрамлення ключа масиву усередині конструкції {}. Якщо опустити їх, інтерпретатор виведе попередження.

Такий код:

```
echo "Вибраний елемент: $action[first]";
```

виведе такий же рядок, але цього разу вже без попереджень.

Here-документ

У четвертій версії PHP з'явився і ще один спосіб запису рядкових констант, який історично називається here-документом (вбудований документ). Фактично він є альтернативою для запису багаторядкових констант. Виглядає це так:

```
$name = "Петренко Іван";
$text = <<MARKER
Далі йде якийсь текст можливо, із змінними, які інтерполюються:
наприклад $name буде інтерпольована тут.
MARKER;
```

Рядок MARKER може бути будь-яким алфавітно-цифровим ідентифікатором, що не зустрічається в тексті here-документа у вигляді окремого рядка. Синтаксис накладає два обмеження на here-документи:

- після <<MARKER і до кінця рядка не повинні йти ніякі непробільні символи;
- завершуючий рядок MARKER, - повинен закінчуватися крапкою з комою, після якої до кінця рядка не повинно бути ніяких інструкцій.

Обмеження настільки ускладнюють свободу при використанні here-документів, що доцільність використання такого методу є сумнівною.

Базові функції

```
int strlen (string $st)
```

Одна із найбільш корисних функцій. Повертає просто довжину рядка, тобто кількість символів, які містяться в $st. Рядок може містити довільні символи, в тому числі і з нульовим кодом. Функція strlen буде працювати правильно і з такими рядками.

```
int strpos(string $where, string $what, int $from)
```

Робить спробу знайти в рядку $where підрядок (тобто послідовність символів) $what і у випадку успіху повертає позицію (індекс) цього підрядка у рядку. Перший символ рядка, має індекс 0. Необов'язковий параметр $from можна задавати, якщо пошук потрібно вести не з початку рядка $where, а з якоїсь іншої позиції. У цьому випадку потрібно позицію передати в $from. Якщо підрядок знайти не вдалось, функція повертає false.

```
int strrpos(string $where, char $what)
```

Дана функція подібна до strpos(), вона шукає у рядку $where останню позицію, в якій зустрічається підрядок $what. У випадку, коли співпадіння не знайдено, повертається false.

Робота з підрядками

Тут описуються функції, які дозволяють працювати у програмі з частинами рядків (підрядками).

```
string substr(string $str, int $start [,int $length])
```

Дана функція також застосовується досить часто. Її призначення – повертати частину рядка $str, починаючи із позиції $start і довжиною $length. Якщо $length не задана, то розуміється підрядок від $start до кінця рядка $str. Якщо $start більше, ніж довжина рядка, або ж значення $length рівне нулю, то повертається пустий підрядок.

Проте ця функція може робити ще деякі корисні речі. Наприклад, якщо передати в $start від'ємне число, то буде рахуватись, що це число є індексом підрядка, але тільки відраховується з кінця $str (наприклад, -1 означає "починаючи з останнього символу рядка"). Параметр $length, якщо він заданий, також може бути від'ємним. У цьому випадку останнім символом у рядку, що

повертається функцією буде символ із $str з індексом $length, визначеним від кінця радка.

Заміна

Перелічені нижче функції частіше за все є корисними, коли потрібно проводити однотипні операції заміни з блоками тексту, які задані у рядковій змінній.

```
string str_replace (string $from, string $to, mixed $text)
```

Замінює у рядку $text всі входження підрядка $from (із врахуванням регістру) на $to і повертає результат. Вхідний рядок, переданий третім параметром, при цьому не змінюється. Ця функція працює значно швидше, чим більш універсальні ereg_replace () і preg_replace(), і її часто використовують, якщо немає необхідності в якихось екзотичних правилах пошуку підрядка.

Наприклад, ось як можна замінити усі символи переводу рядка на їх HTML-еквівалент – тег
:

```
$st = str_replace("\n", "<br>\n", $st);
```

Те, що у рядку
\n також є символ переводу рядка, ніяк не впливає на роботу функції, тобто функція виконує лише однократний прохід по рядку.

Параметр $text описаний вище як mixed. Справа у тому, що можна передавати замість нього цілий масив рядків, а не тільки один-єдиний рядок. Якщо $text – масив, то заміна виконується у кожному його елементі, а повертає функція результуючий список.

```
string str_ireplace(string $from, string $to, string $text)
```

Дана функція появилась лише в PHP 5. Вона працює так, як і str_replace(), але тільки замінює рядки без врахування регістру символів.

```
string substr_replace(string $text, string $to, int $start [,int
$len])
```

Функція призначена для заміни у рядку $text частини, яка починається з позиції $start і довжиною $len. Ця частина замінюється на значення параметра $to.

Завдання

Завдання 1

0. Дано: послідовність символів s1,…,sn.. Вияснити, чи є серед цих символів пара сусідніх букв “но” або ”он”.
1. Дано: послідовність символів s1,…,sn.. Підрахувати загальне число входжень символів “+”, “-”, “=” в послідовність s1,…,sn.
2. Дано послідовність символів s1,…,sn. Визначити число входжень в послідовність групи букв “abc” .
3. Дано послідовність символів s1,…,sn. Вияснити, чи є в цій послідовності такі елементи si, si+1 що si – це кома (“,”), а si+1 – тире (“–”), та обчислити їх кількість.
4. Дано послідовність символів s1,…,sn. Визначити число входжень в послідовність групи букв “школа” .
5. Дано послідовність символів s1,…,sn. Вияснити, чи є серед цих символів четвірка сусідніх однакових символів.
6. Дано послідовність символів s1,…,sn. Вияснити, чи зустрічається в даній послідовності група з чотирьох знаків оклику, які стоять підряд .
7. Дано: послідовність символів s1,…,sn. Підрахувати, скільки разів серед даних символів зустрічається кожний символ , що входить в слово “BASIC”.
8. Дано послідовність символів s1,…,sn. Отримати число, рівне першому номеру i, для якого символи si-1, si, співпадають з буквою “Z”. Якщо такої пари символів в даній послідовності немає, то відповіддю повинно бути число 0.
9. Дано: послідовність символів s1,…,sn.. Підрахувати загальне число входжень символів “1”, “2”, “3”, “4” в цю послідовність.
10. Дано: послідовність символів s1,…,sn. Вияснити, чи є в цій послідовності такі символи si-1, si+1 що si-1 – це знак “-”, а si+1 – це знак “+”.
11. Дано: послідовність символів s1,…,sn. Підрахувати, скільки разів серед даних символів зустрічається символ “0”, і скільки разів – символ “1”.
12. Дано: послідовність символів s1,…,sn. Вияснити, чи є серед цих символів група si-2, si, si+2 однакових символів.
13. Дано: послідовність символів s1,…,sn. Визначити число входжень в послідовність групи букв “END”.
14. Дано: послідовність символів s1,…,sn. Вияснити, чи зустрічається в даній послідовності група з чотирьох символів “5”, які стоять підряд .
15. Дано: послідовність символів s1,…,sn. Вияснити, чи є серед цих символів пара сусідніх букв “ДА” або ”ТА”.

Завдання 2

0. Дано натуральне k і послідовність символів s1,…,sn Знайти кількість знаків оклику, які знаходяться між символом sk і першою після sk крапкою.

1. Дано натуральне k і послідовність символів s1,…,sn Знайти номер другої коми, яка стоїть після символу sk.
2. Дано натуральне k і послідовність символів s1,…,sn Знайти позицію третього знаку оклику, який стоїть після символу sk.
3. Дано натуральне r і послідовність символів s1,…,sn Знайти номер передостанньої крапки, яка стоїть перед символом sr.
4. Дано: послідовність символів s1,…,sn Підрахувати кількість слів, довжина яких рівна 2. (слово – це група символів, які відрізняються від пробілу і йдуть підряд).
5. Дано натуральне r і послідовність символів s1,…,sn Знайти номер останньої крапки, яка стоїть перед символом sr.
6. Дано натуральне p і послідовність символів s1,…,sn Знайти кількість знаків запитання, які знаходяться між першою комою і символом sp.
7. Дано натуральне p і послідовність символів s1,…,sn Знайти кількість знаків “ * ”, які знаходяться між третьою крапкою і символом s2p-1.
8. Дано: послідовність символів s1,…,sn Знайти кількість знаків оклику, які знаходяться між першою і останньою крапкою.
9. Дано: послідовність символів s1,…,sn Знайти кількість знаків оклику, які знаходяться між другою і третьою крапкою.
10. Дано: послідовність символів s1,…,sn і натуральне число k. Переставити символи в порядку: sk, sk-1 ,sk-2 ,…,s1, sn , sn-1, sn-2, …, sk+1.
11. Дано: послідовність символів s1,…,sn і натуральне число k. Переставити символи в порядку: s2, s1 ,s4 ,…,s3 , s6 , s5 , , …, sn s2n-1.
12. Дано послідовність символів s1,…,sn. Виключити з послідовності кожний третій символ, який не співпадає з символом“ a ”,
13. Дано: послідовність символів s1, s2 ,s3 , …,s2т. Переставити символи в порядку: s1 , s3 , s5 , …, s2n-1, s2 , s4 , s6 , …, s2n .
14. Дано натуральне k і послідовність символів s1,…,sn Знайти кількість знаків “ + ”, які знаходяться між символом sk і першою після sk комою.
15. Дано натуральне k і послідовність символів s1,…,sn Знайти номер другого символу “+”, який стоїть після символу s2k.

Хід роботи

1. Створити новий файл у текстовому редакторі (Dreamweaver, Notepad або іншому).
2. Написати розв’язок завдань.
3. Зберегти програму на диск під новим ім`ям 1.php у папку <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB04
4. Перевірити виконання скрипта у браузері.
5. При виникненні помилки – відкоригувати програму та виконати її.
6. Оформити звіт по виконаній роботі.

Методичні рекомендації

Для введення даних використовуються параметри в URL, для виконання програми потрібно у браузері ввести <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB01\1.php?k=<зн1>&s=<зн2>.

Розв’язок завдання (текст програми):

```
<?php
  $k = $_GET['k'];
  $s = $_GET['s'];
/* Завдання 1 */
if ((strpos($s, 'он') !== false) || (strpos($s, 'но') !== false))
$mess = "Так"; else $mess = "Ні";;
echo "Завдання 1: $mess<br/><br/>";

/* Завдання 2 */
$dotpos = strpos($s, '.', $k-1);
if ($dotpos !== false) {
  $s = substr($s, $k, $dotpos-($k-1)-1);
  $mess = "Символ '!' входить".substr_count($s, '!')." раз.";
} else $mess = "Не знайдено крапки після $k-го символу";
echo "Завдання 2: $mess";
?>
```

Контрольні запитання

1. Як визначити довжину рядка?
2. Як знайти позицію входження деякого підрядка у рядок?
3. Як отримати підрядок із рядка?
4. Як замінити усі входження одного підрядка у рядок іншим?
5. Яким чином можна отримати доступ до певного символу рядка?

Тема 5: Робота з датою та часом

Мета роботи: навчитись працювати з датою та часом в PHP.

Теоретичні відомості

В PHP присутній повний набір засобів, які призначені для роботи з датами і часом в різних форматах. Додаткові модулі (які входять в дистрибутив PHP і є стандартними) дозволяють також працювати із календарними функціями і календарями різних народів світу. Далі представлені тільки самі популярні з цих функцій.

```
string date (string format [, int timestamp])
```

Повертає рядок, який відформатований відповідно до рядка формату з використанням заданого цілочисельного timestamp або поточного локального часу, якщо timestamp ("штамп часу") не заданий.

Правильний діапазон значень для timestamp зазвичай: з Fri, 13 Dec 1901 20:45:54 GMT до Tue, 19 Jan 2038 03:14:07 GMT. (Це дати, відповідні максимальному і мінімальному значенню 32-бітового цілого числа). У Windows цей діапазон обмежений датами від 01-01-1970 до 19-01-2038.

У рядку формату розпізнаються наступні символи:

Символ формату	Опис	Приклад даних, які повертаються
День		
d	день (число) місяця, 2 цифри з ведучим нулем, якщо необхідно	від 01 до 31
D	день тижня, символьний, 3 букви	Fri
j	день (число) місяця без ведучих нулів	від 1 до 31
l ('L' в нижньому регістрі)	повне текстове представлення дня тижня	Friday
N	день тижня, числовий (згідно стандарта ISO-8601) (добавлено в	від 1 (Понеділок) до 7 (Неділя)

	PHP 5.1.0)	
S	простий англійський суфікс для дня (числа) місяця, 2 символу	"st", "nd", "rd" або "th"
w	день тижня, числовий	від 0 (Неділя) до 6 (Субота)
z	день року	від 0 до 365
Тиждень		
W	номер тижня в році, тижні починаються з понеділка (стандарт ISO-8601) (добавлено в PHP 4.1.0)	42 (42-ий тиждень у році)
Місяць		
F	повна назва місяця	від January до December
m	номер місяця, 2 цифри з ведучим нулем, якщо необхідно	від 01 до 12
n	номер місяця без ведучих нулів	від 1 до 12
M	назва місяця, 3 букви	від Jan до Dec
t	кількість днів в даному місяці	від 28 до 31
Рік		
L	вказує, чи високосний рік	1 якщо високосний, 0 в іншому випадку
o	номер року (стандарт ISO-8601), приймає те ж значення що і Y, за винятком того, що якщо число тижнів ISO (W) належить до попереднього або наступного року, використовується цей рік (добавлено в PHP 5.1.0)	Приклад: 1999 or 2003
Y	рік, 4 цифри	Приклад: 1999
y	рік, 2 цифри	Приклад: 99
Час		
a	Нижній регістр до полудня і після полудня	am або pm
A	Верхній регістр до полудня і після полудня	AM або PM
B	час Swatch Internet	від 000 до 999
g	година, 12 годинний формат без	від 1 до 12

	ведучих нулів	
G	година, 24 годинний формат без ведучих нулів	від 0 до 23
h	година, 12 годинний формат з ведучими нулями	від 01 до 12
H	година, 24 годинний формат з ведучими нулями	від 00 до 23
i	хвилини з ведучими нулями	від 00 до 59
s	секунди з ведучими нулями	від 00 до 59
U	мілісекунди (добавлено в PHP 5.2.2)	Приклад: 654321
Часова зона		
e	Ідентифікатор часової зони (добавлено в PHP 5.1.0)	Приклад: UTC, GMT, Atlantic/Azores
I(велика i)	Визначає чи дата в зимовому чи літньому часі	1, якщо Daylight Savings Time, 0 в іншому випадку
O	Різниця з часом по Грінвічу, в годинах	Приклад: +0200
P	Різниця з часом по Грінвічу, в годинах з двокрапка між годинами і хвилинами (добавлено в PHP 5.1.3)	Приклад: +02:00
T	встановлений часовий пояс на даній машині, скорочений формат	Приклад: "EST" або "MDT"
Z	зміщення часового пояса, в секундах. Зміщення часових поясів до заходу від UTC завжди від’ємне, а для поясів до сходу від UTC – завжди додатне.	Від -43200 до 50400
Повні дата/час		
c	дата (стандарт ISO-8601) (добавлено в PHP 5)	2004-02-12T15:19:21+00:00
r	RFC 822 формат дати	Thu, 21 Dec 2000 16:01:07 +0200
U	секунди епохи Unix Epoch (починаючи з January 1 1970 00:00:00 GMT)	

Приклад date():

```
echo date ("l dS of F Y h:i:s A");
echo "July 1, 2000 is on a " . date ("l",
mktime(0,0,0,7,1,2000));
```

Можна запобігти розгортанню розпізнаваного символу в рядку формату, мнемонізуючи (замінюючи) його вставкою перед ним зворотного слеша (\). Якщо символ із зворотним слешем вже є спеціальною послідовністю, може знадобитися також мнемонізувати і зворотний слеш.

Приклад.

```
echo date("l \\t\h\e jS"); // друкує що-небудь вроді 'Saturday
the 8th'
```

Є можливість сумісного використання date() і mktime() для пошуку дат в майбутньому або у минулому.

Приклад date() і mktime().

```
$tomorrow  = mktime (0,0,0,date("m")  ,date("d")+1,date("Y"));
$lastmonth = mktime (0,0,0,date("m")-1,date("d"),  date("Y"));
$nextyear  = mktime (0,0,0,date("m"),  date("d"),  date("Y")+1);
```

Приклад форматування date().

```
/* Сьогодні March 10th, 2001, 5:16:18 pm */
$today = date("F j, Y, g:i a");                 // March 10,
2001, 5:16 pm
$today = date("m.d.y");                         // 03.10.01
$today = date("j, n, Y");                       // 10, 3, 2001
$today = date("Ymd");                           // 20010310
$today = date('h-i-s, j-m-y, it is w Day z ');  // 05-16-17, 10-
03-01, 1631 1618 6 Fripm01
$today = date('\i\t \i\s \t\h\e jS \d\a\y.');   // it is the 10s
day.
$today = date("D M j G:i:s T Y");               // Sat Mar 10
15:16:08 MST 2001
$today = date('H:m:s \m \i\s\ \m\o\n\t\h');     // 17:03:17 m is
month
$today = date("H:i:s");                         // 17:16:17


array getdate ([int timestamp])
```

Повертає асоціативний масив, що містить інформацію дати з timestamp або поточного локального часу, якщо timestamp не заданий, з наступними елементами масиву:

"seconds" - секунди
"minutes" - хвилини
"hours" - години
"mday" - день (число) місяця
"wday" - день тижня цифрою: від 0 - Неділя до 6 - Субота
"mon" - місяць (цифрою)
"year" - рік (цифрою)
"yday" - день року (цифрою); наприклад, "299"
"weekday" - день неділі (текст) повний; наприклад, "Friday"
"month" - місяць (текст) повний; наприклад, "January"

Приклад getdate().

```
$today = getdate();
$month = $today['month'];
$mday = $today['mday'];
$year = $today['year'];
echo "$month $mday, $year";
```

```
int mktime (int hour, int minute, int second, int month, int day,
int year [, int is_dst])
```

Повертає Unix timestamp, що відповідає заданим аргументам. Цей timestamp є довгим цілим числом (long integer), що містить кількість секунд між початком Unix Epoch (January 1 1970) і специфікованим часом.

Аргументи можуть бути пропущені в порядку справа наліво; будь-який пропущений аргумент отримає поточне значення відповідно до локальної дати і часу.

is_dst може бути встановлений в 1, якщо це період з поправкою на літній час, в 0 - якщо немає, або -1 (за умовчанням), якщо не відомо, знаходиться даний час в періоді з поправкою на літній час чи ні. Якщо це не відомо, PHP намагається

визначити це сам. Це може привести до несподіваних (але не некоректних) результатів.

mktime() використовується для перекладу дати в арифметичне значення і перевірки, оскільки автоматично обчислюватиметься коректне значення для введення поза діапазоном значень. Наприклад, кожний з наступних рядків виведе рядок "Jan-01-1998".

Приклад mktime().

```
echo date ("M-d-Y", mktime (0,0,0,12,32,1997));
echo date ("M-d-Y", mktime (0,0,0,13,1,1997));
echo date ("M-d-Y", mktime (0,0,0,1,1,1998));
echo date ("M-d-Y", mktime (0,0,0,1,1,98));
```

Year може бути двох- або 4-значним числом із значеннями 0-69, що відображаються як 2000-2069, і 70-99 - як 1970-1999 (у системах, де time_t це 32-бітове знакове ціле, як в більшості сучасних систем, правильним діапазоном year буде приблизно 1902-2037).

Останній день даного місяця може бути виражений як "0" наступного місяця, а не як -1. Обидва наступні приклади дадуть рядок "The last day in Feb 2000 is: 29".

Приклад: Останній день наступного місяця.

```
$lastday = mktime(0,0,0,3,0,2000);
echo strftime("Last day in Feb 2000 is: %d", $lastday);

$lastday = mktime(0,0,0,4,-31,2000);
echo strftime ("Last day in Feb 2000 is: %d", $lastday);
```

Дата з year, month і day, рівними нулю, розглядається як неприпустима.

```
int strtotime (string time [, int now])
```

Функція отримує рядок, що містить англійський формат дати, і намагається розібрати цей формат в UNIX timestamp відносно timestamp, заданого в параметрі now, або відносно поточного часу, якщо нічого не задано. При невдачі повертає -1.

Приклад strtotime().

```
echo strtotime ("now"), "\n";
echo strtotime ("10 September 2000"), "\n";
echo strtotime ("+1 day"), "\n";
echo strtotime ("+1 week"), "\n";
echo strtotime ("+1 week 2 days 4 hours 2 seconds"), "\n";
echo strtotime ("next Thursday"), "\n";
echo strtotime ("last Monday"), "\n";
```

Завдання

0. Вивести останнє число попереднього місяця.
1. Вивести біжучу дату у форматі `день.місяць.рік`.
2. Вивести біжучу дату у форматі `рік-місяць-день`.
3. Вивести день тижня заданої дати.
4. Вивести кількість днів між двома датами.
5. Визначити та вивести час виконання скрипта.
6. Привести введену дату та час у вигляді стрічки до числового типу.
7. Визначити який по рахунку біжучий тиждень у році.
8. Отримати дату наступної неділі.
9. Вивести дату та час вказуючи корекцію на часовий пояс.
10. Вивести дату пешої суботи у наступному місяці.
11. Вивести перше число наступного місяця.
12. Вивести дату пешої неділі вересня даного року.
13. Визначити кількість днів до заданої дати.
14. Вивести кількість четвергів наступного місяця.
15. Вивести скільки годин до Нового року.

