МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Практична робота № 4**

з дисципліни « Web-програмування мовою PHP »

*назва дисципліни*

на тему: «ВИКОРИСТАННЯ РЕГУЛЯРНИХ ВИРАЗІВ. ЗАСТОСУВАННЯ HTTP COOKIES І МЕХАНІЗМУ СЕСІЙ»

Виконав: студент 3 курсу групи № 632п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Зайченко Ярослав Ігорович

(прізвище й ініціали студента)

Прийняв: старший викладач

Дем'яненко Владислав Анатолійович

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2024

**ЗМІСТ**

[Постановка завдання 3](#_Toc179963230)

[Порядок виконання роботи 3](#_Toc179963231)

[Зміст звіту 3](#_Toc179963232)

[Теоретичні відомомості 4](#_Toc179963233)

[Виконання роботи 8](#_Toc179963234)

[Заключення 12](#_Toc179963235)

[*Додаток А* 13](#_Toc179963236)

[*Додаток Б* 14](#_Toc179963237)

[*Додаток В* 15](#_Toc179963238)

Мета роботи: вивчення можливостей обробки текстових даних із застосуванням регулярних виразів, а також способів передачі даних між скриптами за допомогою HTTP cookies та механізму сесій.

Постановка завдання

Розробити і реалізувати на мові PHP серверний сценарій (скрипт) для обробки запитів користувача та подання результатів у вигляді генерованого документа HTML.

Завдання варіанту 3:

"Гостьова книга"

Видалити всі теги розмітки HTML з тіла повідомлення. Перевірити адресу електронної пошти на коректність за допомогою регулярного виразу. У разі неправильної адреси виводити попередження (повідомлення до БД не вносити, а заново виводити у формі для додавання). Після 3 невдалих спроб введення (для рахунку спроб використовувати сесії) встановити значення cookie, за наявності якого блокується форма для додавання (або як варіант - виводиться попередження про перевищення ліміту спроб введення).

Вихідна форма документа повинна містити заголовок з назвою роботи, внизу документа – ПІБ виконавця.

Сервер запускається/зупиняється файлами run.exe/stop.exe з па- пки /webservers/etc/ , яка знаходиться у робочій директорії.

Файли ЛР повинні бути розміщені (при запущеному сервері) у папці

*z:/home/localhost/www/прізвище\_виконавця/*

та будуть відповідно доступні за адресою

*http://localhost/прізвище\_виконавця/ім'я\_скрипту.php*

Порядок виконання роботи

1. Отримати у викладача індивідуальне завдання для виконання роботи.
2. Вивчити теоретичний матеріал.
3. Визначити завдання, які має вирішувати сценарій, що розробляється.
4. Розробити текстовий інтерфейс користувача відповідно до завдання, а також кінцевий вид документа, що генерується, з результатами роботи сценарію.
5. Реалізувати мовою PHP спроектований сценарій.
6. Протестувати локально розроблений сценарій.
7. Зробити висновки щодо роботи.

Зміст звіту

1. Постановка задачі. Опис завдань, які вирішуються серверним сценарієм.
2. Короткий опис алгоритму сценарію.
3. Опис використовуваних вхідних даних та функцій.
4. Лістинг вихідного коду сценарію з коментарями, а також результати його роботи (скриншот або текстове подання).
5. Висновки щодо роботи.

Теоретичні відомомості

Регулярний вираз це патерн, який порівнюється з рядком-суб'єктом зліва направо. Більшість символів у патерні представляють себе і збігаються з відповідними символами в рядку-суб'єкті. Як тривіальний приклад - патерн The quick brown fox збігається з частиною рядка-суб'єкта, який повністю ідентичний йому.

Метасимволи

Потужність регулярним виразам надає можливість включати до патерну альтернативи та повторення. Вони кодуються в патерне метасимволами, які не уявляють себе, а інтерпретуються особливим чином.

Є два різних набори метасимволів: патерна, що розпізнаються в будь- якому місці, крім квадратних дужок, і ті, які розпізнаються в квадратних дужках.

* \ - загальний escape-символ, використовується для екранування спеціальних символів.
* ^ - позначає початок виразу або заперечення класу символів (якщо використовується всередині []).
* $ - позначає кінець виразу.
* . - збігається з будь-яким символом, окрім символу нового рядка (за замовчуванням).
* [ - початок визначення класу символів.
* ] - кінець визначення класу символів.
* | - позначає альтернативу (логічне "або").
* ( - початок субпатерну (групування виразу).
* ) - кінець субпатерну.
* ? - квантифікатор "0 або 1" або мінімізатор інших квантифікаторів.
* - квантифікатор "0 або більше".
* + - квантифікатор "1 або більше".
* { - початок визначення min/max кількості повторів у квантифікаторі.
* } - кінець визначення min/max кількості повторів у квантифікаторі.
  + - у класі символів позначає діапазон символів (наприклад, [a-z]), але якщо використовується першим чи останнім символом у класі, сприймається як звичайний символ.

Backslash/зворотний слеш

Символ backslash використовується по-різному. По-перше, якщо після нього йде неалфавітний символ, він скасовує будь-яке спеціальне значення, яке може мати символ. Таке використання зворотного слеша як escape-символу застосовується як усередині, і поза класів символів.

Наприклад, якщо ви хочете знайти збіг із символом "\*", ви записуєте в патерні "\\*". Це буде працювати незалежно від того, чи може наступ ний символ інтерпретуватися як метасимвол, тому завжди надійніше записувати неалфавітний символ з "\", щоб специфікувати, що він уяв ляє себе. Особливо якщо ви хочете знайти збіг з backslash тоді ви записуєте "\\".

Якщо патерн компілюється із опцієюPCRE\_EXTENDED, то пробіли в патерні (крім пробілів у класі символів) і символи між "#" поза класом символів і наступним символом newline ігноруються.

По-друге, backslash надає спосіб кодування в патерні недрукованих символів видимим чином. Обмежень на появу недрукованих символів немає, за винятком двійкового нуля, який закінчує патерн, але якщо патерн готується шляхом редагування тексту, то зазвичай легше вико ристовувати одну з наступних escape-послідовностей (замін), а не бінарний символ.

* \a - alarm/попередження, тобто символ BEL (hex 07).
* \cx - "Control-x", де x — це будь-який символ.
* \e - Escape (hex 1B).
* \f - Form feed/переведення сторінки (hex 0C).
* \n - Newline/новий рядок (hex 0A).
* \r - Carriage return/повернення каретки (hex 0D).
* \t - Tab/табуляція (hex 09).
* \xhh - Символ з 16-річним кодом `hh`.
* \ddd - Символ з 8-річним кодом `ddd`, або backreference/зворотне посилання.

Ефект від застосування \cx такий: якщо "x" це символ у нижньому регістрі, він конвертується у верхній регістр. Потім біт символу 6 (hex 40) інвертується. Таким чином, \cz стає hex 1A, \c{ стає hex 3B, а \c; ставати hex 7B.

Після "x" читаються не більше двох 16-річних цифр (букви можуть бути в будь-якому регістрі).

Після "\0" читаються не більше чотирьох 8-річних цифр. В обох випадках, якщо є менше двох цифр, використовуються ті, які представлені. Таким чином, послідовність "\0\x\07" специфікує два бінарних нулі з наступним символом BEL. Переконайтеся, що ви надали дві цифри після початкового нуля, якщо наступний символ є 8-річним числом.

Обробка backslash з наступними цифрами, відмінними від 0, складніша. Поза класом символів, PCRE читає його та будь-які наступні символи як 10-річне число. Якщо число менше 10 або якщо у виразі є щонайменше стільки захоплюючих лівих дужок, вся послідовність вважається back reference\зворотним посиланням. Опис того, як цей механізм працює, буде дано пізніше у дискусії про субпатерни у дужках.

Усередині класу символів, або якщо 10-річне число більше 9 і немає такої кількості захоплюючих субпатернів, PCRE зчитує до трьох 8-річних цифр, що йдуть після backslash, і генерує один байт з останніх значних 8 бітів цього значення. Будь-які наступні цифри представляють себе. Наприклад:

* \040 - інший спосіб запису символу пробілу (space).
* \40 - те саме, за умови, що є менше 40 попередніх захоплюючих субпатернів.
* \7 - це завжди back reference/зворотне посилання.
* \11 - може бути back reference або інший спосіб запису символу табуляції (tab).
* \011 - це завжди символ табуляції (tab).
* \0113 - це символ табуляції (tab) з наступним символом "3".
* \113 - це символ з 8-річним кодом 113 (оскільки не може бути більше 99 зворотних посилань).
* \377 - це байт, що складається з бітових "1".
* \81 - це back reference або бінарний нуль з наступними двома символами "8" та "1".

Зауважте, що 8-річні значення 100 або більше повинні не вводитися з провідним 0, оскільки читається не більше трьох 8-річних цифр.

Усі послідовності, що визначають однобайтне значення, можуть використовуватися як усередині, так і поза класами символів. Крім того, всередині класу символів послідовність "b" інтерпретується як символ backspace (hex 08). Поза класом символів вона має інше значення

Третій варіант використання backslash - специфікація загального типу символів:

* \d - це будь-яке 10-річне число.
* \D - це будь-який символ, який не є 10-річним числом.
* \s - це будь-який пробільний символ.
* \S - це будь-який непробільний символ.
* \w - це будь-який "word/словниковий" символ.
* \W - це будь-який "non-word/несловниковий" символ.

Кожна пара escape-послідовностей поділяє повний набір символів на два різні набори. Будь-який цей символ збігається з однією, і лише з однією парою.

Символ "word" це будь-яка літера, або цифра, або символ підкреслення, тобто будь-який символ, який може бути частиною "word" у Perl. Визначення літер та цифр контролюється таблицями символів PCRE і може змінюватись, якщо має місце підстановка з локальною специфікою (див. раніше "Підтримка локалізації"). Наприклад, при локалізації "fr" (French) використовуються деякі символи з кодами вище 128 для введення букв з акцентами і вони збігаються з \w.

Ці послідовності типів символів можуть з'являтися як усередині, так і за межами класів символів. Кожен із них збігається з одним символом відповідного типу. Якщо поточна точка збігу є кінцем рядка-суб'єкта, всі вони зазнають невдачі, оскільки немає символу для порівняння.

Четвертий варіант - використання backslash для деяких простих тверджень. Затвердження специфікує умову, яка має бути знайдена в певній точці під час встановлення, не використовуючи жодних символів з рядка-суб'єкта. Використання субпатернів для складніших тверджень розглядається далі. Твердження зі зворотними слешами це:

* \b - це межа слова.
* \B - це не межа слова.
* \A - це початок суб'єкта (не залежить від багаторядкового режиму).
* \Z - це кінець суб'єкта або символ newline в кінці (не залежить від багаторядкового режиму).
* \z - це кінець суб'єкта (не залежить від багаторядкового режиму).

Ці твердження не можуть з'являтися в класах символів (але зауважте, що "b" має інше значення, а саме символ backspace, усередині класу символів).

Кордон слова це така позиція в рядку-суб'єкті, де поточний і попередній символи не збігаються з \w або \W (тобто один збігається з \w, а інший - з \W), або початок або кінець рядка, якщо перший або останній символ збігається з \w відповідно.

Твердження \A, \Z і \z відрізняються від традиційних circumflex і dollar (описано далі) тим, що вони збігаються тільки з початком і кінцем рядка суб'єкта, незалежно від встановлених опцій. На них не впливають опції PCRE\_NOTBOL або PCRE\_NOTEOL. Різниця між \Z та \z у тому, що \Z збігається до newline, тобто є останнім символом рядка, а також кінцем рядка, тоді як \z збігається лише наприкінці рядка.