Хід роботи

1. Створити новий файл у текстовому редакторі (Dreamweaver, Notepad або іншому).
2. Написати розв’язок завдання.
3. Зберегти програму на диск під новим ім`ям 1.php у папку <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB05
4. Перевірити виконання скрипта у браузері.

5. При виникненні помилки – відкоригувати програму та виконати її.
6. Оформити звіт по виконаній роботі.

Методичні рекомендації

Розв’язок завдання (текст програми):

```
<?php
$d = mktime(0, 0, 0, date("m"), 0, date("Y"));
echo "Останнє число попереднього місяця: ".date("d.m.Y", $d);
?>
```

Для виконання програми потрібно у браузері ввести <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB05\1.php.

Контрольні запитання

1. Як вивести дату за допомогою функції date?
2. Як сформувати дату за допомогою функції mktime?
3. Які символи формату дати функції date Вам відомі?
4. Яка функція дозволить отримати масив з елементами: число, місяць і т.д.?
5. Як перетворити рядок із датою у дату типу timestamp?

Тема 6: Масиви. Одновимірні, багатовимірні, асоціативні

Мета роботи: Навчитись працювати із масивами в PHP.

Теоретичні відомості

Асоціативні масиви - один з наймогутніших інструментів в PHP. Масиви досить часто реалізуються в інтерпретаторах типу PHP (у Perl асоціативні масиви влаштовані навіть трохи гірше, ніж в PHP).

Масиви - це своєрідні контейнери-змінні для зберігання відразу декількох величин, до яких можна потім швидко і зручно звернутися. Звичайно, ніхто не забороняє взагалі їх не використовувати, а, наприклад, давати своєрідні імена змінним, такі як $al, $a2 і т.д. Але якщо потрібно тримати в пам'яті, скажімо, тисячу таких змінних. Крім того, даний спосіб організації масивів має і ще один недолік - дуже важко перебрати всі значення в циклі, хоча це і можливо:

```
for ($i=0; ; $i++) {
  $v = "a$i";
  if (!isset($$v)) break;
  // робимо що-небудь з $$v
}
```

Тут використовується можливість PHP по роботі з посилальними змінними, яку не рекомендовано де небудь застосовувати. Все це представлено для того, щоб проілюструвати, наскільки незручно буває працювати без масивів.

Нехай у програмі потрібно описати список з декількох імен людей. Можна зробити це так:

```
$namesList[0] = "Петро Іваненко";
$namesList[1] = "Григорій Вовк";
$namesList[2] = "Юрій Грім";
```

Таким чином, поодинці додаємо в масив $namesList елементи, наприклад, пронумеровані від 0. PHP дізнається, що потрібно створити масив по квадратних дужках (потрібно відмітити, що для цього змінну $namesList на початку не потрібно ініціалізувати).

Список можна роздрукувати наступним чином (лістинг 6.1).

Лістинг 6.1.

```
<?php ## Демонстрація роботи із списками.
$namesList[0] = "Петро Іваненко";
$namesList[1] = "Григорій Вовк";
$namesList[2] = "Юрій Грім";
echo "А ось перший елемент масиву: ".$namesList[0]."<hr>";
// Друкуємо в циклі всі елементи масиву.
for($i=0; $i<count($namesList); $i++)
  echo $namesList[$i]."<br>";
?>
```

Як видно, найпростіший спосіб - скористатися циклом for. Кількість елементів в масиві легко можна визначити, задіюючи функцію count() або її синонім sizeof().

Створення масиву "на льоту". Автомасиви

У прикладі з лістингу 6.1, здавалося б, все гладко. За винятком одного невеликого недоліку: кожного разу, додаючи ім'я, потрібно вибирати для нього номер і піклуватися, щоб ненароком не вказати такий що вже існує. Щоб цього уникнути, можна написати ті ж команди так:

```
$namesList[0] = "Петро Іваненко";
$namesList[1] = "Григорій Вовк";
$namesList[2] = "Юрій Грім";
```

В цьому випадку PHP сам почне (звичайно, якщо змінна $namesList ще не існує) нумерацію з нуля і кожного разу додаватиме до лічильника по одиниці, створюючи список. Зрозуміло, можна використовувати дужки [] і не тільки в такому простому контексті, дуже часто вони застосовуються для більш загальної дії - додавання елементу в кінець масиву, наприклад:

```
unset($FNames);
//  про всяк випадок стираємо масив
while ($f = наступне_імя_файлу_в_поточній_папці)
if (розширення_$f_є_txt)
```

```
    $FNames[]=$f;
// тепер $FNames містить список файлів з розширенням txt
```

Якщо ж потрібно створити асоціативний масив, все робиться абсолютно аналогічно, тільки замість цифрових ключів потрібно вказувати рядкові. При цьому слід пам'ятати, що в рядкових ключах букви нижнього і верхнього регістрів вважаються різними. І ще: ключем може бути абсолютно будь-який рядок, що містить пропуски, символи переходу рядка, нульові символи і т.д. Тобто, ніяких обмежень на ключі не накладаються.

Нехай потрібно написати сценарій, який працює, як записник: по логіну абонента він видає його ім'я. Можна організувати базу даних цієї книжки у вигляді асоціативного масиву з ключами - логінами і відповідними ним значеннями імен людей:

```
$names["pups"]  = "Юрій";
$names["hero"]  = "Петро";
$names["first"] = "Сергій";
```

Далі, можна роздрукувати ім'я будь-якого абонента командою:

```
$f = "hero";
echo $names["pups"]." сказав: Привіт, пане ".$names[$f]."...";
```

Тут все працює абсолютно аналогічно спискам, тільки з нецифровими ключами. Проте для цього масиву не можна скористатися циклом for, для виведення всіх персоналій.

Оператор list()

Нехай є деякий масив-список $list з трьома елементами: ім'я людини, його прізвище і вік. Потрібно привласнити змінним $name, $surname і $age ці величини. Це, звичайно, можна зробити так:

```
$name    = $list[0];
$surname = $list[1];
$age     = $list[2];
```

Але набагато витонченіше буде скористатися оператором list(), призначеним якраз для таких цілей:

```
list($name, $surname, $age) = $list;
```

Оператор list() можна застосувати для будь-якої кількості змінних: якщо в масиві не вистачить елементів, щоб їх заповнити, їм просто привласняться невизначені значення.

Якщо потрібні тільки другий і третій елементи масиву $list, то потрібно пропустити перший параметр в операторові list(), ось так:

```
list(, $surname, $age) = $list;
```

Таким чином в змінних $surname і $age отримали прізвище і вік людини, не звертаючи уваги на його ім'я в першому аргументі.

У операторі list() можна пропускати будь-яку кількість елементів, як зліва або справа, так і посередині списку. Головне - не забути проставити потрібну кількість ком.

Списки і асоціативні масиви.

Всі масиви в PHP є асоціативними (зокрема, списки - теж). Асоціативні масиви в PHP є направленими, тобто в них існує певний (і передбачений) порядок елементів, не залежний від реалізації. А значить, є перший і останній елементи, і для кожного елементу можна визначити наступний за ним.

Операція [] завжди додає елемент в кінець масиву, привласнюючи йому при цьому такий числовий індекс, який би не конфліктував з вже наявними в масиві (точніше, вибирається номер, що перевершує всі наявні цифрові ключі в масиві). Будь-яка операція `$змінна[ключ]=значення` завжди додає елемент в кінець масиву, звичайно, за винятком тих випадків, коли ключ вже присутній в масиві. Якщо потрібно змінити порядок проходження елементів в асоціативному масиві, не змінюючи в той же час їх ключів, це можна зробити одним з двох способів: скористатися функціями сортування або створити новий порожній масив і заповнити його в потрібному порядку, перебравши елементи початкового масиву.

Оператор array() і багатовимірні масиви

Повернемося до попереднього прикладу. Необхідно написати програму, яка по логіну деякого користувача видаватиме його ім'я.

```
$names["pups"]  = "Юрій";
$names["hero"]  = "Петро";
```

Тепер можна, написати:

```
echo $names["hero"];   // виведе Петро
echo $names["first"];  // помилка: у масиві немає такого
елементу!
```

Існує другий спосіб створення масивів, що виглядає значно компактніше - це використання оператора array(). Наприклад:

```
// створює порожній масив $names
$names = array();
// створює такий же масив, як в попередньому прикладі з іменами
$names = array("pups"=>"Юрій", "hero"=>"Петро");
// створює список з іменами (нумерація 0, 1, 2)
$namesList = array("Петро Іваненко", "Григорій Вовк", "Юрій
Грім");
```

Значеннями змінних (і значеннями елементів масиву теж) може бути все, що завгодно, зокрема - знову ж таки масив. Так, можна створювати асоціативні масиви (а можна - списки) з будь-яким числом вимірів. Наприклад, якщо окрім імені про користувача відомий також його вік, то можна ініціювати масив $names так:

```
$users["pups"] = array("name"=>"Юрій" , "burn"=>"1992-03-11");
$users["hero"] = array("name"=>"Петро", "burn"=>"1994-09-02");
```

або навіть так:

```
$users= array(
"pups" => array("name"=>"Юрій" , "burn"=>"1992-03-11"),
"hero" => array("name"=>"Петро", "burn"=>"1994-09-02"));
```

До елементів заданого масиву можна дістатись наступним чином:

```
echo $users["pups"]["name"]; // надрукує "Юрій"
echo $users["hero"]["diff"];   // помилка: немає елементу "diff"
```

До речі, асоціативні масиви в PHP зручно використовувати як якісь структури, що зберігають дані.

Операції над масивами

Існує досить багато операцій, які можна виконувати з масивами (на додаток до загальних операцій над змінними).

Доступ по ключу

Асоціативні масиви - об'єкти, які найбільш пристосовані для вибірки з них даних шляхом вказання потрібного ключа. У PHP і для всіх масивів, і для списків використовується один і той же синтаксис. От як це виглядає:

```
echo $names["pups"];  // виводить елемент масиву з ключем "pups"
echo $users["hero"]["name"]; // так використовуються двовимірні масиви
echo (SomeFuncThatReturnsArray())[5]; // ПОМИЛКА! Так не можна!
// От так правильно:
$result = SomeFuncThatReturnsArray();
echo $result[5];
```

Останній приклад показує, що PHP сильно відрізняється від мови C з погляду роботи з масивами: у нім немає такого поняття, як контекст масиву, а значить, не можна застосувати [] безпосередньо до значення, поверненого функцією.

Величина $змінна[ключ] є повноцінним "лівим значенням", тобто може стояти в лівій частині оператора привласнення, від неї можна брати посилання за допомогою оператора &, і т.д. Наприклад:

```
$names["first"] = "Сергій"; // елементу масиву привласнюється рядок "Сергій"
$ref = &$users["pups"]["name"]; // $ref - синонім елементу масиву
$namesList[] = "Григорій Вовк";    // додаємо новий елемент
```

Функція count()

Визначити розмір (кількість елементів) в масиві можна за допомогою стандартної функції count():

```
$num = count($namesList);  // в $num - кількість елементів масиву
```

Функція count() працює не тільки з масивами, але і з об'єктами і навіть із звичайними змінними (для останніх результат виконання count() завжди рівний 1, неначе змінна- це масив з одним елементом). Втім, її дуже рідко застосовують для чого-небудь, відмінного від масиву.

Злиття масивів

Ще одна цікава операція - злиття масивів, тобто створення масиву, що містить як елементи одного, так і іншого масиву. Реалізується це за допомогою оператора +. Наприклад:

```
$good = array("pups"=>"Юрій", "hero"=>"Петро");
$bad = array("first"=>"Сергій", "rio"=>"Григорій");
$all = $good + $bad;
```

В результаті в $all опиниться асоціативний масив, що містить всі 4 елементи, причому порядок проходження елементів залежатиме від порядку, в якому масиви зливаються. Спрямованість масивів примушує оператор + стати некомутативним, тобто $good + $bad не рівно $bad + $good.

Злиття списків

Не потрібно зливати списки за допомогою оператора +. Приклад, програма, представлена в лістингу 6.1.

Лістинг 6.1.

```
<?php ## Неправильне злиття списків.
$good = array("Петро Іваненко", "Григорій Вовк", "Юрій Грім");
$bad  = array("Павло Петренко", "Степан Притула");
$ugly = array("Дмитро Дідух");
$all = $good + $bad + $ugly;
print_r($all);
?>
```

Очікується, що в $all буде масив, що складається з 3 + 2 + 1 = 6 елементів? Це невірно! Виклик print_r () надрукує наступний результат:

```
Array (
[0] => Петро Іваненко
```

```
[1] => Григорій Вовк
[2] => Юрій Грім)
```

Все відбулося так, ніби то $bad і $ugly взагалі не згадувалися. От чому так відбувається. При конкатенації масивів з деякими однаковими елементами (тобто елементами з однаковими ключами) в результуючому масиві залишиться тільки один елемент з таким же ключем - той, який був в першому масиві, і на тому ж самому місці.

Оновлення елементів

Останній факт може злегка спантеличити. Здавалося б, елементи масиву $bad по логіці повинні замінити елементи $good. Проте все відбувається навпаки. Остаточно вибиває з колії наступний приклад:

```
$a = array ('a'=>10, 'b'=>20);
$b = array ('b'=>'new');
$a += $b;
```

Очікується, що оператор += обновить елементи $a за допомогою елементів $b. А марно. В результаті цих операцій значення $a не зміниться.

Змінити елементи в масиві $a можна за допомогою стандартної функції array_merge(), позбавленою вказаного недоліку:

```
$a = array_merge($a, $b);
```

Або ж скористайтеся циклом:

```
foreach ($b as $k=>$v) $a[$k]=$v;
```

Ще декілька слів щодо операції злиття масивів, Ланцюжок

```
$z = $a + $b + $c + ...і т.д.;
```

еквівалентний

```
$z = $a; $z += Sb; $z += Sc; ...і т.д.
```

Як неважко здогадатися, оператор += для масивів робить приблизно те ж, що і оператор += для чисел, а саме - додає в свій лівий операнд елементи, перераховані в правому операнді-масиві, якщо вони ще не містяться в масиві зліва.

Отже, в масиві ніколи не може бути двох елементів з однаковими ключами, тому що всі операції, застосовні до масивів, завжди контролюють, щоб цього не відбулося.

Непрямий перебір елементів масиву

Досить часто при програмуванні на PHP доводиться перебирати всі без виключення елементи деякого масиву

Перебір списку

Якщо наш масив-список, то це завдання не буде особливо обтяжливим:

Лістинг 6.2.

```
<?php ## Перебір списку.
$users = array(
  array("name"=>"Юрій Грім" , "burn"=>"1992-03-11"),
  array("name"=>"Петро Іваненко", "burn"=>"1994-09-02")
);
for($i=0; $i<count($users); $i++)
echo "{$users[$i]['name']} народився {$users[$i]['burn']}<br>";
?>
```

Перебір асоціативного масиву

Нехай є асоціативний масив $burn: його ключі - імена людей, а значення, зіставлені ключам, - наприклад, вік цих людей. Для перебору такого масиву можна скористатися конструкцією, на зразок приведеної в лістингу 6.3.

Лістинг 6.3.

```
<?php ## Перебір асоціативного масиву.
$birth = array(
  "pups" => "1992-03-11"
  "hero" => "1994-09-02"
);
for (reset($birth); ($k=key($birth)); next($birth))
  echo "$k народився {$birth[$k]}<br>";
?>
```

Представлена конструкція спирається на ще одну властивість асоціативних масивів в PHP. А саме, мало того, що масиви є направленими, в них є ще і таке поняття, як поточний елемент. Функція reset() просто встановлює цей елемент на першу позицію в масиві. Функція key() повертає ключ, який має поточний елемент (якщо він указує на кінець масиву, повертається порожній рядок, що дозволяє використовувати виклик key в контексті другого виразу for). Ну а функція next() переміщає поточний елемент на одну позицію вперед.

Насправді, дві прості функції, - reset() і next(), - крім виконання свого основного завдання, ще і повертають деякі значення, а саме:

- функція reset() повертає значення першого елементу масиву (або false, якщо масив порожній);
- функція next() повертає значення елементу, наступного за поточним (або false, якщо такого елементу немає).

Іноді буває потрібно перебрати масив з кінця, а не з початку. Для цього скористайтеся такою конструкцією:

```
for (end($birth); ($k=key($birth)); prev($birth))
 echo "$k народився {$birth[$k]}<br>";
```

Функція end() встановлює позицію поточного елементу в кінець масиву, а prev() пересуває її на один елемент назад.

У PHP є функція current(). Вона дуже нагадує key(), тільки повертає не ключ, а величину поточного елементу (якщо він не указує на кінець масиву).

Недоліки непрямого перебору

Основна перевага непрямого перебору - "читабельність" і ясність коду, а також те, що масив можна перебрати як в однин, так і в інший бік. Проте існують і недоліки.

- Вкладені цикли

Перший недолік досить фундаментальний: не можна одночасно перебирати масив у двох вкладених циклах або функціях. Причина цілком очевидна: другий

вкладений цикл for "зіпсує" положення поточного елементу у першого циклу for. На жаль, цю проблему ніяк не можна обійти (хіба що зробити копію масиву, і у внутрішньому циклі працювати з нею). Проте практика показує, що такі перебори зустрічаються украй рідко.

- Нульовий ключ

  Якщо в масиві зустрінеться ключ 0 то приклад:

```
for (reset($birth); ($k=key($birth)); next($birth))
echo "$k народився {$birth[$k]}<br>";
```

буде працювати не коректно, тут вираз ($k=key($birth)), буде рівний нулю, і цикл обірветься.

Для коректної роботи циклу доведеться писати так:

```
for(reset($birth); ($k=key($birth))!==false; next($birth))
echo "$k народився {$birth[$k]}<br>";
```

Такий спосіб є досить незручним і розробники PHP запропонували інший, хоч і менш універсальний, але набагато зручніший метод перебору масивів.

Прямий перебір масиву

На відміну від непрямого перебору (коли спочатку обчислюється черговий ключ, а вже потім по ньому побічно знаходиться значення елементу масиву), прямий перебір лаконічний і набагато простіший. Ідея методу полягає в тому, щоб відразу на кожній ітерації циклу одночасно отримувати і ключ, і значення поточного елементу.

Старий спосіб перебору

От як можна перебрати наш масив за допомогою прямого перебору:

```
for (reset($birth); list($k, $v)=each($birth); /*пусто*/)
echo "$k народився $v<br>";
```

На самому початку заголовка циклу є вже знайома функція reset(). Далі змінним $k і $v привласнюється результат роботи функції each(). Третій параметр циклу просто відсутній.

Функція each():

- по-перше, повертає невеликий масив, нульовий елемент якого зберігає величину ключа поточного елемента масиву $birth, а перший - значення поточного елементу.
- по-друге, вона просуває покажчик поточного елементу до наступної позиції. Слід відмітити, що якщо наступного елементу в масиві немає, то функція повертає не список, а false. Саме тому вона і розміщена в умові циклу for. Третій блок операторів в циклі for просто не потрібний, адже покажчик на поточний елемент і так зміщується функцією each().

Перебір в стилі PHP 4

Прямий перебір масивів застосовувався так часто, що розробники PHP вирішили в четвертій версії мови додати спеціальну інструкцію перебору масиву - foreach. От як з її допомогою можна перебрати і роздрукувати наш масив людей:

```
foreach ($birth as $k=>$v) echo "$k народився $v<br>";
```

Рекомендується скрізь, де не потрібна сумісність з PHP третьої версії, використовувати тільки цей спосіб перебору.

Посилальний синтаксис foreach

Цикл foreach перед початком своєї роботи виконує копіювання масиву. Це дозволяє, наприклад, використовувати замість змінної-масиву результат роботи деякої функції або навіть складний вираз:

```
foreach (array (101, 314, 606) as $magic)
echo "На стіні було написано: $magic. <br>";
```

Робота з копіями в більшості випадків виявляється зручною, проте вона не дозволяє змінювати елементи, що перебираються. Наприклад, наступний цикл foreach (лістинг 6.4), який, здавалося б, збільшує всі елементи масиву на одиницю, насправді не змінює змінну.

Лістинг 6.4.

```
<?php ## Цикл перебирає копію масиву, а не оригінал.
$numbers = array(100, 313, 605);
foreach (Snumbers as $v) $v++;
echo "Елементи масиву: ";
foreach ($numbers as $elt) echo "$elt";
?>
```

Запустивши приклад з лістингу 6.4, можна переконатись, що оператор інкремента ніяк не вплинув на ті дані, що містяться в масиві $numbers числа.

У PHP версії 5 нарешті з'явився різновид циклу foreach, що дозволяє змінювати ітеруючий масив. Виглядає він дуже просто і природно - використовується посилальний оператор &, вказаний перед ім'ям змінної-елементу (лістинг 6.5).

Лістинг 6.5.

```
<?php ## Зміна елементів при переборі.
$numbers = array (100, 313, 605);
foreach ($numbers as &$v) $v++;
echo "Елементи масиву: ";
foreach ($numbers as $elt) echo "$elt ";
?>
```

Тепер можна переконатися, що нова версія програми дійсно змінює масив $numbers, а не працює з його копією.