Circumflex та dollar

Поза класом символів, у режимі підстановки за замовчуванням, символ circumflex (^) є твердженням, яке є true, тільки якщо поточна точка збігу є початком рядка-суб'єкта. Усередині класу символів circumflex має зовсім інше значення (див. далі).

Circumflex не повинен бути першим символом патерна, якщо використовуються кілька альтернатив, але повинен бути першим у кожній альтернативі, в якій з'являється, якщо патерн збігається з цією гілкою. Якщо всі наявні альтернативи починаються з circumflex (^), тобто якщо патерн обмежений для збігу тільки на початку суб'єкта, говориться, що це "заякорений/anchored" патерн. (Є також інші конструкції, які можуть викликати заякорювання патерна.)

Символ dollar є твердженням, яке TRUE, тільки якщо поточна точка збігу знаходиться в кінці рядка-суб'єкта або відразу після символу newline, який є останнім символом рядка (за замовчуванням). Dollar не повинен бути останнім символом патерна, якщо дано кілька альтернатив, але має бути останнім символом у будь-якій гілки, в якій він з'являється. Dollar не має спеціального значення у класі символів.

Значення dollar може бути змінено так, щоб він збігався тільки з кінцем рядка, через установку опції PCRE\_DOLLAR\_ENDONLY під час компіляції чи підстановки. Це не впливає на затвердження \Z.

Значення символів circumflex та dollar змінюється, якщо встановлена опція PCRE\_MULTILINE. В цьому випадку вони збігаються відразу після або відразу до внутрішнього символу \n, відповідно, на додаток до збігу на початку і в кінці рядка-суб'єкта. Наприклад, патерн /^abc$/ збігається з рядком-суб'єктом "def\nabc" у багаторядковому режимі, але не інакше. Відповідно, патерни, які заякорені в однорядковому режимі, якщо всі гілки починаються з "^", не є заякореними в багаторядковому режимі. Опція PCRE\_DOLLAR\_ENDONLY ігнорується, якщо встановлена PCRE\_MULTILINE.

Зауважте, що послідовності \A, \Z та \z можуть використовуватися для збігу з початком і кінцем суб'єкта в обох режимах, і якщо всі верви на початку патерну з \A завжди заякорені, незалежно від того, встановленаPCRE\_MULTILINEчи ні.

FULL STOP/ПОВНИЙ ЗУПИНОК

Поза класом символів точка в патерні збігається з одним із символів суб'єкта, включаючи символ, що не друкується, але не з символом (за замовчуванням) newline. Якщо встановлено опцію PCRE\_DOTALL, точки збігаються також із символами newline. Обробка точки залежить від обробки circumflex і dollar, їх ріднить тільки те, що вони обидва вводять символи newline. Крапка не має спеціального значення у класі символів.

Квадратні дужки

Відкриваюча квадратна дужка вводить клас символів, що закінчується квадратною дужкою, що закриває. Квадратна дужка, що закриває, сама по собі не є спеціальним символом. Якщо квадратна дужка, що закриває, необхідна як член класу символів, вона повинна бути пер- шим символом даних класу (після початкового circumflex, якщо він є) або повинна замінюватися за допомогою backslash.

Клас символів збігається з одиночним символом рядка-суб'єкта; Сим- вол повинен бути з набору символів, визначеного в даному класі, якщо тільки першим символом у класі не є circumflex - у цьому випадку символ рядка-суб'єкта повинен не входити в набір, визначений у да- ному класі. Якщо circumflex необхідний як член класу, переконайтеся, що він не є першим символом, або escape-ируйте його за допомогою зворотного слеша (\).

Наприклад, клас символів [aeiou] збігається з будь-якою голосною бук- вою в нижньому регістрі, а [^aeiou] збігається з будь-яким символом, що не є голосною буквою в нижньому регістрі.Зверніть увагу, що circumflex це просто зручний символ для позначення символів, які не використовуються для збіги. Він не є твердженням: він просто викори- стовує символи з рядка-суб'єкта і зазнає невдачі, якщо покажчик зна- ходиться наприкінці рядка.

Якщо встановлено збіг без урахування регістру, то будь-яка буква класу представляє символи як нижнього, так і верхнього регістрів, тому, наприклад, безреєстровий [aeiou] збігається з "A" і з "a", безреєс- тровий [^aeiou] збігається з не- "A" і з не-"a", в той час як реєстрова ве- рсія збігається.

Символ newline ніколи не розглядається спеціально у класах симво- лів, незалежно від того, встановлені опції PCRE\_DOTALL або PCRE\_MULTILINE чи ні. Клас, такий як [^a], зав- жди збігатиметься з newline.

Знак "мінус" (дефіс) можна використовувати для специфікації діапа- зону символів у класі символів. Наприклад, [dm] збігається з будь- якою літерою від d до m включно. Якщо знак мінус необхідний як член класу, він повинен вводитися заміною за допомогою backslash або з'являтися в позиції, де він не може інтерпретуватися як діапазон, що позначає, тобто. зазвичай, як перший або останній символ класу.

Неможливо мати літеральний символ "]" як кінець діапазону. Патерн [W-46] інтерпретується як клас із двох символів ("W" і "-") з наступним літеральним рядком "46]", тому він збігається з "W46]" або з "-46]". Од- нак, якщо "]" escape-ований зі зворотним слешем, він інтерпретується як кінець діапазону, тому [W-\]46] інтерпретується як єдиний клас, що містить діапазон, з наступними двома окремими символами. 8-річна або 16-річна вистава "]" також може використовуватися для вказівки на кінець діапазону.