Списки і рядки

Є декілька функцій, які надзвичайно часто використовуються при програмуванні сценаріїв. Серед них - функції для розбиття якого-небудь рядка на дрібніші частини (наприклад, ці частини розділяються в рядку якимсь специфічним символом типу |), і, навпаки, злиття декількох невеликих рядків в однин великий, причому не впритул, а вставляючи між ними роздільник. Першу з цих можливостей реалізує стандартна функція explode(), а другу - implode().

Функція exploded має наступний синтаксис:

```
list explode (string $token, string $Str [, int $limit])
```

Функція отримує рядок, заданий в її другому аргументі, і намагається знайти в ній підрядки, рівні першому аргументу. Потім по місцю входження цих підрядків рядок "розрізає" на частини, що поміщаються в масив-список, який і повертається. Якщо заданий параметр $limit, то враховуються тільки перші ($limit-1) ділянок "розрізу". Таким чином, повертається список з не більше ніж $limit елементів.

Це дозволяє проігнорувати можливу наявність роздільника в тексті останнього поля, якщо не відомо, що всього полів, скажімо, 6. Ось приклад:

```
$st = "4597219361|Іван Петренко|1992-03-11|Якийсь коментар!";
$person = explode("|", $st, 4); // Відомо, що у рядку тільки 4 поля.
 // Розподіляємо по змінним:
list($id, $name, $burn, $comment) = $person;
```

Звичайно, рядком розбиття може бути не тільки один символ, але і невеликий рядок.

Функція implode() і її синонім join() проводять дію, в точності до навпаки виклику explode().

```
string implode (string $glue, list $list)
```

або

```
string join (string $glue, list $list)
```

Вони беруть асоціативний масив (зазвичай це список) $list, заданий в другому параметрі, і "склеюють" його значення за допомогою "рядка-клею" $glue з першого параметра. Замість списку в другому аргументі можна передавати будь-який асоціативний масив - в цьому випадку розглядатимуться тільки його значення.

Завдання

Завдання 1

0. Написати програму, яка міняє місцями максимальний та мінімальний елементи вектора A[1…N].

1. Написати програму, яка шукає індекс найменшого непарного елемента вектора A[1…N].
2. Написати програму, яка обчислює кількість непарних елементів вектора A[1…N].
3. Написати програму, яка обчислює середнє арифметичне індексів максимального та мінімального елементів вектора A[1…N].
4. Написати програму, яка обчислює середнє арифметичне непарних елементів вектора A[1…N].
5. Написати програму, яка міняє місцями перший елемент із найменшим парним елементом вектора A[1…N].
6. Написати програму, яка шукає максимальний та мінімальний елементи вектора A[1…N].
7. Написати програму, яка міняє місцями елементи вектора A[1…N] так, щоб вони розмістилися в зворотньому порядку: AN, AN-1,…,A2,A1.
8. Написати програму, яка обчислює суму індексів непарних елементів вектора A[1…N].
9. Написати програму, яка обчислює середнє арифметичне елементів вектора A[1…N] з парними індексами.
10. Написати програму, яка міняє місцями останній елемент із найбільшим непарним елементом вектора A[1…N].
11. Написати програму, яка шукає індекси найбільшого та найменшого елементів вектора A[1…N].
12. Написати програму, яка обчислює середнє арифметичне максимального та мінімального елементів вектора A[1…N].
13. Написати програму, яка обчислює суму елементів вектора A[1…N] з непарними індексами.
14. Написати програму, яка обчислює суму максимального та мінімального елементів вектора A[1…N].
15. Написати програму, яка шукає найменший парний елемент вектора A[1…N].

Завдання 2

0. Написати програму, яка шукає суму максимальних елементів по всіх рядках матриці $^{(k\times n)}$. Пояснити призначення індексів.
1. Написати програму, яка транспонує матрицю $^{(n\times n)}$. Пояснити призначення індексів. Нової матриці створювати не можна.
2. Написати програму, яка шукає суму мінімальних елементів по всіх рядках матриці $^{(k\times n)}$. Пояснити призначення індексів.
3. Написати програму, яка шукає суму максимальних елементів по парних рядках матриці $^{(k\times n)}$. Пояснити призначення індексів.
4. Написати програму, яка шукає суму мінімальних елементів по непарних рядках матриці $^{(k\times n)}$. Пояснити призначення індексів.

5. Написати програму, яка шукає найменший з максимальних елементів по всіх рядках матриці $(k \times n)$. Пояснити призначення індексів.
6. Написати програму, яка шукає найбільший з мінімальних елементів по всіх рядках матриці $(k \times n)$. Пояснити призначення індексів.
7. Написати програму, яка шукає найменший з максимальних елементів по парних рядках матриці $(k \times n)$. Пояснити призначення індексів.
8. Написати програму, яка шукає найбільший з мінімальних елементів по непарних рядках матриці $(k \times n)$. Пояснити призначення індексів.
9. Написати програму, яка шукає суму максимальних елементів по непарних стовпцях матриці $(k \times n)$. Пояснити призначення індексів.
10. Написати програму, яка шукає суму мінімальних елементів по парних стовпцях матриці $(k \times n)$. Пояснити призначення індексів.
11. Написати програму, яка шукає найменший з максимальних елементів по непарних стовпцях матриці $(k \times n)$. Пояснити призначення індексів.
12. Написати програму, яка шукає найбільший з мінімальних елементів по парних стовпцях матриці $(k \times n)$. Пояснити призначення індексів.
13. Написати програму, яка міняє місцями максимальний і мінімальний елементи кожного рядка матриці $(k \times n)$. Пояснити призначення індексів.
14. Написати програму, яка міняє місцями максимальний і мінімальний елементи кожного парного рядка матриці $(k \times n)$. Пояснити призначення індексів.
15. Написати програму, яка міняє місцями максимальний і мінімальний елементи кожного непарного стовпця матриці $(k \times n)$. Пояснити призначення індексів.

Завдання 3

0. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Вивести прізвища студентів, які вчаться на відмінно.
1. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Обчислити кількість студентів, які вчаться без трійок (на відмінно і добре).
2. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Вивести прізвища студентів, які вчаться без трійок (на відмінно і добре).
3. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Обчислити кількість студентів, які вчаться на відмінно.
4. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Обчислити процент студентів, які вчаться на відмінно.

5. Сформувати масив записів про: прізвище студента. оцінки з фізики, математики, інформатики. Обчислити процент студентів, які вчаться без трійок (на відмінно і добре).
6. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Для кожного студента вивести: прізвище і середній бал.
7. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Для кожного предмета обчислити середній бал.
8. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Обчислити кількість оцінок ”відмінно” по кожному предмету.
9. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Обчислити кількість оцінок ”добре” по кожному предмету.
10. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Обчислити кількість оцінок ”задовільно” по кожному предмету.
11. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Обчислити кількість кожної з оцінок ”5”, “4”, “3” по фізиці.
12. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Обчислити кількість кожної з оцінок ”5”, “4”, “3” по математиці.
13. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Обчислити кількість кожної з оцінок ”5”, “4”, “3” по інформатиці.
14. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Обчислити найбільший середній бал (порівнюючи середні бали для кожного студента).
15. Сформувати масив записів про: прізвище студента, оцінки з фізики, математики, інформатики. Обчислити кількість студентів, середній бал яких вищий 4,5.

Хід роботи

1. Створити новий файл у текстовому редакторі (Dreamweaver, Notepad або іншому).
2. Написати розв’язок завдань.
3. Зберегти програму на диск під новим ім`ям 1.php у папку <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB06
4. Перевірити виконання скрипта у браузері.
5. При виникненні помилки – відкоригувати програму та виконати її.
6. Оформити звіт по виконаній роботі.

Методичні рекомендації

Для введення даних використовуються параметри в URL, для виконання програми потрібно у браузері ввести <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB01\1.php?n=<зн1>.

Розв’якок завдання (текст програми):

```
<?php
$n = $_GET['n'];
/* Завдання 1 */
$a = array();
echo "Завдання 1:<br/>";
echo "Згенерваний масив:";
for ($i=1;$i<=$n;$i++) {
  $a[$i] = rand();
  echo $a[$i]." ";
}
$mini = 1;
$maxi = 1;
$min = $a[1];
$max = $a[1];
for ($i=2;$i<=$n;$i++) {
  if ($a[$i] < $min) { $min = $a[$i]; $mini = $i; }
  if ($a[$i] > $max) { $max = $a[$i]; $maxi = $i; }
}
$k = $a[$mini];
$a[$mini] = $a[$maxi];
$a[$maxi] = $k;
echo "Змінений масив:";
for ($i=1;$i<=$n;$i++) {
  echo $a[$i]." ";
}

/* Завдання 2 */
$a = array();
echo "Завдання 2:<br/>";
echo "Згенерваний масив:";
```

```
for ($i=1;$i<=$n;$i++) {
  echo "<br/>";
  for ($j=1;$j<=$n;$j++) {
    $a[$i][$j] = rand();
    echo $a[$i][$j]." ";
  }
}
echo "<br/>";
$s = 0;
for ($i=1;$i<=$n;$i++) {
  $s += max($a[$i]);
}
echo "Сума максимальних елементів по всіх рядках: $s";
/* Завдання 3 */
$a = array();
$a[] = array('surname' => 'Іванов', 'fiz' => rand(3,5) , 'mat' =>
rand(3,5) , 'inf' => rand(3,5));
$a[] = array('surname' => 'Петров', 'fiz' => rand(3,5) , 'mat' =>
rand(3,5) , 'inf' => rand(3,5));
$a[] = array('surname' => 'Сидоров', 'fiz' => rand(3,5) , 'mat'
=> rand(3,5) , 'inf' => rand(3,5));
// ...
echo "Завдання 2:<br/>";
echo "Згенерваний масив:";
echo "<table>\n ";
echo
"<tr><th>Прізвище</th><th>Фізика</th><th>Математика</th><th>Інфор
матика</th></tr>\n";
foreach($a as $stud){
echo
"<tr><td>{$stud['surname']}</td><td>{$stud['fiz']}</td><td>{$stud
['mat']}</td><td>{$stud['inf']}</td></tr>\n ";
}
echo "</table>\n";
echo "Відмінники: ";
foreach($a as $stud){
if (($stud['fiz'] == 5) && ($stud['mat'] == 5) && ($stud['inf']
== 5)) echo $stud['surname'];
}
```

?>

Контрольні запитання

1. Як сформувати одновимірний масив?
2. Як сформувати двовимірний масив?
3. Як пройти по усіх значеннях масиву?
4. Що таке асоціативний масив?
5. Як очистити масив?

Тема 7: Регулярні вирази

Мета роботи: навчитись застосовувати регулярні вирази для перевірки співпадіння та заміни співпадінь

Теоретичні відомості

Регулярні вирази - це один з інструментів, що часто використовується у Веб-програмуванні, і незнання їх основ може сильно ускладнити подальшу творчість у Веб.

Нехай маємо вираз і рядок. Операцію перевірки, чи задовольняє рядок цьому виразу (або вираз - рядку), надалі називатимемо зіставленням рядка і виразу. Якщо якась частина рядка успішно зіставилася з виразом, назвемо це збігом. Наприклад, збігом від зіставлення виразу "група букв, оточена пробілами" до рядку "Ab cde fgh" буде рядок "cde" (адже тільки він задовольняє нашому виразу).

Мови регулярних виразів

Існує декілька різних мов регулярних виразів, на які можна переводити наші "словесні твердження". Найбільш поширені - це мова PCRE (Perl Compatible Regular Expression, регулярний вираз мови Perl), а також вирази POSIX (Portable Operating System Interface, переносимий інтерфейс операційної системи), їх ще іноді називають RegEx або RegExp. У PHP є функції для роботи з обома форматами.

Потрібно враховувати, що формат POSIX на практиці поволі, але вірно застаріває, а мова PCRE обходить його практично по всіх параметрах (і по кількості можливостей, і за швидкістю на більшості виразів). Але оскільки механізми роботи аналізатора PCRE і POSIX сильно розрізняються (перший використовує апарат недетермінованих кінцевих автоматів (НКА), другий же, - детермінованих (ДКА)), неможливо сказати абсолютно для всіх випадків, який з них працює швидше. Існують вирази, що записуються однаково на мовах PCRE і

POSIX, проте розрізняються за швидкістю роботи в десятки і сотні разів - як на користь PCRE, так і на користь POSIX. Проте в більшості типових випадків виразу PCRE показують значно кращі результати, чим POSIX. Тому далі розглядатимемо тільки формат PCRE.

Мова PCRE

Кожен вираз мови складається з однієї або декількох команд, що управляють. Деякі з них можна групувати (як групуються інструкції в програмі за допомогою фігурних дужок), і тоді вони вважаються за одну команду. Всі команди, що управляють, розбиваються на три класи:

- прості символи, а також символи, що управляють, грають роль їх "замінників", - їх ще називають літералами;
- конструкції, що управляють (квантифікатори повторень, оператор альтернативи, дужки, що групують і т. д.);
- так звані уявні символи (у рядку їх немає, але вони як би позначають якусь частину рядка - наприклад, його кінець).

До символів, що управляють, відносяться наступні: ., *, +, ?, |, (,), [,],{, }, $,^. Всі символи, окрім цих, позначають в регулярному виразі самі себе і не мають яких-небудь спеціальних призначень.

Обмежувачі

Будь-який регулярний вираз в PHP представлений у вигляді звичайного рядка символів. Крім того, мова PCRE вимагає, щоб всі команди усередині виразу були поміщені між обмежувачами - двома слешами, що також є частиною рядка. Наприклад, рядок "expr" є синтаксично некоректним регулярним виразом, тоді як "/expr/" - правильний.

Після останнього слеша можна вказувати різні модифікатори, такий як /i: він говорить PHP, що враховувати регістр символів при пошуку збігів не слід.

Наприклад, вираз /expr/i співпадає як з рядком "expr", так і з рядками "eXpr" і "Expr".

Отже, загальний вид запису регулярного виразу – '/вираз/M', де M позначає нуль або більше модифікаторів. Якщо символ / зустрічається в самому виразі, перед ним необхідно поставити зворотний слеш \, щоб його екранувати:

```
if (preg_match('/path\/to\/file/i', "path/to/file")) echo
"збіг!";
```

Альтернативні обмежувачі

Синтаксис запису рядків в PHP вимагає, щоб всі зворотні слеші в програмі були подвоєні. Тому отримуємо вельми непоказну конструкцію - '/path\\/to\\/file/i'. Проблема в тому, що символи-обмежувачі співпадають з символами, які ми шукаємо.

Спеціально для того, щоб спростити запис, мова PCRE підтримує використання альтернативних обмежувачів. У їх ролі може виступати буквально все, що завгодно. Наприклад, наступні регулярні вирази означають одне і те ж:

```
// Можна використовувати будь-які однакові символи як
обмежувачі...
'/path\\/to\//file/i'
'#path/to/file#i'
'"path/to/file"i'
//А можна - парні дужки.
'{path/to/file}i'
'[path/to/file]i'
'(path/to/file]i'
```

Останні три приклади особливо цікаві: якщо як початковий обмежувач виступає дужка, то фінальний символ повинен бути рівний парній з нею дужці.

Користь від парних дужок величезна. Річ у тому, що при їх використанні дужки, що зустрічаються усередині виразів, вже не потрібно екранувати зворотним слешем: аналізатор PCRE самостійно "рахує" вкладеність дужок і не генерує помилки. Наприклад, наступний вираз коректний:

```
echo preg_replace('[(/file)[0-9]+]i', '$1', "/file123.txt");
```

Хоча квадратні дужки в регулярних виразах - це спецсимволи, що позначають "альтернативу символів", не доводиться ставити перед ними зворотний слеш, не дивлячись на те, що вони співпадають з обмежувачами.

Відміна дії спецсимволів

Спочатку досить легко заплутатися у всіх цих слешах, апострофах, доларах... Складність полягає в тому, що такі символи є спеціальними як в PCRE, так і в PHP, а тому іноді їх потрібно екранувати не один раз, а декілька.

Розглянемо, наприклад, регулярний вираз, який шукає в рядку деяке ім'я файлу, що передує зворотним слешем \ (як в Windows). Він записується так:

```
$re = '/\\\\filename/';
```

Подвоєний слеш в рядках PHP позначає один слеш. Якщо вставити в програму оператор echo $re, то буде надрукований текст /\\filename/.

Подвоєний слеш в PCRE означає один слеш. Таким чином, отримавши на вхід вираз /\\filename/, аналізатор зрозуміє, що від нього потрібно знайти підрядок, що починається з одного слеша.

Розглянемо складніший приклад: шукатимемо будь-яке ім'я каталогу, після якого йде також будь-яке ім'я файлу. Вираз записуватиметься так:

```
$re = '/\\S+\\\\\\S+/';
```

Послідовність \S у PCRE позначає будь-який "непробільний" символ, а + після команди - повтор її один або більше разів.

"Розмноження" слешів називають в літературі по регулярних виразах "синдромом зубочистки". Зазвичай цей "синдром" характерний для мов, в яких регулярні вирази представлені у вигляді звичайних рядків.

Якщо якийсь регулярний вираз з "зубочистками" навідріз відмовляється працювати, варто спробувати записати його змінну, а після цього вивести її у браузер:

```
echo "<tt>".htmlspecialchars($re)."</tt>";
```

Цей прийом допоможе побачити вираз "очима PCRE", вже після того, як PHP отримає рядкові дані виразу.

Якщо ж потрібно вставити у вираз один з символів, що управляють, але тільки так, щоб він "не діяв", досить передувати його зворотним слешем. Наприклад, якщо ми шукаємо рядок, що містить підрядок a*b, то потрібно використовувати для цього вираз a*b, оскільки символ * є таким, що управляє.

Прості символи (літерали)

Клас простих символів, дійсно, найпростіший. А саме будь-який символ в рядку на PCRE (і, до речі, на мові POSIX теж) позначає сам себе, якщо він не є таким, що управляє. Наприклад, регулярний вираз at "реагуватиме" на рядки, в яких зустрічається послідовність at, - чи в середині слова, чи на початку - не важливо:

```
echo preg_replace('/at/', 'AT', "What is the Perl Compatible
Regex?");
```

В результаті буде виведений рядок:

```
WhAT is the Perl CompATible Regex?
```

Якщо в рядок необхідно вставити символ (наприклад *), що управляє, потрібно поставити перед ним \, щоб відмінити його спеціальну дію. Тобто не можна (вірніше, не рекомендується) написати:

```
$re = "/a\*b/"
```

але можна:

```
$re = "/a\\*b/"
```

У останньому випадку в рядку $re виявляється /a*b/.

Класи символів

Існують декілька спецсимволів, що позначають відразу клас букв. Найважливіший з таких знаків - крапка (".") - позначає один будь-який символ. Наприклад, вираз /a.b/s має збіг для рядків azb або aqb, але не "спрацьовує", скажімо, для aqwb або ab.

У PCRE (на відміну від POSIX) існує ще декілька класів:

- \s - відповідає "пробільному" символу: пропуску (" "), знаку табуляції (\t), перенесенню рядка (\n) або поверненню каретки (\r);
- \S - будь-який символ, окрім "пробільного";
- \w - будь-яка буква або цифра;
- \W - не буква і не цифра;
- \d - цифра від 0 до 9;
- \D - все, що завгодно, але тільки не цифра.

Альтернативи

Для пошуку не довільного символу, а одного з декількох вказаних, потрібно необхідні символи вкласти у квадратні дужки. Наприклад, вираз /a[xXyY]c/ відповідає рядкам, в яких є підрядки з трьох символів, що починаються з a, потім одна з букв x, X, y, Y і, нарешті, буква c. Якщо потрібно вставити всередину квадратних дужок символ [або], то слід просто поставити перед ним зворотний слеш, щоб відмінити його спеціальну дію.

Якщо букв-альтернативи багато, і вони йдуть підряд, то не обов'язково перераховувати їх усі усередині квадратних дужок - досить вказати першу з них, потім поставити дефіс і потім - останню. Такі групи можуть повторюватися. Наприклад, вираз /[a-z]/ позначає будь-яку букву від a до z включно, а вираз /[a-zA-z0-g] / задає будь-який алфавітно-цифровий символ.

Негативні класи

Коли альтернативних символів багато, досить утомливо перераховувати їх всіх в квадратних дужках. Особливо, якщо влаштовує будь-який символ окрім декількох (наприклад, окрім > і <). В цьому випадку, звичайно, не варто вказувати 254 символи, натомість краще скористатися конструкцією [^<>], яка позначає будь-який символ, окрім тих, які перераховані після [^ і до]. Наприклад,

вираз /<[^>]+>/ "спрацьовує" на всі HTML-теги в рядку, тому простий спосіб видалення тегів виглядає так:

```
echo preg_replace('/<[^>]+>/', '', $text);
```

У негативному класі можуть бути задіяні будь-які символи і вирази, які допустимі в конструкції [...].

Квантифікатори повторень

Квантифікатори - спеціальні знаки, що використовуються для уточнення дії символів першого класу, що стоять перед ними.

Нуль або більше повторень

Найбільш популярний квантифікатор - зірочка (*). Вона означає, що попередній символ може бути повторений нуль або більше разів (тобто можливо, і жодного разу). Наприклад, вираз /a-*-/ відповідає рядку, в якому є буква a, потім - нуль або більше мінусів і, ще один мінус.