Діапазони оперують у ASCII-кодуванні символів. Вони також можуть використовуватися для символів, специфікованих числами, наприклад [000-037]. Якщо діапазон літер встановлено при вимкненому розпізна- ванні регістру символів, він збігається з літерами у будь-якому регістрі. Наприклад, [Wc] еквівалентно [][\^\_`wxyzabc], збігаючись без ураху- вання регістру і при використанні таблиць символів для локалізації "fr", [xc8-xxb] збігається з акцентованими символами E в будь-якому регіс- трі.

Типи символів \d, \D, \s, \S, \w та \W також можуть з'являтися в класах символів і додавати до класу символи, з якими вони збігаються. На- приклад, [\dABCDEF] збігається з будь-яким 16-річним числом. Сим- вол circumflex (^) можна використовувати з типами символів верхнього регістру, щоб специфікувати більш обмежений набір символів - лише типи нижнього регістру. Наприклад, клас [^\W\_] збігається з будь-якою літерою чи цифрою, але не символом підкреслення.

Всі неалфавітні символи, крім \, -, ^ (на початку) та заключного ], є не- спеціальними в класах символів, але не зашкодить вводити їх за допо- могою escape-послідовностей.

Vertical bar/Вертикальна характеристика

Символи вертикальної риси (|) використовуються для поділу альтер- нативних патернів. Наприклад, патерн

gilbert | sullivan

збігається з "gilbert" або "sullivan". Можна вводити будь-яку кількість альтернатив, а також порожні альтернативи (збігаються з порожнім рядком). Процес підстановки відчуває альтернативи по черзі, зліва на- право, і використовується перший знайдений збіг. Якщо альтернативи знаходяться у субпатерні (визначено далі), "використання" означає збіг залишку головного патерну, а також альтернативи у субпатерні.

Встановлення внутрішніх опцій

Налаштування опцій PCRE\_CASELESS, PCRE\_MULTILINE, PCRE\_DOTALL і PCRE\_EXTENDED можуть бути змінені з патерна за допомогою послідовності літер- опцій Perl, укладених між символами "(?" та ")". Ці літери-опції такі:

i для PCRE\_CASELESS

m для PCRE\_MULTILINE

s для PCRE\_DOTALL

x для PCRE\_EXTENDED

Наприклад, (?im) встановлює багаторядковий пошук збігу без ураху- вання регістру. Можна також скасовувати установку опції, передуючи букві дефісом, і комбінувати установку та скасування опцій, як тут (?im-sx), де встановлюютьсяPCRE\_CASELESSіPCRE\_MULTILINEта скасовуютьсяPCRE\_DOTALLіPCRE\_EXTENDED. Якщо літера з'явля- ється як до, так і після дефісу, опція скасовується.

Область видимості цих змін опцій залежить від цього, де у патерне зміни з'являються. Для установок поза субпатерном (визначено далі), ефект буде таким же, як і при установці та скасуванні опцій на початку збігу. Наступні патерни поводяться абсолютно однаково:

(?i)abca(?i)bcab(?i)cabc(?i)

що, у свою чергу, те саме, що компіляція патерну abc з установ- коюPCRE\_CASELESS. Інакше висловлюючись, такі установки " верх- нього рівня " застосовуються до всього патерну (якщо відсутні інші зміни у субпатернах). Якщо є кілька параметрів однієї опції на верх- ньому рівні, використовується найправіша установка.

Якщо зміна опції виникає всередині субпатерну, ефект може бути різ- ним. Ця зміна поведінки була зроблена в Perl 5.005. Зміна опції в субпатерні впливає тільки на ту частину субпатерну, яка слідує за ним; так

(a(?i)b)c

збігається з abc і з aBC і більше ні з чим (припускаючи, що PCRE\_CASELESS не використовується). Це означає, що опції мо- жуть бути змінені для отримання різних установок у різних частинах патерну. Будь-які зміни, зроблені в одній альтернативі, впливають на наступні гілки всередині того самого субпатерну. Наприклад,

(a(?i)b|c)

збігається з "ab", "aB", "c" і "C", хоча при збігу з "C" перша гілка зали- шається до установки опції. Це відбувається тому, що установки опцій відбуваються на етапі компіляції. Інакше поведінка буде непередбачуваною.

PCRE-специфічні опції PCRE\_UNGREEDY і PCRE\_EXTRA можуть бути змінені так само, як і Perl-сумісні опції шляхом використання символів U і X відповідно. Установка прапора (?X) є спеціальної тому плані, що він повинен з'являтися в патерне раніше, ніж буде включена будь-яка додаткова можливість, навіть якщо вона на верхньому рівні. Найкраще поміщати його при старті.

Субпатерни

Субпатерни обмежені дужками (круглими), які можуть вкладатись. Ма- ркування частини патерну як субпатерну виконує дві дії:

1. Локалізує набір альтернатив. Наприклад, патерн cat(aract|erpillar|)збігається з одним із слів: "cat", "cataract" або "caterpillar". Без дужок він збігається з "cataract", "erpillar" або з порож- нім рядком.
2. Встановлює субпатерн як захоплюючий субпатерн (як визначено вище). Коли весь патерн збігається повністю, частина рядка-суб'єкта, що збіглася з субпатерном, передається назад, що викликає за допо- могою аргументу ovector функції pcre\_exec(). Відкриваючі дужки обчи- слюються ліворуч (починаючи з 1) для отримання кількості захоплюю- чих субпатернів.

Наприклад, якщо рядок "the red king" зіставляється з патерном ((red | white) (king | queen)) будуть захоплені підрядки "red king", "red" і "king", і вони будуть пронумеровані 1, 2 і 3.

Фактично таке виконання звичайними дужками двох функцій який зав- жди допомагає. Трапляється, коли необхідний угруповання субпатер- нів без необхідності захоплення. Якщо після відкритої дужки йде "?:", субпатерн не виконує захоплення і не враховується при підрахунку кі- лькості субпатернів, що захопили. Наприклад, якщо рядок "the white queen" зіставляється з патерном((?:red|white) (king|queen)), то будуть захоплені підрядки "white queen" і "queen", і вони будуть пронумеро- вані 1 і 2. Максимальна кількість захоплюваних підрядок - 99, а макси- мальна кількість всіх субпатернів, захоплюючих та не захоплюючих, дорівнює 200.

Як зручна абревіатура, якщо будь-які установки опцій потрібні на поча- тку незахоплюючого субпатерну, літери опцій можуть з'являтися між "?" та ":". Таким чином, два субпатерна (? i: saturday | sunday) (?: (? i) saturday | sunday) збігаються з одним і тим же набором рядків. Оскі- льки альтернативні гілки пробуються зліва направо, а опції не віднов- люють значення, поки не буде досягнуто кінця субпатерну, установка опцій в одній гілки не впливає на наступні гілки, і тому наведені вище патерни збігаються з "SUNDAY", а також з "Saturday".

Повторення

Повторення специфікується квантифікаторами, які можуть йти після будь-якого з наступних елементів:

одиночного символу, можливо, escape-ованогометасимвола .класу си- мволівзворотного посилання/back reference (див. наступний роз- діл)субпатерна в дужках (якщо це не затвердження/assertion, див. далі)

Квантифікатор загального повторення специфікує мінімальну та мак- симальну кількість допустимих збігів, маючи два числа у фігурних дуж- ках, розділені комою. Число має бути менше 65536, а перше має бути менше або дорівнює другому. Наприклад: z {2,4} збігається з "zz", "zzz" або "zzzz". Фігурна дужка, що закриває, сама по собі не є спеціальним символом. Якщо друге число відсутнє, але кома є, верхньої межі не- має; якщо відсутні друге число і кома, квантифікатор специфікує точну кількість необхідних збігів. Таким чином, [aeiou] {3,} збігається з як міні- мум трьома 3 послідовними голосними, але може і з великою кількі- стю, а d {8} збігається точно з 8 цифрами. Фігурна дужка, що відкри- ває, яка з'являється в позиції, де квантифікатор неприпустимий, або дужка, яка не відповідає синтаксису квантифікатора, вважається літе- ральним символом. Наприклад, {,6} це не квантифікатор, а літераль- ний рядок із 4 символів.

Квантифікатор {0} припустимо, змушуючи вираз поводитися так, ніби попередній елемент та квантифікатор не існують.

Для зручності (і зворотної сумісності) три найбільш поширені квантифі- катори мають односимвольні скорочення: \* еквівалентний {0,}+ еквіва- лентний {1,}? еквівалентний {0,1} Можна конструювати нескінченні ци- кли, ввівши після субпатерну, який не збігається з жодним символом, квантифікатор, що не має верхньої межі, наприклад: (a?) \* Ранні версії Perl і PCRE є джерелами помилок у процесі компіляції таких патернів . Однак, оскільки трапляються випадки, коли це необхідно, такі патерни приймаються, але якщо будь-яке повторення такого субпатерну фак- тично не збігається з жодними символами, цикл форсовано перерива- ється.

За замовчуванням квантифікатори є "жадібними", тобто вони збіга- ються максимально можливу кількість разів (до максимально допусти- мої кількості разів), не викликаючи невдачі виконання решти патерну. Класичний приклад, коли це створює проблеми – спроба знайти збіги у коментарі C-програм. Коментарі з'являються між символами /\* та \*/, а всередині можуть з'являтися окремі символи \* та /. Спроба знайти збіг з C-коментарями, застосувавши патерн/\\*.\*\\*/до рядка/\* перший ко- ментар \*/ не коментар /\* другий коментар \*/ зазнає невдачі, оскільки відбувається збіг із цілим рядком через жадіб- ність елемента .\*.

Однак, якщо після квантифікатора йде питання, він перестає бути жа- дібним і збігається мінімально можливу кількість разів, тому па- терн/\\*.\*?\\*/вірно виконується з C-коментарями. Значення різних інших квантифікаторів не зміниться, лише переважна кількість збігів. Не плу- тайте це використання питання питання з його використанням як вла- сне квантифікатор. Оскільки він може використовуватися подвійно, вони іноді може з'являтися подвоєним: що збігається з однією циф- рою, переважно, але може збігатися і з двома, якщо це єдиний спосіб збігу частини патерну, що залишилася.

Якщо встановлено опціюPCRE\_UNGREEDY(відсутня в Perl), то кван- тифікатори не жадібні за замовчуванням, але окремі можуть бути жа- дібними, якщо після них стоїть знак питання. Інакше кажучи, питання інвертує поведінка за умовчанням.

Коли субпатерн у дужках квантифікований мінімальною кількістю по- вторень, яка більша за 1, або має обмеження максимуму, для відком- пільованого патерну потрібно більше місця, пропорційно розміру міні- муму або максимуму.

Якщо патерн починається з .\* або з .{0,} та встановлено оп- ціюPCRE\_DOTALL(еквівалентна Perl'івській /s), дозволяючи таким чи- ном збіг . з символами нового рядка, то патерн неявно заякорюється, оскільки все, що йде слідом, випробовуватиметься щодо кожної сим- вольної позиції в рядку-суб'єкті, тому після першої немає іншої позиції для відновлення спроб знайти повний збіг. PCRE розглядає такий па- терн так, якби йому передувало \A. Коли відомо, що рядок-суб'єкт не містить символів нового рядка, краще встановити PCRE\_DOTALLякщо патерн починається з .\*, щоб отримати цю опти- мізацію, або, альтернативно, використовувати ^ для явного позна- чення заякорювання.