У простому випадку при цьому робиться спроба знайти якомога довший рядок, тобто зірочка "поглинає" так багато символів, як це можливо. Наприклад, для рядка a--b знайдеться підрядок a-- (зірочка "заковтнула" 2 мінуси), а не a- (зірочка захопила 1 мінус). Це - так звана "жадність" квантифікатора.

Одне або більше повторень

Попередній приклад досить незграбний. Насправді, фактично складений вираз, який шукає рядки з a і одним або більше мінусом. Можна було б записати його і так: /a--*/, але краще скористатися спеціальним квантифікатором, який якраз і позначає "один або більше збігів", - символом плюса (+). З його допомогою можна було б вираз записати лаконічніше: /a-+/, що буквально і читається як "a і один або більше мінусів". Ось приклад виразу, який визначає, чи є в рядку англійське слово, написане через дефіс: /[a-zA-z]+-[a-zA-z]+/.

Нуль або одне повторення

І вже щоб зовсім полегшити життя, іноді використовують ще один квантифікатор - знак питання (?). Він означає, що попередній символ може бути повторений нуль або один раз. Наприклад, вираз /[a-zA_z]+\r?\n/ визначає рядки, в яких останнє слово притиснуте до правого краю рядка. Якщо ми працюємо в Unix, то в кінці рядка символ \r зазвичай відсутній, тоді як в текстових файлах Windows кожен рядок закінчується парою \r\n. Для того, щоб сценарій правильно працював в обох системах, ми повинні врахувати цю особливість - можливу наявність \r перед кінцем рядка.

Задане число збігів

Є ще один спосіб задати число збігів, з його допомогою можна реалізувати всі перераховані вище способи, правда, і виглядає він декілька громіздкіше. Квантифікатор "фігурні дужки". Існує декілька форматів його запису:

- x{n,m} - указує, що символ x може бути повторений від n до m разів;
- x{n} - указує, що символ x повинен бути повторений рівно n разів;
- x{n, } - указує, що символ x може бути повторений n або більше разів.

Значення n і m в цих прикладах обов'язково повинні належати діапазону від 0 до 65 535 включно.

Уявні символи

Уявні символи - це просто ділянка рядка між сусідніми символами, що задовольняє деяким властивостям. Фактично уявний символ - це якась позиція в рядку. Наприклад:

- ^ – відповідає початку рядка (не першому символу рядка, а в точності початку рядка, позиції перед першим символом);
- $ – відповідає кінцю рядка (знову ж таки, позиції за кінцем рядка);
- \b – відповідає початку або кінцю слова. Фактично будь-яка позиція між \w\W або \W\w примушує \b "спрацювати";

- \B - будь-яка позиція, окрім початку або кінця слова.

Приклади:

'/^\w:/' – відповідає будь-якому рядку, що починається з букви, після якої йде двокрапка; абсолютний шлях в Windows виглядають саме таким чином;

'/\.txt$/i' - відповідає рядку, що зберігає ім'я файлу з розширенням txt;

'/^$/' - збігає тільки з порожнім рядком, тому що говорить: "відразу після початку рядка йде його кінець".

Оператор альтернативи

Фактично конструкція […] – це не що інше, як оператор альтернативи, що працює тільки з окремими символами (і тому досить швидко).

Але в регулярних виразах є можливість задавати альтернативи не одиночних символів, а відразу їх груп. Це робиться за допомогою оператора |.

Ось декілька прикладів його роботи.

Вираз '/1|2|3/' повністю еквівалентно виразу '/[123]/', але зіставлення відбувається повільніше.

Виразу /\.gif$|\.jpe?g$/ відповідають імена файлів у форматі GIF або JPEG.

Виразу '#^\w:|^\\\\|^/#' відповідають тільки абсолютні файлові шляхи. Дійсно, його можна прочитати так: "на початку рядка йде або буква диска, або ж прямий або зворотний слеш".

Групуючі дужки

Останній приклад наводить на міркування про те, чи не можна як небудь згрупувати окремі символи, щоб не писати по кілька разів одне і те ж. У цьому прикладі уявний символ ^ зустрічається у виразі аж 3 рази. Але не можна написати вираз так: ‘#^\w:|\\\\|/#’, тому що оператор |, природно, намагається застосувати себе до якомога довшої послідовності команд.

Саме для мети управління оператором альтернативи (але не тільки) і служать групуючі круглі дужки (...). Неважко здогадатися по сенсу, що вираз з останнього прикладу можна записати з їх допомогою так: '#^ (\w:|\\\\|/) #'.

Звичайно, дужки можуть мати довільний рівень вкладеності. Це буває корисно для складних виразів, які містять багато умов, а також для ще одного застосування круглих дужок – "Кишені".

"Кишені"

На практиці часто буває потрібно не просто дізнатися, де в рядку є збіг, але також і розбити цей збіг на частини, ради яких, власне, і велася вся робота.

Ось приклад, що прояснює ситуацію. Хай в рядку задана дата у форматі dd-mm-yyyy, і в ній можуть знаходитися паразитні пропуски на початку і кінці, а замість дефісів випадково зустрічаються взагалі будь-які знаки пунктуації, "пересипані" пропусками (слеші, крапки, символи підкреслення). Нас цікавить, що ж все-таки за дату нам передали. Тобто, точно відомо, що цей рядок - саме дата, але де в ній день, де місяць і де рік?

Спершу встановимо, що всі правильні дати повинні відповідати виразу:

```
|^\s*((\d+)\s*[[:punct:]]\s*(\d+)\s*[[:punct:]]\s*(\d+))\s*$|xs
```

Тут, модифікатор /x, примушує аналізатор PCRE ігнорувати всі пробільні символи у виразі (за винятком явно вказаних як \s або усередині квадратних дужок) - це дозволяє записати вираз красивіше.

Будь-який блок, обрамлений у виразі дужками, виділяється як єдине ціле і записується в так звану "кишеню" (номер кишені відповідає порядковому номеру відкриваючої дужки). У нашому випадку:

- в першу кишеню запишеться дата, але вже без початкових і кінцевих пропусків (це забезпечує сама зовнішня пара дужок);
- в другу кишеню запишеться день;
- в третю – місяць;

- в четверту – рік.

Як вже згадувалося, в нульову кишеню у будь-якому випадку записується весь знайдений збіг. У даному прикладі це буде весь рядок разом з можливими початковими і кінцевими пропусками.

Для отримання вмісту кишень призначений третій параметр функції preg_match(). Виходячи з цього, маємо програму на PHP, що виконує необхідні дії, представлену в лістингу 7.1.

Лістинг 7.1.

```
<?php ## Простий розбір дати.
$str = " 15-16/2000  "; // наприклад
$re = '{
  ^\s*(                    # початок рядка
    (\d+)                  # день
      \s* [[:punct:]] \s*  # роздільник
    (\d+)                  # місяць
      \s* [[:punct:]] \s*  # роздільник
    (\d+)                  # рік
  )\s*$                  # кінець рядка
}xs';
// Розбиваємо рядок на шматки за допомогою preg_match().
preg_match($re, $str, $pockets) or die("Не дата: $str");
// Тепер розбираємося з кишенями.
echo "Дата без пропусків: '$pockets[1]' <br>";
echo "День: $pockets[2] <br>";
echo "Місяць: $pockets[3] <br>";
echo "Рік: $pockets[4] <br>";
?>
```

Всупереч правилам, у цьому прикладі не подвоюються слеші в конструкціях \s і \d. Взагалі, їх слід було б записати як \\s і \\d, але PHP проявляє диво кмітливості: якщо після слеша стоїть буква, що не входить ні в один рядковий метасимвол, він залишає все, як є.

Використання кишень у функції заміни

Нехай потрібно всі слова в рядку, що починаються з "долара" ($), зробити "жирними", - обрамити тегами і , - для подальшого виводу у браузер. Це може знадобитися, якщо потрібно текст деякої програми на PHP вивести так, щоб в нім виділялися імена змінних. Очевидно, вираз для виявлення імені змінної в рядку буде таким: /\$[a-z]\w*/i.

У лістингу 7.2 приведена програма, яка "розфарбовує" свій власний код.

Лістинг 7.2.

```
<?php ## Заміна по шаблону.
$text = htmlspecialchars(file_get_contents(__FILE__));
$html = preg_replace('/(\$[a-z]\w*)/is', '<b>$1</b>', $text);
echo "<pre>$html</pre>";
?>
```

Під час заміни скрізь замість поєднання $1 підставляється вміст кишені з номером 1.

Також, замість $1 можна також використовувати \1, або, при записі у вигляді рядка "\\1".

Використання кишень у функції зіставлення

Вміст кишень можна задіяти і у функції preg_match() - раніше, ніж закінчиться зіставлення. А саме, управляти ходом пошуку на основі даних в кишенях. Така дія називається зворотним посиланням.

Розглянемо такий приклад. Відомо, що в рядку є підрядок, обрамлений якимось HTML-тегами (наприклад, або <pre>), але невідомо, якими. Потрібно помістити цей підрядок в кишеню, щоб надалі з нею працювати. Зрозуміло, закриваючий тег повинен відповідати що відкриває, наприклад, до тегу парний - , а до <pre> - </pre>.

Завдання вирішується за допомогою такого регулярного виразу:

|<(\w+)[^>]*>(.*?)</\1>|xs

Конструкція .*? примушує зірочку "стримати апетит" і співпасти не з максимальним, а з мінімальним можливим числом символів.

Внутрішній текст опиниться в другій кишені, а ім'я тега - в першому. От як це працює: PHP намагається знайти відкриваючий тег, і, як тільки знаходить, записує його ім'я в першу кишеню (оскільки це ім'я обрамлене у виразі першою парою дужок). Далі він дивиться вперед і, як тільки натрапляє на </, визначає, чи слідує за ним те саме ім'я тега, яке у нього лежить в першій кишені. Це дійсно примушує його зробити конструкція \1, яка заміщається на вміст першої кишені кожного разу, коли до неї доходить черга. Якщо ім'я не співпадає, то такий варіант PHP відкидає і "йде" далі, а інакше сигналізує про збіг.

У лістингу 7.3 приведена програма, яка знаходить в рядку слово, обрамлене будь-якими парними тегами.

Лістинг 7.3.

```
<?php ## Зворотні посилання.
$str = "Hello, this <b>word</b> is bold!";
$re = '|<(\w+)[^>]*>(.*?)</\1>|xs';
preg_match($re, $str, $pockets) or die("Немає тегів.");
echo htmlspecialchars("'$pockets[2]' обрамлено тегом
'$pockets[1]'");
?>
```

"Жадібність" квантифікаторів

Замінимо у попередньому прикладі символи **.*?** на **.*** . Давайте спробуємо (лістинг 7.4).

Лістинг 7.4.

```
<?php
$str = "Hello, this <b>word</b> is <b>bold</b>!";
$re = '|<(\w+)[^>]*>(.*)</\1>|xs';
preg_match($re, $str, $pockets) or die("Немає тегів.");
echo htmlspecialchars("'$pockets[2]' обрамлено тегом
'$pockets[1]'");
?>
```

В результаті отримаємо наступний текст:

'word is bold' обрамлене тегом ‘b’

Вираз .* захопив максимально можливе число символів, а тому кінець виразу співпав не з парним тегом , а з найостаннішим в рядку.

Поставивши знак питання після будь-якого з квантіфікаторов *, +, {} або навіть ?, ми даємо йому "пігулки від жадності". Як говорять в літературі по регулярних виразах, ми робимо квантифікатор "ледачим".

Зазвичай "ледачі" квантифікатори застосовують для пошуку конструкцій, що претендують на роль парних. Наприклад, наступний код видаляє всі теги з деякого рядка:

```
echo preg_replace('/<.+?>/s', '', $str);
// Виглядає явно витонченіше, ніж /<[^>]+>/s.
```

Код для заміни "псевдотегів" [b]...[/b] на їх HTML-еквіваленти міг би виглядати так:

```
echo preg_replace('|\[b\](.*?)\[/b\]|ixs', '<b>$1/b>', $str);
```

В цій ситуації без "ледачої" версії зірочки вже не обійтися ніяк (на відміну від попереднього прикладу).

На жаль, "ледачі" квантифікатори - теж не панацея. Вони не роблять ніяких припущень щодо вкладеності конструкцій. Спробуємо застосувати останній приклад з "псевдотегами" до наступного рядка:

```
'[b]жирний  текст [b]а тут - ще жирніше[/b] повернулися[/b]'
```

Давайте подивимося, як різні вирази співпадуть з цим рядком (лістинг 7.5).

Лістинг 7.5.

```
<?php ## Порівняння "жадібних" і "ледачих" квантифікаторов.
$str = '[b]жирний текст [b]а тут - ще жирніше[/b]
повернулися[/b]';
$to  = '<b>$1</b>';
$re1 = '|\[b\](.*)\[/b\]|ixs';
$re2 = '|\[b\](.*?)\[/b\]|ixs';
$result = preg_replace($re1, $to, $str);
```

```
echo "Жадібна версія: ".htmlspecialchars($result)."<br>";
$result = preg_replace($re2, $to, $str);
echo "Ледача версія: ".htmlspecialchars($result)."<br>";
?>
```

Побачимо наступний результат:

```
Жадібна версія: <b>жирний текст [b]а тут - ще жирніше[/b]
вернулись</b>
Ледача версія: <b>жирный текст [b]а тут - ще жирніше</b>
повернулися[/b]
```

Як видно, ні "жадібна", ні "ледача" зірочка не змогли справитися з роботою: перша пропустила внутрішні "псевдотеги", а друга - виконала непарну заміну.

Групування без захоплення

Кожна ділянка рядка, що відповідає шматочку виразу, обрамленому круглими дужками, у результаті потрапляє в ту або іншу кишеню. В більшості випадків так і потрібно, проте трапляються ситуації, коли дужки використовуються тільки для опису конструкцій, що повторюються, і немає ніякої необхідності записувати щось в кишені.

До таких ситуацій, наприклад, практично завжди відноситься конструкція

```
(підвираз)*
```

Тут дужки використовуються тільки для того, щоб "застосувати" зірочку не до одного останнього символу, а до цілого підвиразу. У кишеню в цьому випадку потрапить лише останній збіг. Для того, щоб записати в кишеню текст, що співпав, доведеться написати:

((підвираз)*)

Але ж тоді вийде вже дві кишені, а не одна...

Рішення - застосування "незберігаючих" дужок. З їх використанням останній приклад може бути переписаний так:

```
((?:підвираз)*)
```

Ось тепер створиться одна-єдина кишеня, в яку і буде поміщено весь збіг, "накручений" зірочкою.

Модифікатори

Будь-який регулярний вираз у форматі PCRE повинен бути обрамлений символами-обмежувачами. Після останнього обмежувача можуть йти декілька так званих модифікаторів, призначених для уточнення дії регулярного виразу.

Модифікатор /i - ігнорування регістра

Вираз /вираз/i співпадає безвідносно до регістра символів в цільовому рядку. При цьому автоматично враховують настройки локалі. А значить, при вірно вказаному імені локалі спецсимволи \w, \b, конструкція [[:alpha:]] і т.д. коректно працюють з кириличними буквами. Наприклад, вираз [[:alpha:]]+ задовольняє будь-якому слову як на англійській, так і на українській мові.

При зіставленні регулярного виразу з рядком в кишені потрапляє ділянка рядка, що співпала, незалежно від того, чи вказаний модифікатор /i чи ні. Наприклад, при пошуку по виразу /(a+)/i в рядку "BAaB" в першій кишені опиниться "Aa", а не "aa".

Модифікатор /x - пропуск пропусків і коментарів

Він дозволяє писати регулярні вирази у витонченішому, читабельному вигляді. Допускається вставляти у вираз пробільні символи (зокрема перехід рядка), а також однорядкові коментарі, перед якими йде решітка (#).

Приклад регулярного виразу з модифікатором /x:

```
$re = '{
\[(\w+)\]  # відкриваючий тег
(.*?)      # мінімум будь-яких символів
\[/\1\]    # і закриваючий тег, парний до відкриваючого
}ixs';
```

Модифікатор /m - багаторядковість

У змінних можуть міститися "багаторядкові" рядки - величини, що містять символи переходу рядка. У попередніх прикладах малось на увазі, що регулярний вираз зіставляється з рядком цілком, і символ ^ співпадає з початком рядка, а

символ $ - з його кінцем. Така поведінка не завжди виявляється зручною і, більш того, вона навіть не завжди діє "по замовчуванні".

Нехай потрібно додати знак табуляції в кожний рядок деякої "багаторядкової" змінної. Лістинг 7.6 ілюструє, як це зробити.

Лістинг 7.6.

```
<?php
$str = file_get_contents(__FILE__);
$str = preg_replace('/^/m', "\t", $str);
echo "<pre>".htmlspecialchars($str)."</pre>";
?>
```

Регулярний вираз '/^/m' тепер співпадає з початком кожного внутрішнього рядка змінної $str. Функція замінює "початок рядка" на символ табуляції - фактично, замінюємо 0 символів на 1.

Як видно, модифікатор /m примушує деякі символи, що управляють, поводитися по-іншому. Список ролей, що змінилися.

- Уявний символ ^ тепер відповідає початку кожного внутрішнього підрядка в змінній, що зіставляється.
- Уявний символ $ співпадає з позицією перед символом \n, а також з кінцем рядка.
- Крапка "." як і раніше співпадає з будь-яким символом, але тепер - за виключенням \n.
- Уявний символ \A співпадає з "початком даних", тобто з позицією перед першим символом рядкової змінної. Без модифікатора /m уявний символ ^ поводиться точно так, як і \A.
- Уявний символ \z співпадає з "кінцем даних" - позицією після останнього символу рядкової змінної.

Саме режим /m діє за умовчанням, коли пряцюється із функцією preg_match() для пошуку збігів у рядку.

Модифікатор /s - однорядковий пошук

Модифікатор /s форсує "однорядковий" варіант пошуку. Якщо модифікатор /m не вказаний явно у функції заміни, то мається на увазі саме /s. У функції пошуку, навпаки, за умовчанням діє /m і потрібно вручну указувати /s всякий раз, коли потрібно працювати з рядком, як з єдиним цілим.

Модифікатор /e - виконання PHP-програми при заміні

Він працює тільки у функції заміни preg_replace() і примушує PHP трактувати другий параметр, як код на PHP, результат роботи якого підставляється замість знайденої ділянки.

Наприклад, потрібно перевести у верхній регістр всі HTML-теги в деякій змінній. Це робить наступний оператор:

```
$str = preg_replace (
'{(</?)(\w+)(.*?>)}es', # знаходимо відкриваючий або закриваючий
тег
"'$1'.strtoupper('$2').'$3", # переводить у верхній регістр
$str
);
```

На жаль, модифікатор /e за своєю природою дуже недосконалий: він виконує код вже після підстановки значень $1, $2 і т.д. В результаті, якщо, наприклад, всередині тега зустрінеться апостроф, то вийде некоректний код: адже '$3' перетвориться на підрядок, що містить зайвий апостроф. На жаль, боротися з цим, використовуючи одну лише функцію preg_replace(), неможливо. Отже, даного модифікатора краще уникати, використовуючи замість нього виклик preg_replace_callback().

Функції PHP для роботи із регулярними виразами

Пошук збігів

Всі функції пошуку по регулярному виразу за умовчанням працюють в багаторядковому режимі, нібито вказаний модифікатор /m. Рекомендується явно використовувати /s, коли це необхідно.

```
bool preg_match(string $expr, string $str [,list &$pockets] [,int
$flags] [,int $offset])
```

Як видно з прототипу, функція preg_match() має куди багатші можливості, ніж ті, що використовувались вище. Перші три аргументи вже знайомі: це регулярний вираз, рядок, в якому проводиться пошук, а також необов'язкова змінна, в яку будуть записані всі збіги виразів у круглих дужках $exprg. Функція повертає 1 у разі виявлення збігу і 0- інакше. Якщо регулярний вираз містить помилки (наприклад, непарні дужки), то згенерує відповідне попередження.

Необов'язковий параметр $offset може указувати позицію, з якою потрібно починати проглядання рядка.

Параметр $flags на справжній момент може приймати тільки одне значення - PREG_OFFSET_CAPTURE. Він примушує PHP трохи змінити формат списку $pockets, тепер разом з текстом, що співпав, зберігається також і позиція збігу в результатному рядку. Лістинг 7.7 ілюструє сказане.

Лістинг 7.7

```
<?php ## Використання PREG_OFFSET_CAPTURE.
$st = '<b>жирний текст</b>';
$re = '|<(\w+).*?>(.*?)</\1>|s';
preg_match($re, $st, $p, PREG_OFFSET_CAPTURE);
echo "<pre>"; print_r($p); echo "</pre>";
?>
```

Результат роботи цієї програми виглядає приблизно так:

```
Array (
[0] => Array([0] => <b>жирний текст</b> [1] => 0 )
[1] => Array([0] => b [1] => 1 )
[2] => Array([0] => жирний текст [1] => 3)
)
```

Масив $pockets як і раніше містить декілька елементів, проте, якщо раніше це були звичайні рядки, то з використанням PREG_OFFSET_CAPTURE – списки з двох елементів: array (підрядок, зсув).

```
int preg_match_all (string $expr, string $str, list &$pockets,
[,int $flags] [,int $offset])
```

Якщо функція preg_match() шукає тільки перший збіг виразу в рядку, то preg_match_all() шукає всі збіги. Сенс аргументів майже той же. Функція повертає число знайдених підрядків (або 0, якщо нічого не знайдено).

Формат результату, який опиниться в $pockets (цього разу аргумент вже обов'язковий), істотно залежить від параметра $flag, що приймає ціле значення (зазвичай використовують константу). Проте у будь-якому випадку в $pockets опиниться двовимірний масив.

Можливі константи для параметра $flag:

PREG_PATTERN_ORDER

Список $pockets містить елементи, впорядковані по номеру відкриваючої дужки. Іншими словами, до масиву потрібно звертатися так: $pockets[B][N], де B - порядковий номер відкриваючої дужки у виразі, а N - номер збігу, якщо їх було декілька. Наприклад, в $pockets[0] міститиметься список підрядків, що повністю співпали з виразом $expr у рядку $str, у $pockets[1] - список збігів, яким відповідає перша відкриваюча дужка (якщо вона є), і т.д. даний режим є за умовчанням.

PREG_SET_ORDER

Список $pockets виявляється відсортованим по номеру збігу. Іншими словами, скільки разів вираз $expr співпало в рядку $str, стільки елементів і опиниться в $pockets. При цьому кожен елемент матиме таку саму структуру, як і при виклику звичайної функції preg_match() - а саме, це список виразів у дужках, що співпали (нульовий елемент - весь вираз, перший - перша дужка і т. д.). Звернення до масиву організовується так: $pockets[N][B], де N - порядковий номер збігу, а B - номер дужки.

PREG_OFFSET_CAPTURE

Це не окреме значення прапора, а просто величина, яку можна додати до PREG_PATTERN_ORDER або PREG_SET_ORDER. Вона примушує PHP повертати цифрові зсуви знайдених елементів разом з їх значеннями - точно так, як і було описано вище для функції preg_match ().

Щоб познайомитися на практиці з різними способами збереження результату, можна запустити програму з лістингу 7.8.

Лістинг 7.8.

```
<?php ## Різні опції preg_match_all().
Header("Content-type: text/plain");
$flags = array(
  "PREG_PATTERN_ORDER",
  "PREG_SET_ORDER",
  "PREG_SET_ORDER|PREG_OFFSET_CAPTURE"
);
$re   = '|<(\w+).*?>(.*?)</\1>|s';
$text = "<b>текст</b> і ще <i>інший текст</i>";
echo "Рядок: $text\n";
echo "Вираз: $re\n\n";
foreach ($flags as $name) {
  preg_match_all($re, $text, $pockets, eval("return $name;"));
  echo "Прапор $name:\n";
  print_r($pockets);
  echo "\n";
}
?>
```

Фрагмент результату, що відповідає прапору - PREG_SET_ORDER.

```
Array (
[0] => Array (
[0] => <b>текст</b>
[1] => b
[2] => текст
)
[1] => Array(
[0] => <i>інший текст</i>
[1] => i
[2] => інший текст
)
)
```

Заміна збігів

Всі функції заміни по регулярному виразу за умовчанням працюють в однорядковому режимі, нібито вказаний модифікатор /s. Рекомендується явно використовувати /m, коли це необхідно.

```
mixed preg_replace (mixed $expr, mixed $to, mixed $str [,int
$limit])
```

Коротко дія функції така: береться регулярний вираз $expr, шукаються всі його збіги в рядку $str і замінюються на рядок $to. Перед заміною спеціальні символи $0, $1 і т. д., а також \0, \1 і т.д. інтерполюються: замість них підставляються підрядки, відповідні виразам у дужках всередині $exprг (відповідно, "нульового" рівня - весь збіг, першого рівня - перша відкриваюча дужка і т. д.). Функція повертає результат роботи.

Якщо вказаний параметр $limit, то буде проведено не більш $limit пошуків і замін. Це зручно, якщо, потрібно провести одноразову заміну в рядку - тільки першого збігу.

Три перших параметри є типу mixed. Тобто вони можуть бути не тільки рядками, але і масивами (точніше, списками). Це дає функції справді колосальні можливості.

Якщо $str є списком рядків, то заміна проводиться в кожному елементі даного списку, і результат, також у вигляді списку, повертається.

Якщо ж $expr є списком регулярних виразів, а $to - звичайним рядком, то всі вирази з $expr будуть по черзі знайдені в $str і замінені на фіксований рядок $to.

Нарешті, якщо і $expr, і $to є списками, тоді PHP діє так. Він попарно витягує елементи з $expr і $to і проводить заміну: $expr[$i] => $to[$i], де $i пробігає всі можливі значення. Якщо чергового елементу $to[$i] не опиниться (масив $to коротше, ніж $expr), то відбудеться заміна на порожній рядок.

Модифікатор /e в регулярних виразах примушує функцію виконати замінюваний рядок, як код на PHP, і використовувати отриманий результат для

підстановки, з цим пов’язані деякі проблеми, які вирішуються з допомогою функції preg_replace_callback.

```
mixed preg_replace_callback(mixed $expr, string $funcName, mixed
$str [,int $limit])
```

Замість того щоб відразу замінювати знайдені збіги на рядкові значення, ця процедура запускає функцію, чиє ім'я передане їй в параметрі $funcName, і передає їй вміст кишень. Результат, повернений функцією, використовується для підстановки.

Далі наведений коректний код, який переводить всі теги в HTML-документі у верхній регістр (лістинг 7.9).

Лістинг 7.9.

```
<?php ## Функція preg_replace_callback() у дії.
// Призначена для користувача функція. Викликатиметься для
// кожного співпадіння з регулярним виразом.
function toUpper($pockets){
  return $pockets[1].strtoupper($pockets[2]).$pockets[3];
}
$str = '<hTmL><bOdY bgcolor="white">Тіло
документу</bOdY></html>';
$str = preg_replace_callback('{(</?)(\w+)(.*?>)}s', "toUpper"
$str);
echo htmlspecialchars($str);
?>
```

Ми отримаємо такий результат:

```
<HTML><BODY bgcolor="white">Тіло документу</BODY></HTML>
```

Всі теги були коректно перетворені.

Простір для творчості з використанням функції preg_replace_callback() справді невичерпний. Власне, вона "уміє" все те ж що уміє preg_replace(). Ось деякі речі, які можна робити з її допомогою:

- автоматично проставляти атрибути width і height у тегів <img>, отримані в результаті перехоплення вихідного потоку скрипта (функції ob_start() і т.д.);

- реалізовувати "розумну" заміну "псевдотегів" з параметрами (наприклад [font size=10]), що зазвичай потрібний у форумах і гостьових книгах;
- виконувати підстановки PHP-коду в різні шаблони і т.д.

Завдання

0. Написати регулярний вираз, який визначає чи заданий рядок є GUID з/без фігурних дужок. Де GUID це стрічка, яка складається з 8, 4, 4, 4, 12 шістнадцяткових цифр розділених тире.

   Приклад правильних рядків:
   ```
   {e02fa0e4-01ad-090A-c130-0d05a0008ba0}
   e02fd0e4-00fd-090A-ca30-0d00a0038ba0
   ```
   Приклад неправильних рядків:
   ```
   02fa0e4-01ad-090A-c130-0d05a0008ba0}
   e02fd0e400fd090Aca300d00a0038ba0
   ```

1. Написати регулярний вираз, який визначає чи є заданий рядок правильним MAC-адресом.

   Приклад правильних рядків:
   ```
   01:32:54:67:89:AB
   aE:dC:cA:56:76:54
   ```
   Приклад неправильних рядків:
   ```
   01:33:47:65:89:ab:cd
   01:23:45:67:89:Az
   ```

2. Написати регулярний вираз, який визначає чи є заданий рядок валідною URL адресою. В данім завданні правильним URL рахується адреса http и https, явне вказання протоколу також може бути відсутнім. Враховуються тільки адреса, що складаються з символів, тобто IP адреса в якості URL не потрібно провіряти. Допускаються піддомени, вказання порта доступа через двокрапку, GET запроси з передачею параметрів, доступ до підпапок на домені, допускається наявність якоря через решітку. Однобуквенні домени рахуються забороненими. Заборонені спецсимволи, наприклад «-» на початку і в кінці імені домена. Заборонений символ «_» і пробіл в імені домена. При складанні регулярного виразу потрібно орієнтуватись на список правильних і неправильних стрічок:

   Приклад правильних рядків:
   ```
   http://www.zcontest.ru
   http://zcontest.ru
   http://zcontest.com
   https://zcontest.ru
   https://sub.zcontest-ru.com:8080
   http://zcontest.ru/dir%201/dir_2/program.ext?var1=x&var2=my%20val
   zcon.com/index.html#bookmark
   ```

Приклад неправильних рядків:

```
Just Text.
http://a.com
http://www.domain-.com
```

3. Написати регулярний вираз, який визначає чи є заданий рядок шістнадцятковим ідентифікатором кольору в HTML. Де #FFFFFF для білого, #000000 для чорного, #FF0000 для червоного і.т.д.

Приклад правильних рядків:

```
#FFFFFF
#FF3421
#00ff00
```

Приклад неправильних рядків:

```
232323
f#fddee
#fd2
```

4. Написати регулярний вираз, який визначає чи є заданий рядок датою у форматі dd/mm/yyyy. Починаючи з 1600 року до 9999 року

Приклад правильних рядків:

```
29/02/2000
30/04/2003
01/01/2003
```

Приклад неправильних рядків:

```
29/02/2001
30-04-2003
1/1/1899
```

5. Написати регулярний вираз, який визначає чи є заданий рядок валідним E-mail адресом згідно RFC під номером 2822

Приклад правильних рядків:

```
mail@mail.ru
valid@megapochta.com
aa@aa.info
```

Приклад неправильних рядків:

```
bug@@@com.ru
@val.ru
Just Text2
val@val
val@val.a.a.a.a
```

```
12323123@111[][][]
```

6. Написати регулярний вираз, який визначає чи є заданий рядок IP-адресою, записаним в десятковому виді

Приклад правильних стрічок:
```
127.0.0.1
255.255.255.0
192.168.0.1
```

Приклад неправильних стрічок:
```
1300.6.7.8
abc.def.gha.bcd
254.hzf.bar.10
```

7. Перевірити чи надійно складений пароль. Пароль рахується надійним, якщо він складається з 8 або більше символів. Де символом може бути англійська буква, цифра і знак підкреслення. Пароль повинен містити хоча б одну заголовну букву, одну маленьку букву і одну цифру.

Приклад правильних рядків:
```
C00l_Pass
SupperPas1
```

Приклад неправильних рядків:
```
Cool_pass
C00l
```

8. Перевірити чи є заданий рядок шестизначним числом, записаним в десятковій системі числення без нулів у старших розрядах.

Приклад правильних рядків:
```
123456
234567
```

Приклад неправильних рядків:
```
1234567
12345
```

9. Перевірити чи задана стрічка відповідає відформатованому числу, з правильними розділювачами між розрядами тисяч и правильним знаком після коми.

Приклад правильних рядків:
```
10,000,000.45
10 000 000,45
```

Приклад неправильних рядків:
`123.456.789`

10. Написати частину аналізатора MakrDown, який задає курсив стрічці, яка оточена зірками, наприклад стрічка *this is italic* повинна бути перетворена в this is italic. Проте це не повинно вплинути на виділення жирним, який задається двома зірками.

11. Необхідно обернути слово, яке повторюється в елемент `<strong>`. Наприклад, стрічка `this is is a test` повинна бути перетворена в `this is <strong>is</strong> a test`.

12. Написати частину аналізатора MakrDown, який перетворює рядок
`# Заголовок першого рівня`, або два рядки
`Заголовок першого рівня`
`==========`
у `<h1>Заголовок першого рівня</h1>`.

13. Написати регулярний вираз, який визначає чи є заданий рядок коректним номером українського стільникового зв'язку.
14. Написати регулярний вираз, який визначає чи є заданий рядок коректним автомобільним номером зареєстрованим у Вашій області.
15. Написати частину аналізатора bb-code, який перетворює частини тексту `[img]деякий URL[/img]` у `<img src="деякий URL" />`.

Хід роботи

1. Створити новий файл у текстовому редакторі (Dreamweaver, Notepad або іншому).
2. Написати розв'язок завдання.
3. Зберегти програму на диск під новим ім`ям 1.php у папку <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB07
4. Перевірити виконання скрипта у браузері.
5. При виникненні помилки – відкоригувати програму та виконати її.
6. Оформити звіт по виконаній роботі.

Методичні рекомендації

Для введення даних використовуються параметри в URL, для виконання програми потрібно у браузері ввести <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB07\1.php?s=<зн1>.

Розв’якок завдання (текст програми):

```
<?php
  $s = $_GET['s'];
  $re = '/^([0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-
9a-f]{12})|(\{[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-
9a-f]{12}\})$/i';
  if (preg_match($re, $s)) echo "Рядок '{$s }' є GUID"; else echo
"Рядок '{$s }' не є GUID";
?>
```

Контрольні запитання

1. Яку функцію небхідно застосовувати для пошуку збігу?
2. Як замінювати підрядки у рядку які задовольняють певному регулярному виразу?
3. Як знайти усі збіги у тексті?
4. Що таке 'кишені'?
5. Які модифікатори Вам відомі?

Тема 8: Передача даних методами GET та POST, їх перевірка на коректність, опрацювання та вивід

Мета роботи: навчитись користатисть методами GET та POST для передачі даних PHP-сценарію з подальший опрацюванням та виводом.

Теоретичні відомості

Передача даних командного рядка

Поставимо перед собою завдання написати сценарій, який приймає в параметрах дві величини: зареєстроване ім'я і пароль. Якщо зареєстроване ім'я рівне root, а пароль – Z10N0101, слід надрукувати: "Доступ відкритий для користувача <ім'я>" і заблокувати сервер (тобто вивести стандартний екран Windows "Блокування" із запитом пароля для розблокування). Якщо ж дані невірні, необхідно вивести повідомлення "Доступ закритий!".

Спочатку розглянемо найбільш простий спосіб передачі параметрів сценарію - безпосередній набір їх в URL після знаку ? - наприклад, у форматі login=ім'я&password=пароль. Правда, навіть програмістові досить утомливо набирати цей рядок вручну. Всякі там ?, &, %... На щастя, існують зручні можливості мови HTML, які, звичайно, підтримуються всіма браузерами.

Отже, хай у нас на сервері в кореневому каталозі є сценарій на PHP під назвою hello.php. Наш сценарій розпізнає 2 параметри: login і password. Таким чином, якщо задати в адресному рядку браузера

```
http://example.com/hello.php?login=root&password=Z10N0101
```

ми повинні отримати необхідний результат.

Як тільки завдання відоме, можна приступати до його рішення. Але перш за все буває корисно вирішити аналогічну, але простішу задачу. Отже, як же нам в сценарії отримати рядок параметрів, переданий після знаку питання в URL при зверненні до сценарію? Для цього можна проаналізувати змінну оточення QUERY_STRING, яка в PHP доступна під ім'ям $_SERVER['QUERY_STRING']. Напишемо невеликий приклад, щоб це проілюструвати (лістинг 8.1).

Лістинг 8.1.

```
<!-- Виведення параметрів командного рядка. -->
<html><body>
<?php
echo "Дані з командного рядка: $_SERVER[QUERY_STRING]";
?>
</body></html>
```

Якщо запустити цей сценарій з браузера (перед цим зберігши його у файлі test.php у кореневому каталозі сервера) приблизно ось таким чином:

```
http://example.com/qs. php?this+is+the+world
```

то отримаємо документ наступного змісту:

```
Дані з командного рядка: this+is+the+world
```

URL-декодування символів не відбулося: рядок $_SERVER['QUERY_STRING'], як і однойменна змінна оточення, завжди приходить в тій же самій формі, в якій вона була послана браузером.

Оскільки PHP спочатку створювався саме як мова для Веб-програмування, то вона додатково проводить деяку роботу із змінній QUERY_STRING перед передачею управління сценарію. А саме, вона розбиває її по пробільних символах (у нашому прикладі пропусків немає, їх замінюють символи +, але ці символи PHP також розуміє правильно) і поміщає отримані шматочки в масив-список $argv, який згодом може бути проаналізований в програмі.

Все ж таки масив $argv використовується при програмуванні на PHP украй рідко, що пов'язане з набагато більшими можливостями інтерпретатора по розбору даних, що поступили від користувача. Проте в деяких ситуаціях його застосування виправдане, так що не варто забувати про цю можливість.

Форми

Для виконання попереднього завдання доведеться створити що-небудь типу діалогового вікна Windows, тільки в браузері. Отже, знадобиться звичайний

HTML-документ (наприклад, form.html у кореневому каталозі) з елементами цього діалогу - текстовими полями - і кнопкою (лістинг 8.2).

Лістинг 8.2.

```
<!-- Сторінка з формою -->
<html><body>
<form action=hello.php>
Логін: <input type=text name="login" value=""><br>
Пароль: < input type=password name="password" value=""><br>
<input type=submit value="Натисніть кнопку, щоб запустити
сценарій!">
</form>
</body></html>
```

Якщо заповнити поля введення і натиснути кнопку, браузер звернеться до сценарію hello.php і передасть через ? всі атрибути, розташовані всередині тегів <input> у формі і розділені символом & у рядку параметрів. У атрибуті action тега <form> заданий відносний шлях, тобто сценарій hello.php шукатиметься браузером в тому ж самому каталозі, що і файл form.html.

Всі перекодування і перетворення, які потрібні для URL-кодування даних, здійснюються браузером автоматично. Зокрема, букви кирилиці перетворяться на %xx, де xx - деяке шістнадцяткове число, що позначає код символу.

Використання форм дозволяє в принципі не навантажувати користувача такою інформацією, як ім'я сценарію, його параметри і т.д. Він завжди матиме справу тільки з полями, перемикачами і кнопками форми.

Залишилося тепер тільки визначитися, як можна витягнути $login і $password з рядка параметрів. Звичайно, можна спробувати розібрати його "вручну" за допомогою стандартних функцій роботи з рядкамиі цей прийом дійсно працюватиме. Проте, перш ніж братися за непотрібну справу, подивимося, що пропонує сама мова PHP.

Трансляція полів форми

Інтерпретатор перед запуском сценарію сам розбирає змінну оточення QUERY_STRING. Причому незалежно від того, яким методом - get або post - скористався браузер. Тобто, PHP сам визначає, який метод був задіяний, і отримує дані або з QUERY_STRING, або із стандартного вхідного потоку.

Всі дані з полів форми PHP поміщає в глобальний масив $_REQUEST. У нашому прикладі значення поля login після початку роботи програми зберігатиметься у $_REQUEST['login'], а значення поля password - в $_REQUEST['password']. Тобто, не треба нічого нізвідки "отримувати" - все вже встановлено і розпаковано з URL-кодування.

Крім того, щоб можна було якось розділити GET-параметри даних, PHP також створює масиви $_GET і $_POST, заповнюючи їх відповідними значеннями.

Остаточний сценарій hello.php приведений в лістингу 8.3.

Лістинг 8.3.

```
<!-- Використання даних форми -->
<html><body>
<?php
if ($_REQUEST['login']=="root" &&
$_REQUEST['password']=="Z10N0101") {
  echo "Доступ відкритий для користувача $_REQUEST[login]";
  // Команда блокування робочої станції (працює в NT-системах)
  system("rundll32.exe user32.dll,LockWorkStation");
} else {
  echo "Доступ закритий!";
}
?>
</html></body>
```

Тепер зробимо так, щоб при запуску без параметрів сценарій видавав документ з формою, а при натисненні кнопки - виводив потрібний текст. Найпростіший спосіб визначити, чи був сценарій запущений без параметрів - перевірити, чи існує змінна з ім'ям, співпадаючим з ім'ям кнопки відправки. Якщо

така змінна існує, то, очевидно, що користувач запустив програму, натиснувши кнопку (лістинг 8.4).

Лістинг 8.4.

```
<! -- Вдосконалений скрипт блокування сервера -- >
<html><body>
<?if (!isset($_REQUEST['doGo'])) {?>
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
Ім'я: <input type=text name="login" value=""<br>
Пароль: <input type=password name="password" value=""<br>
input type=submit name="doGo" value="Натисніть кнопку!">
</form>
<?) else {
if ($_REQDEST['login'] == "root" && $_REQUEST['password'] ==
"Z10N0101") {
 echo "Доступ відкритий для користувача $_REQUEST[login]";
// Команда блокування робочої станції (працює в NT-системах)
System("rundll32.exe user32.dll, LockWorkStation");
) else {
echo  "Доступ закритий!"; }
</html></body>
```

У цьому прикладі використовується конструкція <?=значення?>. Вона є нічим іншим, як просто коротшим позначенням для <?echo вираз?>, і призначена для того, щоб вставляти величини прямо в HTML-сторінку. Також зникла необхідність і в проміжному файлі form.html: його код вбудований в сам сценарій.

Трансляція змінних оточення

Проте "інтелектуальні" можливості PHP на цьому далеко не вичерпуються. Річ у тому, що в змінні перетворяться не тільки всі дані форми, але і змінні оточення (включаючи QUERY_STRING, CONTENT_LENGTH і багато інших).

Наприклад, приведемо сценарій (лістинг 8.5), що друкує IP-адресу користувача, який його запустив, а також тип його браузера (ці дані зберігаються

в змінних оточення REMOTE_ ADDR і HTTP_USER_AGENT, доступий в скрипті через масив $_SERVER).