Коли захоплюючий субпатерн повторюється, захопленим значенням є підрядок, що збігається з останньою ітерацією. Наприклад, після того як(tweedle[dume]{3}\s\*)+збігається з "tweedledum tweedledee", значен- ням захопленого підрядка буде "tweedledee". Однак, якщо є вкладені захоплюючі субпатерни, відповідні захоплені значення можуть бути встановлені у попередніх ітераціях. Наприклад, після того як/(a|(b))+/збігається з "aba", значенням другого захопленого підрядка буде "b".

Посилання/BACK REFERENCES

Поза класом символів, зворотний слеш з наступною цифрою більше 0 (і можливими наступними цифрами) є зворотним посиланням на попе- редній захоплюючий субпатерн (тобто зліва від себе) в патерні, припу- скаючи, що була достатня кількість попередніх захоплюючих лівих ду- жок.

Однак, якщо 10-річне число, що йде після backslash, менше 10, воно завжди вважається зворотним посиланням і викликає помилку лише тоді, коли у всьому патерні немає достатньої кількості захоплюючих лі- вих дужок. Іншими словами, дужки, на які посилаються, не повинні бути ліворуч від посилання для числа менше 10. Див. розділ "Зворот- ний слеш/Backslash" раніше детальну інформацію про обробку чисел, що йдуть після зворотного слеша.

Зворотне посилання збігається з усім тим, із чим збігається захоплюю- чий субпатерн у поточному рядку-суб'єкті, а не з тим, з чим збігається сам субпатерн. Тому патерн(sens|respons)e and \1ibility збігається з "sense and sensibility" і "response and responsibility", але не з "sense and responsibility". Якщо збіг з урахуванням регістру діє у момент появи зворотного посилання, то регістр символів враховується. Напри- клад,((?i)rah)\s+\1збігається з "rah rah" і з "RAH RAH", але не з "RAH rah", хоча оригінальний захоплюючий субпатерн збігається без ураху- вання регістру.

В одному субпатерні може бути більше одного зворотного посилання. Якщо субпатерн на даний момент не використовується у певному збігу, то будь-які зворотні посилання на нього зазнають невдачі. На- приклад, патерн(a|(bc))\2завжди зазнає невдачі, якщо починає збіга- тися з "a" раніше, ніж з "bc". Оскільки може бути до 99 зворотних поси- лань, всі цифри, що йдуть після backslash, вважаються частиною по- тенційної зворотної посилання. Якщо патерн продовжується цифро- вим символом, повинен використовуватися певний обмежувач для за- кінчення зворотного посилання. Якщо опція PCRE\_EXTENDEDвстановлена, це може бути пробіл. Інакше може ви- користовуватися порожній коментар. Зворотне посилання, яке з'явля- ється всередині дужок, до яких вона звертається, зазнає невдачі, якщо спочатку використовується субпатерн; так, наприклад, (a1) ні- коли не збігається. Однак такі посилання можуть використовуватися всередині субпатернів, що повторюються. Наприклад, па- терн(a|b\1)+збігається з будь-якою кількістю "a", а з "aba", "ababaa" etc. При кожній ітерації субпатерна зворотне посилання збігається з ряд- ком символів відповідно попередньої ітерації. Щоб це працювало, па- терн має бути таким, щоб перша ітерація не повинна була співпадати зі зворотним посиланням. Це можна зробити за допомогою чергу- вання, як у попередньому прикладі або квантифікатором з мінімумом 0.

Твердження/Assertions

Затвердження це перевірка символів, що йдуть слідом або попередніх точці збігу, не використовуючи реально жодних символів. Прості твер- дження, кодовані як \b, \B, \A, \Z, \z, ^ і $, були розглянуті раніше. Більш складні твердження кодовані як субпатерни. Є твердження двох видів: ті, що дивляться вперед/ahead від поточної позиції, в рядку-суб'- єкті, і ті, що дивляться назад/behind.

Субпатерн затвердження збігається звичайним чином, крім того, що він не викликає зміни поточної позиції збігу. Твердження вперед почи- наються з (?= для позитивного затвердження і з (?! - для негативного затвердження. Наприклад, w+(?=;)збігається зі словом з наступною то- чкою з комою, але не включає точку з комою в збіг, іfoo(?) !bar)збіга- ється з будь-якою появою "foo", після якої не йде "bar".Зверніть увагу, що схожий патерн(?!foo)barне знайде ніяких входжень "bar", перед якими йде що-небудь, крім "foo"; він знайде, проте, будь-яке вхо- дження "bar", оскільки затвердження (?!foo) завжди TRUE, якщо насту- пні три символи це "bar", яке може збігатися з двома різними розмірами, але це допускається, якщо переписати його з двома відгалуженнями верхнього рі- вня:(?<=abc|abde)Твердження назад/lookbehind реалізовані так, що, для кожної альтернативи, тимчасово зсувається поточна позиція на фіксовану ширину, а потім робиться спроба знайти збіг. Якщо перед поточною позицією недостатньо символів, збіг вважається невдалим. Твердження назад у поєднанні з once-only субпатернами можуть особ- ливо стати в нагоді для пошуку збігів в кінці рядків; Приклад дано в кі- нці цього розділу для once-only субпатернів. тимчасово зсувається по- точна позиція на фіксовану ширину, а потім робиться спроба знайти збіг. Якщо перед поточною позицією недостатньо символів, збіг вважа- ється невдалим. Твердження назад у поєднанні з once-only субпатер- нами можуть особливо стати в нагоді для пошуку збігів в кінці рядків; Приклад дано в кінці цього розділу для once-only субпатернів. тимча- сово зсувається поточна позиція на фіксовану ширину, а потім ро- биться спроба знайти збіг. Якщо перед поточною позицією недостат- ньо символів, збіг вважається невдалим. Твердження назад у поєд- нанні з once-only субпатернами можуть особливо стати в нагоді для пошуку збігів в кінці рядків; Приклад дано в кінці цього розділу для once-only субпатернів.