Лістинг 8.5.

```
<!-- Виведення IP-адреси і браузера користувача -->
<html><body>
Ваша IP-адреса: <?=$_SERVER['REMOTE_ADDR']?><br>
Ваш браузер: <?=$_SERVER['HTTP_USER_AGENT']?>
</body></html>
```

Обробка списків

Механізм трансляції полів форми в PHP працює прийнятно, коли серед них немає полів з однаковими іменами. Якщо ж такі зустрічаються, то в змінну записуються тільки дані останнього поля, що зустрілося. Це досить-таки незручно при роботі, наприклад, із списком множинного вибору <select multiple>

```
<select name=Sel multiple>
<option>First
<option>Second
<option>Third
</select>
```

У такому списку ви можете вибрати (підсвітити) не один, а відразу декілька рядків, використовуючи клавішу <Ctrl> і клацаючи по ним кнопкою миші. Нехай вибрано First і Third. Тоді після відправки форми сценарію прийде рядок параметрів sel=First&sel=Third, і в змінній $_REQUEST['sel'] опиниться, звичайно, тільки Third. Для вирішення подібних проблем в PHP передбачена можливість давати імена полям форми у вигляді "масиву з індексами":

```
<select name="Sel[]" multiple>
<option>First
<option>Second
<option>Third
</select>
```

Тепер сценарію прийде рядок sel[]=First&sel[]=Third, інтерпретатор виявить, що потрібно створити "автомасив" (тобто масив, який не містить пропусків і у якого індексація починається з нуля), і, дійсно, створить запис

```
$_REQUEST['Sel'] типу "масив", вміст якого наступний:
array (0=>"First", 1=>"Third").
```

В результаті отримано в $_REQUEST масив масивів (або двовимірний масив, як його ще називають), доступ до елементів якого можна отримати так:

```
echo $_REQUEST['Sel'][0];   //  виводить перший елемент
echo $_REQUEST['Sel'][1];   //  другий
```

Прийом з автомасивом в полі <Select muitipie>, дійсно, виглядає досить елегантно. Проте не варто думати, що він застосовний тільки до цього елементу форми: автомасиви можна застосовувати і в будь-яких інших полях. Ось приклад, що створює два перемикачі (кнопки із значеннями вкл/викл), один елемент введення рядка і одне текстове (багаторядкове) поле, причому всі дані після запуску сценарію, оброблювального цю форму, будуть представлені у вигляді одного-єдиного автомасиву:

```
<input type=checkbox name=Arr[] value=chl>
<input type=checkbox name=Arr[] value=ch2>
<input type=text name=Arr[] value="Some string">
<textarea name=Arr[]>Some text</textarea>
```

PHP абсолютно немає ніякої справи до того, в яких елементах форми використовуються автомасиви - він у будь-якому випадку обробляє все однаково.

Обробка масивів

Роглянемо ще одну корисну властивість PHP. Нехай є така форма:

```
Ім'я: <input type=text name=Data[name]<br>
Адреса: <input type=text name=Data[address]><br>
Місто:<br>
<input type=radio name=Data[city] value=Kyiv>Київ<br>
<input type=radio name=Data[city] value=Lviv>Львів<br>
<input type=radio name=Data[city] value=Odesa>Одеса<br>
```

Можна здогадатися, що після передачі подібних даних сценарію на PHP в ньому ініціалізується ассоциативний масив $Data з ключами name, address і city. Тобто, імена полям форми можна давати не тільки прості, але і представлені у вигляді одновимірних асоціативних масивів.

$_REQUEST['Data']['city'] позначає значення тієї радіокнопки, яка була вибрана користувачем, а $_REQUEST['Data']['name'] - введене ім'я. У сценарії потрібно брати ключі в лапки або апострофи - інакше інтерпретатором буде виведено попередження. В той же час, в параметрах name поля форми, навпаки, потрібно їх уникати.

Порядок трансляції змінних

Нехай є параметр a=10, що поступив з QUERY_STRING, параметр a=20 з POST-запиту, і Cookie a=30. За умовчанням трансляція виконується в порядку GET-POST-COOKIE(GPC), причому кожна наступна змінна перекриває попереднє своє значення (якщо воно існувало). Отже, в змінну $a (якщо включений режим register_globals) сценарію і в $_request['a'] буде записано 30, оскільки COOKIE перекриває POST і GET.

У режимі register_globals в глобальні змінні потрапляють також значення змінних оточення. Записуються вони відповідно до схеми ENVIRONMENT-GET-POST-COOKIE(EGPC). Іншими словами, змінні оточення в режимі register_globals перекриваються навіть GET-даними, і зловмисник може "підроблювати" будь-яку з них, передавши відповідну змінну QUERY_STRING при запуску сценарію. Отже, для запобігання можливих проблем, навіть в режимі register_globals потрібно звертайтись до змінних оточення тільки через

```
$_SERVER['змінна'] АБО getenv('змінна').
```

Особливості прапорців checkbox

Незалежний перемикач (checkbox або коротше - прапорець) має одну досить неприємну особливість. Коли перед відправкою форми користувач встановив його у вибраний стан, то сценарію в числі інших параметрів приходить

пара імя_прапорця=значення. В той же час, якщо прапорець не був встановлений користувачем, вказана пара не посилається. Часто це буває не зовсім те, що потрібне. Зручніше, щоб в невибраному стані прапорець також присилав дані, але тільки значення було рівне якій-небудь спеціальній величині - наприклад, нулю або порожньому рядку.

Зробити це можна за допомогою прихованого поля (hidden) з тим самим ім'я та із значенням, рівним, наприклад, нулю, помістивши його перед потрібним прапорцем (лістинг 8.6).

Лістинг 8.6.

```
<?php ## Гарантований прийом значень від прапорців.
if (@$_REQUEST['doGo']) {
  foreach (@$_REQUEST['known'] as $k=>$v) {
    if($v) echo "Ви знаєте мову $k!<br>";
    else echo "Ви не знаєте мови $k. <br>";
  }
}
?>
<form action="<?=$_SERVER['SCRIPT_NAME']?>" method=post>
Які мови програмування ви знаєте?<br>
<input type=hidden name="known[PHP]" value="0">
  <input type=checkbox name="known[PHP]" value="1">PHP<br>
<input type=hidden name="known[Perl]" value="0">
  <input type=checkbox name="known[Perl]" value="1">Perl<br>
<input type=submit name="doGo" value="Go!">
</form>
```

Тепер у випадку, якщо користувач не вибере ніякий з прапорців, браузер відправить сценарію пару known['мова'] = 0, що згенерована відповідним прихованим полем, і в масиві $_REQUEST['known'] створиться відповідний елемент. Якщо користувач вибере прапорець, ця пара також буде послана, але відразу ж після неї послідує пара known['мова'] =1, яка "перекриє" попереднє значення.

Завдання

Створити інтерфейс для введення даних про:

0. студента
1. користувача сайту
2. тварину
3. мобільний телефон
4. магазин
5. книгу
6. комп’ютер
7. комп’ютерну програму
8. організацію
9. комп’ютерні комплектуючі
10. музичну композицію
11. фільм
12. автомобіль
13. сайт
14. фотоапарат
15. кімнатну рослину

для цього потрібно:

A. створити html-форму
B. необхідно використовувати такі елементи:
   1) поле вводу
   2) список, випадаючий список
   3) радіо кнопки
   4) чек бокси
   5) приховане поле
   6) багаторядкове поле вводу
   7) кнопки
   8) при доречності: поле пароля, приховане поле
C. забезпечити перевірку отриманих даних на коректність
D. забезпечити перехід на форму, якщо дані не коректні для коригування;
E. вивести введені дані, якщо дані коректні;

Хід роботи

1. Створити новий файл у текстовому редакторі (Dreamweaver, Notepad або іншому).
2. Написати розв’язок завдання.
3. Зберегти HTML-форму у файл form.html, а скрипт у файл form.php та помістити їх у папку <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB08
4. Перевірити виконання скрипта у браузері.

5. При виникненні помилки – відкоригувати програму та виконати її.
6. Оформити звіт по виконаній роботі.

Методичні рекомендації

HTML-форма:

```
<html>
<head>
<title>Форма</title>
</head>
<body>

<form action="form.php" method=post>
<table border="1">
<tr>
    <td><strong>№ зал книги</strong></td>
    <td><input name="nzk" length=6></td>
</tr>
<tr>
    <td><strong>Прізвище</strong></td>
    <td><input name="prizv" length=20></td>
</tr>
<tr>
    <td><strong>Стать</strong></td>
    <td><input type="radio" name="stat" value="m">Чоловіча
        <input type="radio" name="stat" value="f">Жіноча
    </td>
</tr>
<tr>
    <td><strong>Староста?</strong></td>
    <td><input type="checkbox" name="starosta" value="1"></td>
</tr>
<tr>
    <td><strong>Курс</strong></td>
    <td>
    <select name="kurs">
    <option value=0 selected>-Виберіть курс-
```

```
    <option value=1>Перший курс
    <option value=2>Другий курс
    <option value=3>Третій курс
    <option value=4>Четвертий курс
    <option value=5>П'ятий курс
    <option value=6>Шостий курс
    </select>
    </td>
</tr>
<tr>
    <td colspan=2>
      <strong>Примітка:</strong><br/>
      <textarea name="prym"></textarea></td>
</tr>
<tr>
    <td coldpan=2 align=center><input type="submit"
value="Відправити"></td>
</tr>
</table>
</form>


</body>
</html>
```

Скрипт обробки отриманих даних:

```
<?php
  $nzk   =  $_POST["nzk"];
  $prizv =  $_POST["prizv"];
  $stat  =  $_POST["stat"];
  $starosta  =  $_POST["starosta"];
  $kurs  =  (int) $_POST["kurs"];
  $prym  =  $_POST["prym"];


$stata = array("m" => "Чоловіча", "f" => "Жіноча");
$kursa = array("1" => "Перший", "2" => "Другий", "3" => "Третій",
"4" => "Четвертий", "5" => "П'ятий", "6" => "Шостий");
if ($starosta == 1) $starostam = "Так"; else $starostam = "Ні";
echo "Ви ввели:";
```

```
echo "№ зал книги: $nzk<br/>";
echo "Прізвище: $prizv<br/>";
echo "Стать: {$stata[$stat]}<br/>";
echo "Староста: $starostam<br/>";
echo " Курс: {$kursa[$kurs]}<br/>";
echo " Примітка: $prym<br/>";

if (!preg_match('/^\d{6}$/is', $nzk)) echo "Не коректно введений
номер залікової книжки";
if (!preg_match('/^[A-ЯІЇЄ]{1}[a-яіїє]{2,30}$/s', $prizv)) echo
"Не коректно введене прізвище ";

?>
```

Для виконання програми потрібно у браузері ввести <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB08\form.html.

Контрольні запитання

1. Як передати дані методом GET?
2. Як передати дані методом POST?
3. Як визначити поле вводу тексту у формах?
4. Як визначити поле вводу пароля у формах?
5. Як визначити приховані поля у формах?
6. Як визначити кнопки у формах?
7. Як визначити багаторядкове поле вводу у формах?
8. Як визначити меню вибору у формах?
9. Як працювати із глобальною змінною $_GET?
10. Як працювати із глобальною змінною $_POST?

Тема 9: Робота з файловою системою та файлами.

Мета роботи: Навчитись зчитувати з файлів та записувати у файли на мові PHP.

Теоретичні відомості

Інтерпретатор PHP працює як з прямими (/) так і з оберненими (\) флешами, він в любому випадку переведе їх в ту форму, яка потрібна вашій ОС. Функціям по роботі з повними іменами файлів також буде все рівно, який слеш використовується. Також можна працювати з файлами на віддалених Веб-серверах в точності, як і з своїми власними (тільки записувати в них можна не завжди).

Якщо ім’я файлу представлено рядком http:// або ftp://, то PHP розуміє, що потрібно на справді встановити мережене з’єднання і працювати власне з ним, а не з файлом. При цьому у програмі такий файл нічим не відрізняється від звичайного (якщо є відповідні права, то можна і записувати в такий HTTP- або FTP-файл).

Далі представлені основні функції для роботи з файлами.

```
bool file_exists (string filename)
```

Перевіряє, чи існує файл. Повертає TRUE, якщо файл, специфікований параметром filename, існує; FALSE в іншому випадку.

Ця функція не працюватиме з віддаленими файлами; файл, що перевіряється, повинен бути доступний через файлову систему сервера. Результати роботи цієї функції кешуються.

```
int filesize (string filename)
```

Одержує розмір файлу. Повертає розмір файлу в байтах або FALSE - у разі помилки. Результати роботи цієї функції кешуються.

Ця функція не працюватиме з віддаленими файлами; файл, що перевіряється, зобов'язаний бути доступний через файлову систему сервера.

```
int fopen (string filename, string mode [, int use_include_path])
```

Відкриває файл або URL. Якщо filename починається з "http://" (нечутливо до регістру), то HTTP 1.0 з'єднання відкривається із специфікованим сервером, сторінка запрошується методом HTTP GET і покажчик файлу повертається в початок тіла відповіді. 'Host:'-header відправляється разом із запитом, щоб обробити віртуальні хости на основі імен.

Починаючи з PHP 4.3.0, можна використовувати "https://" для відкриття HTTP-з'єднання через SSL.

Варто звернути увагу, що покажчик файлу дозволяє запрошувати тільки тіло відповіді; для запиту шапки HTTP-відповіді потрібно використовувати PHP 4.0.5 або новіше; ці шапки зберігатимуться в змінній $http_response_header. Починаючи з PHP 4.3.0, header-інформація може бути запитана з використанням функції file_get_wrapper_data().

HTTP-з'єднання є тільки для читання; не можна записувати дані або копіювати файли в HTTP-ресурс.

Версії до PHP 4.0.5 не обробляють HTTP-перенаправлення. Тому директорії зобов'язані мати ведучі слеші.

Якщо filename починається з "ftp://" (нечутливо до регістра), відкривається ftp-з'єднання із специфікованим сервером і повертається покажчик на запрошуваний файл. Якщо сервер не підтримує ftp пасивного режиму, це не вдасться зробити. Можна відкривати файли для читання або запису через ftp (але не для того і іншого одночасно). Якщо віддалений файл вже існує на ftp-сервері і йде спроба відкрити його для запису, це не вийде. Якщо потрібно відновити існуючі файли по ftp, потрібно використовувати ftp_connect().

Якщо filename це "php://stdin", "php://stdout" або "php://stderr", буде відкритий відповідний потік stdio.

Якщо filename починається ще з чого-небудь, файл буде відкритий з файлової системи і буде повернений покажчик на відкритий файл.

Якщо відкрити файл не вдалося, ця функція повертає FALSE.

mode може мати значення:

- 'r' - Відкрити тільки для читання; помістити покажчик в початок файлу.
- 'r+' - Відкрити для читання і запису; помістити покажчик в початок файлу.
- 'w' - Відкрити тільки для запису; помістити покажчик в початок файлу і встановити нульову довжину. Якщо файл не існує, робиться спроба створити його.
- 'w+' - Відкрити для читання і запису; помістити покажчик в початок файлу і встановити нульову довжину. Якщо файл не існує, робиться спроба створити його.
- 'a' - Відкрити тільки для запису; помістити покажчик в кінець файлу. Якщо файл не існує, робиться спроба створити його.
- 'a+' - Відкрити для читання і запису; помістити покажчик в кінець файлу. Якщо файл не існує, робиться спроба створити його.

Примітка: mode може містити символ 'b'. Це використовується тільки в системах, що розрізняють двійкові і текстові файли (тобто в Windows. У Unix це марно). Якщо не потрібен, він ігнорується.

Можна використовувати необов'язковий третій параметр і встановити в нього значення "1", якщо потрібно знайти файл також і в include_path.

Приклад. fopen()

```
$fp = fopen ("/home/rasmus/file.txt", "r");
$fp = fopen ("/home/rasmus/file.gif", "wb");
$fp = fopen ("http://www.example.com/", "r");
$fp = fopen ("ftp://user:password@example.com/", "w");
```

Якщо виникли проблеми з читанням і записом файлів і використовується версії серверних модулів PHP, потрібное переконатися, що використовувані файли і директорії доступні для серверного процесу.

На платформі Windows потрібно мнемонізувати (escape) всі зворотні слеші в шляху до файлу або використовуйте звичайні слеши.

```
$fp = fopen ("c:\data\info.txt", "r");
```

fclose - закриває файл

```
bool fclose (int fp)
```

Закривається файл, на який вказує fp. Повертає TRUE при успіху, FALSE при невдачі.

Покажчик файлу повинен бути правильним і повинен вказувати на файл, успішно відкритий функцією fopen() або fsockopen().

```
string fread (int fp, int length)
```

fread() читає length байт з покажчика файлу, на який посилається fp. Читання зупиняється, якщо прочитано length байт або якщо досягнуть EOF (кінець файлу), дивлячись що перше.

```
// одержати вміст файлу в рядок
$filename = "/usr/local/something.txt";
$fd = fopen ($filename, "r");
$contents = fread($fd, filesize ($filename));
fclose ($fd);
```

У системах, які розрізняють двійкові і текстові файли (Windows), файл потрібно відкривати з 'b' як параметром режиму у функції fopen().

```
$filename = "c:\files\somepic.gif";
$fd = fopen ($filename, "rb");
$contents = fread ($fd, filesize ($filename));
fclose ($fd);
```

fgets - одержує рядок з покажчика на файл

```
string fgets (int fp [, int length])
```

Повертає рядок розміром довжина рядка - 1 байт, прочитаного з файлу, на який указує fp. Читання закінчується досягши точки length - 1 байт, символу newline (який включається в значення повернення каретки) або EOF (дивлячись що буде знайдено першим). Якщо length не вказаний, за замовченням length буде 1k, або 1024 байт.

Якщо виникла помилка, повертає FALSE.

Покажчик на файл повинен бути правильним і указувати на файл, успішно відкритий функціями fopen(), popen() або fsockopen().

Приклад. Порядкове читання файлу

```
$fd = fopen ("/tmp/inputfile.txt", "r");
while (!feof ($fd)) {
    $buffer = fgets($fd, 4096);
    echo $buffer;
}
fclose ($fd);
```

fwrite - запис у файл

```
int fwrite (int fp, string string [, int length])
```

fwrite() записує вміст рядка string в потік файлу, специфікованого покажчиком fp. Якщо аргумент length заданий, запис буде зупинений після запису length-байтів або досягнення кінця string, дивлячись що зустрінеться першим.

fwrite() повертає кількість записаних байт, або -1 при помилці.

Якщо аргумент length заданий, опція конфігурації magic_quotes_runtime буде проігнорована і слеші не будуть вирізані з string.

У системах, що розрізняють бінарні і текстові файли (Windows), файл повинен бути відкритий з буквою 'b' як параметр режиму функції fopen().

fputs - записує у файл

```
int fputs (int fp, string str [, int length])
```

fputs() це псевдонім fwrite(), ідентичний у всьому. Параметр length є необов'язковим і, якщо не визначений, записується весь рядок.

Завдання

0. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Вивести прізвища студентів, які вчаться на відмінно.
1. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Обчислити кількість студентів, які вчаться без трійок (на відмінно і добре).

2. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Вивести прізвища студентів, які вчаться без трійок (на відмінно і добре).
3. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Обчислити кількість студентів, які вчаться на відмінно.
4. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Обчислити процент студентів, які вчаться на відмінно.
5. Сформувати файл з інформацією про студентів: прізвище. оцінки з фізики, математики, інформатики. Обчислити процент студентів, які вчаться без трійок (на відмінно і добре).
6. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Для кожного студента вивести: прізвище і середній бал.
7. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Для кожного предмета обчислити середній бал.
8. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Обчислити кількість оцінок ”відмінно” по кожному предмету.
9. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Обчислити кількість оцінок ”добре” по кожному предмету.
10. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Обчислити кількість оцінок ”задовільно” по кожному предмету.
11. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Обчислити кількість кожної з оцінок ”5”, “4”, “3” по фізиці.
12. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Обчислити кількість кожної з оцінок ”5”, “4”, “3” по математиці.
13. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Обчислити кількість кожної з оцінок ”5”, “4”, “3” по інформатиці.
14. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Обчислити найбільший середній бал (порівнюючи середні бали для кожного студента).
15. Сформувати файл з інформацією про студентів: прізвище, оцінки з фізики, математики, інформатики. Обчислити кількість студентів, середній бал яких вищий 4,5.

Хід роботи

1. Створити новий файл у текстовому редакторі (Dreamweaver, Notepad або іншому).
2. Сформувати текстовий файл з інформацією про студентів.
3. Написати програму, яка зчитує дані з файлу та виводить дані, які вимагаються у завданні.
4. Зберегти програму на диск під новим ім`ям 1.php у папку <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB09
5. Перевірити виконання скрипта у браузері.
6. При виникненні помилки – відкоригувати програму та виконати її.
7. Оформити звіт по виконаній роботі.

Методичні рекомендації

Файл може мати наступний формат: інформація про кожного студента міститься у окремому рядку, дані розділяються між собою символом '|'.

Приклад файлу даних:

```
Іваненко|3|4|5
Петренко|4|5|4
Сидоренко|5|5|5
```

Розв’язок завдання (текст програми):

```
<?php
$file_name = "a.txt";
$f = @fopen($file_name, "r");
if ($f) {
echo "<table>\n ";
echo
"<tr><th>Прізвище</th><th>Фізика</th><th>Математика</th><th>Інфор
матика</th></tr>\n";
  while (($buffer = fgets($f, 4096))!==false) {
    list($surname, $fiz, $mat, $inf) = explode('|', $buffer);
    $a[] = array('surname' => $surname, 'fiz' => $fiz, 'mat' =>
$mat, 'inf' => $inf);
    echo
"<tr><td>{$surname}</td><td>{$fiz}</td><td>{$mat}</td><td>{$inf}<
/td></tr>\n ";  }
echo "</table>\n";
}
```

```
fclose($f);
echo "Відмінники: ";
foreach($a as $stud){
if (($stud['fiz'] == 5) && ($stud['mat'] == 5) && ($stud['inf']
== 5)) echo $stud['surname'];
}
?>
```

Для виконання програми потрібно у браузері ввести <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB09\1.php.

Контрольні запитання

1. Як перевірити існування файлу?
2. Як відкриття файл тільки для читання?
3. Як відкриття файл для читання і запису?
4. Як відкриття файл тільки для запису?
5. Як закрити файл?
6. Які є функції для бінарного зчитування з файлу?
7. Які є функції для порядкового зчитування з файлу?
8. Які є функції для запису у файлу?

Тема 10: Сесії та cookie

Мета роботи: навчитись користуватись сесіями та cookie.

Теоретичні відомості

Користувач, зайшовши на деякий сайт, звик вважати, що, починаючи з цього моменту, сайт тільки ним і займається. Наприклад, клієнт може авторизуватися - ввести ім'я користувача і пароль, і після цього подорожувати по сайту вже в режимі підвищених привілеїв (наприклад, він може проглядати розділи, недоступні звичайним користувачам).

Особливо ситуація "відстежування" користувача характерна для Інтернет-магазинів. Будь-який користувач, що зайшов на подібний сайт, відразу ж отримує в своє розпорядження так звану віртуальну корзину. Він може "складати" в корзину різні товари, описані на сайті, простим клацанням миші. "Вміст" корзини зберігається при переході від однієї сторінки Інтернет-магазину до іншої. Після того, як клієнт "набрав" собі товарів в "віртуальний візок", він може оформити їх покупку (наприклад, доставку додому).

Якщо б у процесі "бродіння" користувача по сайту ним "займався" один і той же екземпляр (процес) скрипта (і з'єднання з сервером постійно підтримувалося б відкритим), то жодних особливих проблем не виникло б. Дійсно, корзину покупця можна зберігати в якому-небудь асоціативному масиві сценарію. Проте у Веб все відбувається дещо складніше.

Веб-сервери завжди працюють в режимі "запит-відповідь", причому запити різних користувачів можуть приходити у будь-якому порядку (і навіть оброблятися одночасно). Коли користувач "заходить" браузером на деяку сторінку, з'єднання з сервером встановлюється на короткочасний період і розривається відразу ж після отримання даних HTML-сторінки. І хоча користувач бачить перед собою сторінку, сервер вже давно про нього "забув" і може займатися обробкою інших запитів.

Серверу кожну секунду приходять десятки запитів від різних користувачів. У цій лавині він повинен визначити, який запит якому клієнтові відповідає, і правильно зіставити з ним ту або іншу "віртуальну корзину". Перехід з однієї сторінки на іншу супроводжується запуском нового екземпляра скрипта, тому будь-які дані в "старому" сценарії, що видав попередню сторінку, втрачаються. Сесії надають механізм збереження цих даних.

Що таке сесія?

Сесії, що вперше з'явилися в PHP версії 4, є механізмом, що дозволяє зберігати деякі (і довільні) дані, індивідуальні для кожного користувача, між запусками сценарію. Такими даними можуть бути, наприклад, ім'я клієнта і його номер рахунку, або ж вміст "віртуальної корзини".

Фактично, сесія - це деяке місце довготривалої пам'яті (зазвичай частина на жорсткому диску і частина - в cookies браузера), яке зберігає свій стан між викликами сценаріїв одним і тим же користувачем. Помістивши в сесію змінну (будь-якої структури), при наступному запуску сценарію можна отримати її у повній цілості. Важко переоцінити зручності, які це надає нам, програмістам.

Навіщо потрібні сесії?

У Web-програмуванні є один клас завдань, який може викликати досить багато проблем, якщо писати сценарії "в лоб". Йдеться про слабку сторону CGI - неможливості запустити програму на тривалий час, дозволивши їй при цьому обмінюватися даними з користувачами.

Загалом і в цілому, сценарії повинні запускатися, вмить виконуватися і повертати управління системі. Нехай є форма, в якій таке велике число полів, що було б не зручно розміщати їх на одній сторінці. Тоді потрібно розбити процес заповнення форми на декілька етапів, або стадій, і представити їх у вигляді окремих HTML-документів. Це схоже на роботу майстрів Windows-діалогових вікон для введення даних з кнопками Назад і Далі, завдяки яким можна переміститься на крок в будь-якому напрямі.

Наприклад, в першому документі з діалогом у користувача може запрошуватися його ім'я і прізвище, в другому (якщо перший був заповнений вірно) - дані про його місце проживання, і в третьому - номер кредитної картки. У будь-який момент можна повернутися на крок назад, щоб виправити ті або інші дані. Нарешті, якщо все гаразд, накопичена інформація обробляється - наприклад, поміщається в базу даних.

Реалізація такої схеми виявляється для Web-додатків досить нетривіальною задачею. Дійсно, доведеться зберігати всі раніше введені дані в якому-небудь тимчасовому сховищі, яке повинне анулюватися, якщо користувач раптом передумає і "піде" з сайту. Для цього, можна використовувати функції серіалізації і файли. Проте ними вирішується тільки половину проблеми: потрібно також якось "прив'язувати" конкретного користувача до конкретного тимчасового сховища. Ці проблеми вирішуються із застосуванням сесій PHP.

Механізм роботи сесій

Спершу повинен існувати механізм, який би дозволив PHP ідентифікувати кожного користувача, що запустив сценарій. Тобто при наступному запуску PHP потрібно однозначно визначити, хто його запустив: та ж людина, або інша. Робиться це шляхом привласнення клієнтові так званого унікального ідентифікатора сесії, Session ID. Щоб цей ідентифікатор був доступний при кожному запуску сценарію, PHP поміщає його в cookies браузера (використовувати cookies не обов'язково, існує і інший спосіб).

Тепер, знаючи ідентифікатор (SID), PHP може визначити, в якому ж файлі на диску зберігаються дані користувача.

Для доступу до сесій існує глобальний масив $_SESSION, який PHP обробляє особливим чином. Помістивши в нього деякі дані, можна бути впевненим, що при наступному запуску сценарію тим же користувачем масив $_SESSION набуде того ж самого значення, яке було у нього при попередньому завершенні програми. Це відбудеться тому, що при завершенні сценарію PHP

автоматично зберігає масив $_session у тимчасовому сховищі, ім'я якого зберігається в SID.

Місце зберігання сесій можна налаштувати власноруч задавши відповідні функції і зареєструвавши їх як обробники сесії. Втім, робити це не обов'язково: у PHP вже існують обробники за умовчанням, які зберігають дані у файлах (у системах Unix для цього зазвичай використовується каталог /tmp).

Ініціалізація сесії

Для роботи із сесією, її необхідно ініціалізувати. Робиться це шляхом виклику спеціальної функції session_start().

Якщо в php.ini встановлений режим session.auto_start=l, то функція ініціалізації викликається автоматично при запуску сценарію. Проте, це позбавляє багатьох корисних можливостей (наприклад, не дозволяє вибирати свою, особливу, групу сесій). Отже краще не спокушати долю і викликати session_start в першому рядку програми. Також потрібно стежити за тим, щоб до неї не було ніякого виводу у браузер - інакше PHP не зможе встановити SID для користувача, адже SID зазвичай зберігається в cookies, які повинні бути встановлені до будь-якого оператора виводу в скрипті.

```
void session_start()
```

Ця функція ініціалізовує механізм сесій для поточного користувача, що запустив сценарій. По ходу ініціалізації вона виконує ряд дій:

- якщо відвідувач запускає програму вперше, у нього встановлюється cookies з унікальним ідентифікатором, і створюється тимчасове сховище, що асоціюється з цим ідентифікатором;
- визначається, яке сховище пов'язане з поточним ідентифікатором користувача;
- якщо в сховищі є якісь дані, вони поміщаються в масив $_SESSION;

- якщо параметр register_globals з файлу php.ini рівний on, то всі ключі в масиві $_SESSION і відповідні ним значення перетворюються на глобальні змінні.

Приклад використання сесії (лістинг 10.1).

```
Лістинг 10.1.
<?php ## Приклад роботи з сесіями.
session_start();
// Якщо на сайт тільки-тільки зайшли, обнуляємо лічильник.
if (!isset($_SESSION['count'])) $_SESSION['count'] = 0;
// Збільшуємо лічильник в сесії.
$_SESSION['count'] = $_SESSION['count'] + 1;
?>
<h2>Лічильник</h2>
У поточній сесії роботи з браузером Ви відкрили цю сторінку
<?=$_SESSION['count']?> раз(а).<br>
Закрийте браузер, щоб обнулити лічильник.<br>
<a href="<?=$_SERVER['SCRIPT_NAME']?>" target="_blank">Відкрити
дочірнє вікно браузера</a>.
```

Як видно, все "крутиться" навколо масиву $_SESSION. Робота відбувається з одним з його елементів, збільшуючи його при кожному запуску скрипта на одиницю.

Відкривши скрипт з лістингу 10.1 у браузері і кілька разів натиснути кнопку "Обновити". Буде видно, що лічильник почне збільшуватися. Але досить закрити браузер і відкрити новий, як цифра обнулиться. Отже, дані сесії пропадають при закритті браузера.

Відкривши інше, незалежне, вікно браузера, не закриваючи першого. По черзі натискаючи кнопку "Обновити" в обох вікнах, лічильники збільшуються незалежно один від одного. Отже, дані сесії "прив'язані" до вікна браузера, а не до призначеного для користувача комп'ютера.

Клацнувши на посиланні "Відкрити дочірнє вікно браузера", яке виводить скрипт. Відкриється нове вікно браузера (за рахунок -target="_blank"), проте лічильник не обнулиться. Більш того, "накрутивши" лічильник в новому вікні і

повернувшись до старого, можна бачити, що він і там теж змінився в точності на ту ж величину. При відкритті нового вікна клацанням по посиланню на деякій сторінці дані сесії спільно використовуються цим вікном з його батьком.

Знищення сесії

```
bool session_destroy()
```

Дана функція знищує сховище сесії. При цьому масив $_SESSION не очищається. Щоб повністю видалити сесію, потрібно виконати наступну послідовність команд:

```
// Очистити дані сесії для поточного сценарію.
$_SESSION = array();
// Видалити cookie, яка відповідає SID.
@unset($_COOKIE[session_name()]);
// Знищити сховище сесії.
session_destroy() ;
```

Очищення сховища сесії корисне тим, що починаючи з цього моменту всі сторінки, що використовують те ж саме сховище, отримають порожню сесію. Скрипти "логаута" (явного "виходу" з сайту; протилежність "логіну" - авторизації) найчастіше використовують дану послідовність команд.

Сесії і cookies

Cookies є найбільш елегантним і простим рішенням задачі ідентифікації кожного користувача, що підключився, що необхідно для зв'язку тимчасового сховища і даних сесії.

Проте користувачі можуть відключили cookies в своїх браузерах. Для упевненості в працездатності сценаріїв на будь-якому браузері потрібний механізм, що дозволяє відмовитися від використання cookies при управлінні сесіями. Такий механізм дійсно існує в PHP, і основна його ідея полягає в тому, щоб передавати ідентифікатор сесії не в cookies, а яким-небудь аналогічним шляхом, наприклад, в даних запиту get.

Явне використання константи SID

У PHP існує одна спеціальна константа з ім'ям SID. Вона завжди містить ім'я групи поточної сесії і її ідентифікатор у форматі імя=ідентифікатор.

Саме у такому форматі дані приймаються, коли вони приходять з cookies браузера. Таким чином, достатньо передати значення константи SID в сценарій, щоб він "подумав", ніби то дані прийшли з cookies. Приклад - в лістингу 10.2.

Лістинг 10.2.

```
<?php ## Простий приклад використання сесій без Cookies.
session_name("test");
session_start();
$_SESSION['count'] = @$_SESSION['count'] + 1;
?>
<h2>Лічильник</h2>
У поточній сесії роботи з браузером Ви відкрили цю сторінку
<?=$_SESSION['count']?> раз(а). <br>
Закрийте браузер, щоб обнулити цей лічильник.<br>
<a href="<?=$_SERVER['SCRIPT_NAME']?>?<?=SID?>">
Натисніть сюди для оновлення сторінки!</a>
```

Якщо набрати в браузері адресу на зразок такої: http://exaraple.cora/src/session/sidget.php то створиться нова сесія з унікальним ідентифікатором. Зрозуміло, якщо відразу ж натиснути кнопку "Обновити", лічильник не збільшиться, тому що при кожному запуску створюватиметься нове тимчасове сховище - у PHP просто немає інформації про ідентифікатор користувача. Передостанній рядок лістингу 10.2 передає в сценарій, що запускається через гіперпосилання, дані про ідентифікатор поточної сесії, тепер з його точки зору вони нібито пришли з cookies... Нажавши на останнє посилання, видно, що лічильник почне збільшився. (Цей приклад буде працювати тільки в тому випадку, якщо в браузері дійсно відключені cookies).

Ідентифікатор сесії і ім'я групи

На одному і тому ж сайті можуть співіснувати відразу декілька сценаріїв, які потребують послуг підтримки сесій PHP. Вони "нічого не знають" один про

одного, тому тимчасові сховища для сесій повинні вибиратися не тільки на основі ідентифікатора користувача, але і на основі того, який з сценаріїв запитав обслуговування сесії.

Ім'я групи сесій

Нехай розробник А написав сценарій лічильника, приведений в лістингу 10.1. Він використовує елемент $_SESSION з ключем count і не має жодних проблем. До тих пір, поки розробник В, нічого що не знає про сценарій А, створив систему статистики, яка теж працює з сесіями. Найжахливіше, що він також реєструє ключ count, не знаючи про те, що той вже "зайнятий". В результаті, як завжди, страждає користувач: запустивши спочатку сценарій розробника В, а потім - А, він бачить, що дані лічильників перемішалися.

Потрібно якось розмежувати сесії, що належать одному сценарію, від сесій, що належать іншому. У PHP передбачено такий стан речей. Можнао давати групам сесій унікальні імена, і сценарій, що знає ім'я своєї групи, зможе дістати до неї доступ. У цьому випадку розробники А і В можуть захистити свої сценарії від проблем з перетинами імен змінних. Досить в першій програмі вказати PHP, що потрібно використовувати групу з ім'ям, скажімо, sesA, а в другій - sesB.

```
string session_name([string $newname])
```

Ця функція встановлює або повертає ім'я групи сесії, яка використовуватиметься PHP для зберігання зареєстрованих змінних. Якщо параметр $newname не заданий, то просто повертається поточне ім'я, а його зміни не відбувається. Якщо ж цей параметр вказаний, то ім'я групи буде змінено на $newname, при цьому функція поверне попереднє ім'я.

Функція session_name() лише змінює ім'я поточної групи і сесії, але не створює нову сесію і тимчасове сховище. Це означає, що потрібно у більшості випадків викликати session_name(імя_групи) ще до її ініціалізації - виклику session_start().

Якщо функція session_name() не була викликана до ініціалізації, PHP використовуватиме ім'я за умовчанням - PHPSESID.

Ідентифікатор сесії

Фактично ідентифікатор сесії (SID) є ім'ям тимчасового сховища, яке буде використано для запам'ятовування даних сесії між запусками сценарію. Отже, один SID - одне сховище. Немає SID, немає і сховища, і навпаки.

Ім'я групи сесії - це всього лише збірна назва для декількох сесій (тобто для багатьох SID), запущених різними користувачами. Один і той же клієнт ніколи не матиме два різних SID в межах одного імені групи. Але його браузер цілком може працювати (і часто працює) з декількома SID, розташованими логічно в різних "просторах імен".

Всі SID унікальні і однозначно визначають сесію на комп'ютері, що виконує сценарій, - незалежно від імені сесії. Ім'я ж задає "простір імен", в який будуть згруповані сесії, запущені різними користувачами. Один клієнт може мати відразу декілька активних просторів імен (тобто декілька імен груп сесій).

```
string session_id([string $sid])
```

Функція повертає поточний ідентифікатор сесії SID. Якщо заданий параметр $sid, то у активної сесії змінюється ідентифікатор на $sid.

Фактично, викликавши session_id() до session_start(), можна підключитись до будь-якої (у тому числі і до "чужої") сесії на сервері, якщо відомий її ідентифікатор.

Також можна створити сесію з потрібним ідентифікатором, при цьому автоматично встановивши його в cookies користувача.

Шлях до тимчасового каталога

Замість того щоб возитися з групами сесій, можна встановити інший шлях до тимчасового каталога, в який PHP "складає" файли-сховища сесій. Це дасть

такий самий ефект: сесії, використовувані однією групою скриптів, не перетинатимуться зі всіма іншими.

```
string session_save_path([string $path])
```

Функція повертає ім'я каталога, в якому поміщатимуться файли - тимчасові сховища даних сесії. У випадку якщо вказаний параметр, активне ім'я каталога буде переустановлено на $path. При цьому функція поверне попередній каталог.

Чи варто змінювати групу сесій?

Розділяти всі сесії на групи іноді виявляється не кращою ідеєю. Дійсно, якщо авторизація на сайті відбувається за допомогою механізму сесії, зміна імені групи автоматично її анулює. Припустимо, на сайті є скрипт auth.php, що виводить форму для введення імені користувача і пароля. Якщо дані допустимі, сценарій записує з сесії ознаку успішної авторизації: $_SESSION['isAuthorized'] = true. Надалі всі скрипти аналізують значення цього елементу і, якщо воно істинне, надають додаткові функції користувачеві (наприклад, виводять посилання на приховані підрозділи сайту).

Тепер, що ім'я групи сесій змінилося після авторизації. Нова сесія, звичайно ж, створиться порожньою, а значить, в ній вже не буде елементу isAuthorized. Таким чином, в скриптах, що використовують інше ім'я групи, ніяк не зможемо дістати доступ до ознаки авторизації.

Тому в більшості ситуацій змінювати ім'я групи сесій взагалі не рекомендується. Натомість краще зберігати дані не у всьому масиві $_session, а в деякому його підмасиві. Наприклад, система управління форумом може працювати тільки з елементом $_SESSION['ForumSubsystem'], а система авторизації - з $_SESSION['AuthSubsystem']. В сесії можна зберігати дані будь-якої складності, так що вказані елементи цілком можуть бути асоціативними масивами. Робота з ними організовується так:

```
// Створюємо синонім для елементу $_SESSION['ForumSubsystem'],
// щоб мати до нього зручніший доступ.
$ForumSession =&$_SESSION['ForumSubsystem'];
```

```
//В дійсності працюємо з $_SESSION['ForumSubsystem1]['count'].
$ForumSession['count'] = @$ForumSession['count'] + 1;
// Те ж саме, але для підсистеми авторизації.
$AuthSession =&$_SESSION['AuthSubsystera1];
$AuthSession['count'] = $AuthSession['count'] + 1;
$AuthSession['isAuthorized1] = true;
```

Отже, форум працює тільки в межах $_SESSION['ForumSubsystem'], а підсистема авторизації - в межах $_SESSION['AuthSubsystem']. Як видно, перетину даних сесії не відбувається - в обох випадках використовуються елементи з одним і тим же ім'ям count, але це різні елементи.

Робота з cookies

Всі cookies, що прийшли скрипту, потрапляють в масив $_COOKIES. Для ілюстрації розглянемо скрипт, який вважає, скільки разів його запустив поточний користувач (лістинг 10.3).

Лістинг 10.3.

```
<?php ## Демонстрація роботи з $_COOKIES.
// Спочатку лічильник рівний нулю.
$count = 0;
// Якщо в Cookies щось є, беремо лічильник звідти.
if (isset($_COOKIE['count'])) $count = $_COOKIE['count'];
$count++;
// Записуємо в Cookies нове значення лічильника.
setcookie("count", $count, 0x7FFFFFFF, "/");
// Виводимо лічильник.
echo $count;
?>
```

Функція setcookie() посилає в браузер користувача cookie з вказаним ім'ям і значенням.

```
int setcookie (string name [, string value [, int expire [,
string path [, string domain [, int secure]]]]])
```

Визначає Cookie для відправлення разом з іншою header-інформацією. Cookie повинні бути відправлені до любих інших шапок/headers (це обмеження

Cookie, а не PHP). Тобто потрібно розміщали виклики цієї функції перед тегами <html> або <head>.

Всі аргументи, крім name, є необов'язковими. Якщо є тільки аргумент name, Cookie з цим іменем буде видалена з віддаленого клієнта. Можна також замінити довільний аргумент пустим рядком (""), чтоб пропустити цей аргумент. Аргументи expire і secure це цілі числа/integer і вони не можуть бути пропущені за допомогою пустого рядка. В них потрібно використовувати ноль (0). Аргумент expire це звичайне Unix time integer, яке повертається функціями time() або mktime(). Аргумент secure вказує на те, що даний Cookie повинен передаватись тільки через HTTPS-з'єднання.