Різні твердження (будь-якого виду) можуть слідувати один за одним. Наприклад,(?<=\d{3})(?<!999)fooзбігається з "foo" з попередніми трьома цифрами, які не є "999". Зауважте, що кожне твердження за- стосовується незалежно у тій самій точці рядка-суб'єкта. Спочатку ви- конується перевірка, що попередні три символи це цифри, потім пере- віряється, що ці три цифри не є числом "999". Цей патерн не збіга- ється з "foo" із попередніми шістьма цифрами, перші з яких є циф- рами, а останні три не утворюють "999". Наприклад, він не збігається з "123abcfoo". Це зробить патерн(?<=\d{3}...)(?<!999)fooНа цей раз пе- рше твердження переглядає попередні шість символів, перевіряючи, що перші три є цифрами, а потім друге твердження перевіряє.

Твердження можуть вкладатися у будь-якому поєднанні. Напри- клад,(?<=(?<!foo)bar)bazзбігається з "baz" з попереднім "bar", перед яким, у свою чергу, немає "foo", а(?<=\d{3}(?!). 999)...)fooце інший па- терн, який збігається з "foo" з попередніми трьома цифрами та будь- якими трьома символами - не "999".

Субпатерни тверджень не є захоплюючими субпатернами і не можуть повторюватися, оскільки немає сенсу стверджувати те саме кілька ра- зів (це дивлячись у якій країні... - прим. перекл.). Якщо твердження будь-якого типу містить захоплюючі субпатерни, вони обраховуються для нумерації захоплюючих субпатернів всього патерну. Проте захоп- лення підрядок виконується лише позитивних тверджень, оскільки це немає сенсу для негативних тверджень.

Твердження обраховую максимум до 200 субпатернів.

Once-only/"Одноразові" субпатерни

І при мінімальній, і при максимальній кількості повторень, невдача того, що йде слідом, нормально викликає повторне обчислення повто- рюваного елемента, з метою перевірити, чи не дасть збігу повторення, іншу кількість разів, що залишилася, частини патерну. Іноді потрібно запобігти цьому зміни природи збігу, або щоб викликати невдачу ра- ніше, ніж це могло б бути, якщо автор патерна знає, що більше немає точок для роботи.

Розглянемо, наприклад, патерн \d+foo у застосуванні до рядка-суб'- єкта123456barПісля збігу всіх шести цифр і невдачі збігу з "foo", нор- мальною дією буде - спробувати знову тільки вже з п'ятьма цифрами \d+ item, а потім з чотирма, і так далі , Перш ніж остаточно зазнати не- вдачі. Одноразові субпатерни надають засіб для специфікування дій, як тільки частина патерну збіглася: тоді він не обчислюється повторно, і пошук збігів негайно завершується за першої невдачі збігу з "foo". Но- тується це іншим видом спеціальних дужок, що починаються з (?>(?>\d+)barЦей вид дужок "переглядає" частину патерну, що міс- титься в них, якщо вона збіглася, а подальша невдача в патерні запо- бігає повернення і повторний перегляд. елементам, однак, альтернативний опис такий, що субпатерн цього типу збігається з ряд- ком символів, з яким міг би збігтися ідентичний окремий субпатерн, якщо він заякорений у поточній точці рядка-суб'єкта.

Одноразові субпатерни є захоплюючими субпатернами. Прості випа- дки, на кшталт вищенаведеного прикладу, можна подати як максима- лістську побудову, яка поглинає все, що може. Так, тоді як \d+ та \d+? підготовлені так, щоб уточнювати число цифр для збігу з ними, щоб збіглася решта патерну (?>\d+) може збігатися тільки з повною послі- довністю цифр.

Ця конструкція, зрозуміло, може містити довільно ускладнені субпате- рни і вкладатися.

Одноразові субпатерни можна використовувати у поєднанні із твер- дженнями назад для специфікації ефективного збігу в кінці рядка-суб'- єкта. Розглянемо такий простий патерн: abcd $ застосований до дов- гого рядка, з яким він не збігається. Оскільки збіг виконується зліва на- право, PCRE шукатиме кожну "a" в суб'єкті, а потім подивиться, що з наступного збігається із залишком патерну. Якщо патерн специфікова- ний так:^.\*abcd$то початкове .\* збігається спочатку з усім рядком, але якщо це зазнає невдачі (оскільки немає наступних "a"), виконується повернення і проглядається все, крім останнього символу, потім все, крім останніх двох символів і так далі. Пошук "a" покриває весь рядок, праворуч наліво, і нічого не може бути краще. Однак, якщо патерн за- писано так: ^(?>.\*)(?< =abcd)то може бути повернення назад для еле- мента .\* item; він може збігтися тільки з цілим рядком. Наступне твер- дження назад виконує просту перевірку останніх чотирьох символів. Якщо вона зазнає невдачі, збіг зазнає невдачі негайно. Для довгих ря- дків цей підхід має значну відмінність у часі виконання.

Якщо патерн містить необмежену повторення всередині субпатерну, який сам може повторюватися необмежену кількість разів, викорис- тання once-only субпатерну виявляється єдиним способом уникнути невдалих збігів, які тривають досить тривалий час. з необмеженою кі- лькістю підрядків, які складаються з не-цифр, або з цифр, ув'язнених в <>, з наступним ! або?. Якщо збіг є, він виконується швидко. Однак, якщо застосує патерн буде витрачено багато часу, перш ніж буде повідомлено про невдачу. Це тому, що цей рядок може бути поділена на два повторення великою кі- лькістю способів, і всі вони будуть випробувані. (Приклад використо- вує [!?] замість простого символу в кінці, так як і PCRE, та Perl містять оптимізацію, яка дозволяє прискорити виявлення невдачі, якщо вико- ристовується одиночний символ. вони запам'ятовують останній одино- чний символ, необхідний для збігу, і видають невдачу раніше, якщо цей символ відсутній у рядку.) Якщо змінити патерн так((?>\D+)|<\d+>)\*[!?]послідовності не-цифр не можуть бути розірвані, і невдача виявляється швидше.