Після того як Cookie встановлені, доступ до них може бути отриманий при завантаженні наступної сторінки через масив $_COOKIE.

Звичайні пастки:

- Cookie будуть невидимі до тих пір, поки не буде завантажена наступна сторінка.
- Cookie повинні бути видалені з тими ж параметрами, з якими були встановлені.

У PHP 3 багато викликів setcookie() в тому ж скрипті можуть бути виконані у реверсному порядку. Якщо потрібно видалити один Cookie до встановлення іншого, потрібно зробити встановлення до видалення. У PHP 4 багато викликів setcookie() виконуються в порядку виклику.

Приклади відправки Cookie:

Приклад 1. Відправка Cookie функцією setcookie()

```
setcookie ("TestCookie", $value);
setcookie ("TestCookie", $value,time()+3600);/* період дії - 1
година*/
setcookie ("TestCookie", $value,time()+3600, "/~rasmus/",
".utoronto.ca", 1);
```

При видаленні Cookie потрібно переконатися, що дата закінчення дії пройшла, щоб перемкнути механізм у браузері. Далі йдуть приклади видалення Cookie, створеної в попередньому прикладі:

Приклад 2. Видалення Cookie з допомогою setcookie()

```
// встановити дату закінчення дії на одну годину назад
setcookie ("TestCookie", "", time() - 3600);
setcookie ("TestCookie", "", time() - 3600, "/~rasmus/",
".utoronto.ca", 1);
```

Варто звернути увагу, що частина value Cookie буде автоматично urlencoded при відправленні Cookie, і, коли вона отримана, то автоматично декодується і присвоюється змінній з тим же ім'ям, що й ім'я Cookie. Для перегляду вмісту цієї Cookie у скрипті просто використовується один з наступних прикладів:

```
echo $TestCookie;
echo $_COOKIE["TestCookie"];
```

Можна також встановити Cookie масиву, використовуючи нотацію в імені Cookie. Це дає ефект установки стількох кук, скільки елементів у цьому масиві, але, коли Cookie приходить у скрипт, значення поміщаються в масив з ім'ям куки:

```
setcookie("cookie[three]", "cookiethree");
setcookie("cookie[two]", "cookietwo");
setcookie("cookie[one]", "cookieone");
if (isset($cookie)) {
  while (list($name, $value) = each($cookie)) {
    echo "$name == $value<br>\n";
  }
}
```

Завдання

0. Реалізувати авторизацію на сайті.
1. Забезпечити для відвідувача сайту можливість вибору кольорової схеми сайту, та встановлювати її автоматично при повторному заході на сайт.
2. Забезпечити для відвідувача сайту можливість вибору шрифтового оформлення, та встановлювати його автоматично при повторному заході на сайт.

3. Реалізувати підрахунок відвідувачів, відвідувачів розрізняти по встановлених cookie.
4. Забезпечити для відвідувача сайту змінювати розстановку блоків на сторінці та встановлювати її автоматично при повторному заході на сайт.
5. Запам’ятовувати сторінки які відвідував користувач сайту та пропонувати ці сторінки при повторному заході.
6. Встановити cookie на 1 годину.
7. Встановити cookie до закриття браузера.
8. Запам’ятати час, дату та IP-адрес останнього відвідування та вивести цю інформацію при наступному відвідуванні сайту.
9. Реалізувати повторну автоматичну авторизацію.
10. Реалізувати режим відлагодження для скрипта (увімкнути вивід помилок, а також виводити додаткову інформацію під час виконання скпита), якщо встановлений cookie – debug.
11. Створити кілька сторінок із описом товару та кнопкою «добавити у корзину». Реалізувати можливість перегляду корзини. Завдання реалізувати за допомогою сесій.
12. Запам’ятовувати часті відвідування користувачем сторінок і пропонувати їх на головній сторінці.
13. Забезпечити для відвідувача сайту можливість вибору розміру тексту і кількості колонок, та встановлювати їх автоматично при повторному заході на сайт.
14. Встановити cookie тільки для шляху `/path/`.
15. Повести опитування користувача задавши кілька запитань, проміжні відповіді зберігати у сесії, вивести результати опитування після того, як користувач відповість на усі запитання.

Хід роботи

1. Створити новий файл у текстовому редакторі (Dreamweaver, Notepad або іншому).
2. Написати скрипт для авторизації та скрипт для перевірки.
3. Зберегти програму на диск у файли з іменами auth.php та page.php у папку <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB10
4. Перевірити виконання скрипта у браузері.
5. При виникненні помилки – відкоригувати програму та виконати її.
6. Оформити звіт по виконаній роботі.

Методичні рекомендації

```
<?php
/* auth.php */
```

```
if (($_POST['user'] == 'admin') && ($_POST['password'] == '123'))
{
  session_start();
  $_SESSION['user'] = 'admin';
  $_SESSION['username'] = 'Administator';
}
Header('Location: page.php');
?>

<?php
/* page.php */
session_start();
if (isset($_SESSION['username'])) {
  echo "Hello {$_SESSION['username']}!";
} else { ?>
<form action="auth.php" method="post">
  Логін: <input name="user"><br/>
  Пароль: <input type="password" name="password"><br/>
  <input type="submit" value="Увійти">
</form>
<?php
}
?>
```

Для перевірки роботи програми потрібно у браузері ввести <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB10\page.php. Ввести логін: admin, пароль: 123 і натиснути кнопку "Увійти". Побачимо привітання. Потім введемо ще раз у браузері <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB10\page.php і переконаймось що і далі є привітання.

* Де саме зберігаються пари логін:пароль і чи у відкритому форматі чи якось закодовані відносяться до безпеки та зручності використання. На оцінювання завдання вибаний спосіб перевірки логіна та пароля не впливає.

Контрольні запитання

1. Що таке сесія?
2. Де зберігаються сесії?
3. Як видалити сесію?
4. Що таке cookie?
5. Де зберігаються cookie?

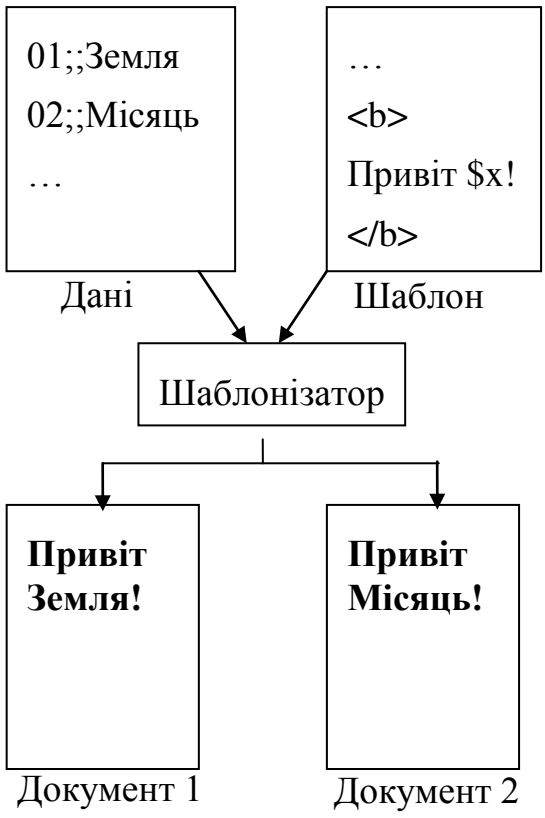
Тема 11: Шаблонізатор

Мета роботи: навчитись створювати та використовувати шаблонізатор для розділення PHP-коду та HTML.

Теоретичні відомості

Шаблонізатор - це програмне забезпечення, що дозволяє використовувати HTML-шаблони для генерації кінцевих HTML-сторінок. Основна мета використання шаблонізаторів - це відділення подання даних від виконуваного коду. Часто це необхідно для забезпечення можливості паралельної роботи програміста і дизайнера-верстальника. Використання шаблонізаторів часто покращує читання коду та внесення змін у зовнішній вигляд, коли проект повністю виконує одна людина.

Використання шаблонізаторів часто ототожнюють з парадигмою MVC. Ці поняття пов'язані, але не тотожні.



Веб шаблон є інструментом для відділення вмісту від візуального представлення у веб-дизайні, для масового створення веб-документів.

Веб-шаблони можуть бути використані для створення сайтів будь-якого типу.

У найпростішому своєму сенсі, веб-шаблон, функціонує аналогічно документу на бланку для використання при створенні веб-сайту.

Рис. 11.1. Схема роботи шаблонізатора

Ефективний поділ

Загальна мета досвідчених веб-розробників - розробка та розгортання гнучких додатків, які легко супроводжувати.

Важливим моментом в досягненні цієї мети є відділення бізнес-логіки від логіки представлення. Розробники можуть використовувати системи веб-шаблонів (з різним ступенем успіху), щоб зберегти такий поділ.

Однією з труднощів при виконанні такого поділу є відсутність чітко визначених критеріїв для оцінки того, що воно дійсно виконано, і наскільки добре це зроблено. Є, однак, досить стандартні евристики, які були запозичені з області програмного забезпечення.

Гнучкість подання

Одним з основних обґрунтувань "ефективного поділу" є необхідність забезпечення максимальної гнучкості коду і ресурсів, що описують логіку представлення. Вимоги клієнта, зміна споживчих переваг або бажання "освіжити обличчя" зі збереженням раніше існуючого змісту часто призводять до необхідності істотно змінити дизайн веб-контенту, по можливості, без порушення основної інфраструктури сайту.

Різниця між "поданням" (дизайном) та "бізнес-логікою" (інфраструктурою), як правило, має важливе значення, тому що:

- Початкова мова коду представлення може відрізнятися від мови коду інших ресурсів;
- У ході виробничого процесу може знадобитися робота по внесенню змін в різний час і в різних місцях;
- Різні працівники володіють різними навичками: дизайнерські вміння не завжди збігаються з навичками кодування бізнес-логіки;
- Коли розрізнені компоненти зберігаються окремо і слабозв’язані, код легше підтримувати, він стає більш зручним для читання;

Блоки шаблону

У шаблоні допускається крім HTML-коду використовувати деякі спеціальні позначення, які саме, правила їх застосування і як вони будуть поводитись задають розробники шаблонізатора. Проте можна виділити деякі типи блоків, які зустрічаються у більшості з них.

Змінні

Здебільшого змінні у шаблонах обрамляються якимось спецсимволами, наприклад: {ЗМІННА}. При опрацюванні шаблону шаблонізатор знайде таку конструкцію та замінить на підготовлені дані з ім'ям ЗМІННА.

Умовний блок

Використовується для того щоб відобразити той чи інший HTML-код взалежності від певної умови. Наприклад може бути така конструкція:

```
<!-- IF УМОВА (ЗМІННА1 [<логічна операція> ЗМІННА2 ...]) -->
Якийсь текст
<!-- ELSE -->
Інший текст
<!-- ENDIF -->
```

Шаблонізатор знайшовши умовний блок аналізує умову і якщо умова істинна то у результуючій веб-сторінці буде тільки "Якийсь текст", якщо ж умова хибна - "Інший текст".

Приклад використання:

```
<!-- IF LOGIN_USER -->
  Привіт {LOGIN_USER}!
<!-- ELSE -->
  <form ...>
    Логін: <input name=login><br/>
    Пароль: <input name=psw type=password><br/>
    <input type=submit value="Увійти">
  </form>
<!-- ENDIF -->
```

Циклічний блок

Використовується для виводу набору даних, наприклад переліку новин. Можна використовувати наступну конструкцію:

```
<!-- BEGIN назва_блоку -->
…
назва_блоку.НАЗВА_ЗМІННОЇ
…
<!-- END назва_блоку -->
```

Тут НАЗВА_ЗМІННОЇ це змінна яка доступна в контексті конкретного блоку, а її значення при кожній ітерації циклу-обробнику такого блоку буде різною.

Приклад використання:

```
<ul>
<!-- BEGIN news -->
<li><b>news.TITLE</b><br/>
news.TEXT</li>
<!-- END news -->
</ul>
```

Блок вкладення

Коли об’єм шаблону достатньо великий доводиться його розбивати на кілька файлів. І для того щоб вказати, що в певному місці одного шаблону потрібно вкласти інший, можна використовувати таку конструкцію:

```
<!-- INCLUDE ім’я_файлу_шаблону -->
```

Завдання

Створити шаблонізатор, який опрацьовує шаблон та на основі вхідних даних обробляє у шаблоні змінні, умовні блоки, циклічні блоки, вкладення іншого шаблону та повертає сформований html-файл.

0. змінні – {ЗМІННА}, включення іншого шаблону – <!-- include(ім’я файлу шаблону) -->
1. змінні - %%ЗМІННА%%, та умовні блоки – <% if ЗМІННА %>якись HTML-код<% endif %>.

2. змінні - !ЗМІННА!, та циклічні блоки – {{ for ЗМІННА_МАСИВ as ЕЛЕМЕНТ_МАСИВУ }} !ЕЛЕМЕНТ_МАСИВУ! {{ endfor }}.
3. умовні блоки – [[if ЗМІННА]]якись HTML-код[[else]] інший HTML-код [[endif]].
4. циклічні блоки – [[for ЗМІННА_МАСИВ as ЕЛЕМЕНТ_МАСИВУ]] %% ЕЛЕМЕНТ_МАСИВУ.ключ1%%, %%ЕЛЕМЕНТ_МАСИВУ.ключ2%% {{ endfor }}.

   * решта варіантів формуються модифікацією символів, що обрамлюють змінні та блоки.

Хід роботи

1. Створити новий файл у текстовому редакторі (Dreamweaver, Notepad або іншому).
2. Написати шаблонізатор та зберегти у файл з іменем templ.php у папку <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB11
3. Створити шаблони для перевірки та зберегти у файли з іменами t1.html та t2.html у цю ж папку.
4. Створити файл для використання шаблонізатора і зберегти у файл з іменем test.php у цю ж папку.
5. При виникненні помилки – відкоригувати програму та виконати її.
6. Оформити звіт по виконаній роботі.

Методичні рекомендації

templ.php:

```
<?php

function replaceVariables($pockets){
  if (isset($GLOBALS['TEMPL_VARS'][$pockets[2]])) return
$GLOBALS['TEMPL_VARS'][$pockets[2]]; else return '';
}

function replaceInclude ($pockets){
 return doTempl($pockets[2]);
}

function doTempl($fn){
```

```
  $text = file_get_contents($fn);
  if ($text) {
    $text = preg_replace_callback('/(\{([A-Z]+)\})/',
"replaceVariables", $text);
    $text = preg_replace_callback('/(<!-- include\(([a-z0-9_-
]+\.html)\) -->)/', "replaceInclude", $text);
  }
  return $text;
}
?>
```

t1.html:

```
<html>
<head>
<title>{TITLE}</title>
</head>
<body>
{CONTENT}
<!-- include(t2.html) -->
</body>
</html>
```

t2.html:

```
<div id="footer">
Developed by <b>{DEV}</b>
</div>
```

test.html:

```
<?php
include('templ.php');
$GLOBALS['TEMPL_VARS']['TITLE'] = 'Some title';
$GLOBALS['TEMPL_VARS']['CONTENT'] = 'Some content';
$GLOBALS['TEMPL_VARS']['DEV'] = 'Some developer';
echo doTempl('t1.html')
?>
```

Для перевірки роботи програми потрібно у браузері ввести <домашній каталог веб-серверу>\<назва групи>\<прізвище студента>\LAB11\test.php.

Контрольні запитання

1. Що таке шаблонізатор?
2. Що таке шаблон?
3. Які шаблонізатори Вам відомі?
4. Для чого використовуються шаблони?
5. Яка схема роботи при використанні шаблонів?

Література

1. Люк Веллинг, Лора Томсон. Разработка веб-приложений с помощью PHP и MySQL. 4-е издание – СПб.: Вильямс, 2013. – 848с.
2. Котеров Д. В., Костарев А. Ф. PHP 5 В Подлиннике – СПб.: БХВ-Петербург, 2006. – 1120с.
3. Эд Леки-Томпсон, Алек Коув, Стивен Новицки, Хьяо Айде-Гудман. PHP 5 для профессионалов – СПб.: Диалектика, 2006. – 608с.
4. Швендимен Б.; PHP 4: Руководство разработчика; М.-СПб.: Вильямс, 2002. – 688с.
5. Глушаков С.В., и др.; Программирование Web-страниц; Х.: Фолио, 2005. – 390с.
6. Марти Холл, Лэри Браун; Программирование для Web; М.-СПб.-К: Вильямс, 2001. – 1264с.

Додаток 1. Зразок оформлення звіту

ЗВІТ

про виконання лабораторної роботи № 1

“Лінійні програми та розгалуження”

з дисципліни “Веб-технології ”

студента групи КН-25

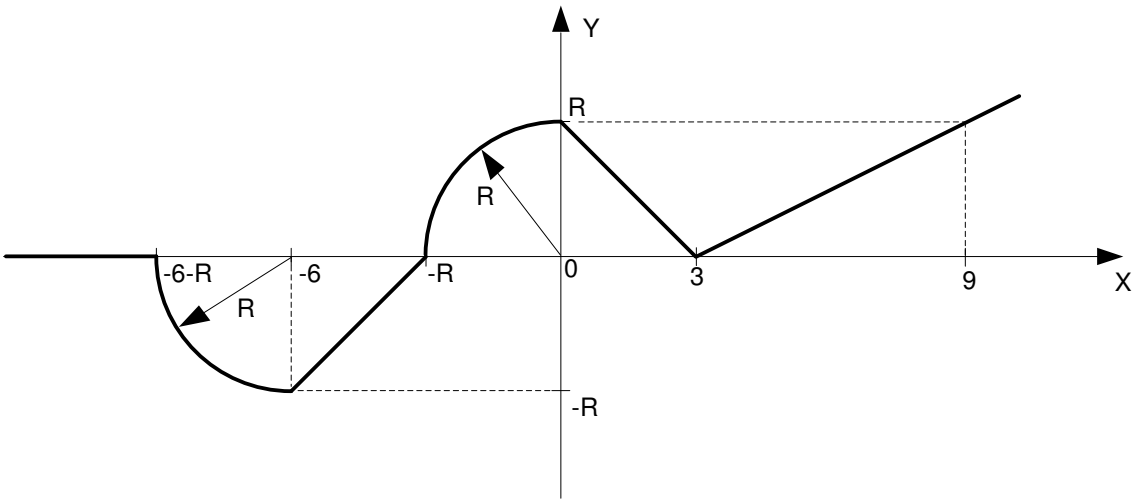
Іванова І.І.

Завдання 1: Напишіть програму для розрахунку за двома формулами (результати обчислень за двома формулами повинні співпадати). Введення значень для змінних вводиться користувачем

$$z_1 = 2\sin^2(3\pi - 2\alpha) \times \cos^2(5\pi + 2\alpha);$$

$$z_2 = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8\alpha\right).$$

Завдання 2: Написати програму, яка за введеним значенням аргумента обчислює значення функції, заданої у вигляді графіка. Параметр R вводиться користувачем.



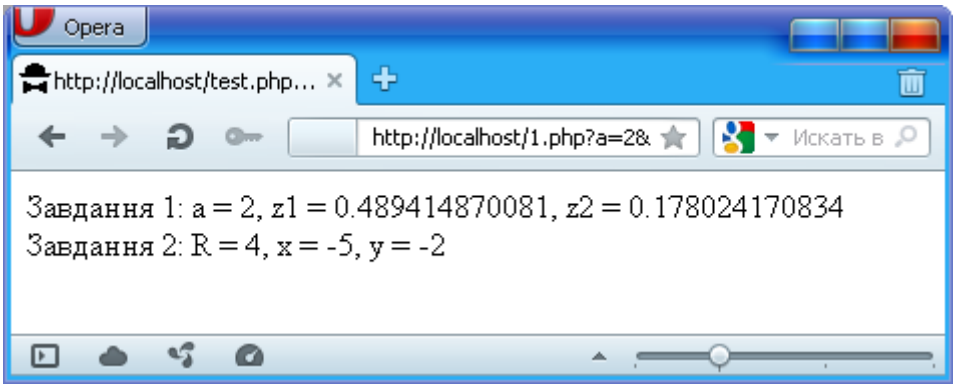
Код програми:

```
<?php
/* Завдання 1 */
  $a = $_GET['a'];
```

```
  $z1 = 2*pow(sin(3*PI-2*$a),2)*pow(cos(5*PI+2*$a),2);
  $z2 = 1/4-1/4*sin(5/2*PI-8*$a);
    print("Завдання 1: a = $a, z1 = $z1, z2 = $z2<br/>");

/* Завдання 2 */
  $x = $_GET['x'];
  $R = $_GET['r'];
  if ($x < -6-$R) $y = 0;
    if ($x < -6) $y = -sqrt($R*$R-($x+6)*($x+6));
      elseif ($x < -$R) $y = ($x+6)*$R/(6-$R)-$R;
        elseif ($x < 0) $y = sqrt($R*$R-$x*$x);
          elseif ($x < 3) $y = ($x-$R)/3+$R;
            else $y = ($x-3)*$R/6;
    print("Завдання 2: R = $R, x = $x, y = $y");
?>
```

Результат виконання програми для параметрів: a=2, R=4, x=-5.



Висновок: на цій лабораторній роботі я навчився створювати php-сценарій з лінійними виразами та розгалуженнями.