Умовні субпатерни

Можна змусити субпатерн у процесі збігу підкорятися умовно або ви- бирати з двох альтернативних субпатернів, залежно від результату за- твердження або від того, чи збігся попередній захоплюючий субпатерн чи ні. Ось дві можливі форми умовного субпатерну:(?(condition)yes- pattern)(?(condition)yes-pattern|no-pattern)Якщо умова виконана, вико- ристовується yes-pattern; інакше використовується no-pattern (якщо є). Якщо субпатерне є більше двох альтернатив, виникає помилка компі- ляції.

Є умови двох видів. Якщо текст між дужками складається із послідов- ності цифр, то умова виконана, якщо захоплюючий субпатерн цього числа раніше збігся. Розглянемо наступний патерн, який містить не- значну прогалину для зручності читання (припустимо наявність оп- ціїPCRE\_EXTENDED) і для поділу патерну на три частини для полег- шення обговорення:( \( )?[^()]+(?(1) \) )Перша частина збігається з не- обов'язковою дужкою, що відкриває, і, якщо символ є, встановлює його як першу збіглу підрядок.Друга частина збігається з одним або бі- льше символів, які не є дужками. Третя частина це умовний субпа- терн, який перевіряє, чи збігається перший набір дужок чи ні. Якщо збігся, тобто якщо суб'єкт починається з дужки, що відкриває, умова буде TRUE, буде виконуватися yes-pattern і необхідна закриваюча ду- жка. Інакше оскільки no-pattern відсутня, субпатерн не збігається ні з чим. Іншими словами, цей патерн збігається з послідовністю не-дужок, можливо, укладеної у дужки.

Якщо умова перестав бути послідовністю цифр, вона має бути твер- дженням. Це може бути позитивне або негативне lookahead або lookbehind твердження. Розглянь патерн, знову містить незначний пробіл і дві альтернативи в другому рядку: }-\d{2} |\d{2}-\d{2}-

\d{2} )Умова це позитивне випереджаюче/lookahead твердження, яке збігається з необов'язковою послідовністю не-літер з наступною літе- рою. Іншими словами, воно перевіряє наявність мінімум однієї літери у суб'єкті. Якщо букву знайдено, знову перевіряється збіг суб'єкта з першою альтернативою; інакше перевіряється збіг із другою альтерна- тивою. Цей патерн збігається з рядками однієї з двох форм: dd-aaa-dd або dd-dd-dd, де aaa – це букви, а dd – це цифри.

Коментарі

Послідовність символів (?# означає початок коментаря, який триває до наступної дужки, що закриває. Вкладення дужок не допускається. Символи, що утворюють коментар, не є частиною збігу патерну.

Якщо встановлено опціюPCRE\_EXTENDED, не escape-ований символ # поза класом символів починає коментар, який продовжується до на- ступного символу нового рядка в патерні.

Рекурсивні паттерни

Розглянемо проблему збігу рядка у дужках, коли допускається необ- межене вкладення дужок. Без використання рекурсії, найкраще, що можна зробити, це використовувати патерн, який збігається на деяку фіксовану глибину вкладення. Неможливо обробити вкладення дові- льно велику глибину. Perl 5.6 має експериментальну можливість, що дозволяє (крім іншого) виконувати рекурсію регулярних виразів. Спеці- альний елемент (?R) надано для цього специфічного case/варіанту ре- курсії. PCRE-патерн вирішує проблему дужок (припускаючи, що опція PCRE\_EXTENDEDвстановлена так, що пробіл ігнору-

ється):\( ( (?>[^()]+) | (?R) )\* \) Спочатку він збігається з відчиняючою дужкою. Потім - з будь-якою кількістю підрядків, які можуть бути або послідовностями не-дужок, або рекурсивним збігом самого патерну (тобто коректно укладеного в дужки підрядком). Нарешті йде закрива- юча дужка.

Цей особливий приклад патерну містить вкладене нескінченне повто- рення, і, таким чином, використання одноразового субпатерну для збігу з рядками з не-дужок дуже важливо, коли патерн застосовується до рядків, які не збігаються. Наприклад, якщо його застосувати до (aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa) то він швидко дасть "немає збігів". Однак, якщо одноразовий/once-only субпатерн не використовується, пошук можуть кроїти рядок-суб'єкт, і всі вони повинні бути перевірені, перш ніж буде видано повідомлення про невдачу пошуку.

Значення, встановлені для будь-якого субпатерну, беруться із зовніш- нього рівня рекурсії, на якому встановлюється значення субпатерну. Якщо вищенаведений патерн підставити щодо (ab(cd)ef) то значенням для захоплюючих дужок буде "ef", яке є останнім значенням, що прий- мається на верхньому рівні. Якщо додати додаткові ду-

жки\( ( ( (?>[^()]+) | (?R) )\* )\) то рядок, що захоплюється ними, буде "ab(cd)ef" - вміст дужок верх- нього рівня. пам'ять потім через pcre\_free. Якщо неможливо виділити пам'ять, він зберігає дані тільки для перших 15 захоплюючих дужок, тому що немає способу видати помилку out-of-memory зсередини ре- курсії.

Продуктивність

Деякі елементи, які можуть з'являтися у патернах, працюють ефектив- ніше, ніж інші. Більш ефективно використовувати клас символів, такий як [aeiou], ніж набір альтернатив, такий як (a|e|i|o|u). В цілому, прос- тіша конструкція є більш ефективною. Книга Jeffrey Friedl'а містить ве- лику дискусію щодо оптимізації регулярних виразів для підвищення продуктивності.

Якщо патерн починається з .\* та встановлено опціюPCRE\_DOTALLПа- терн неявно заякорюється PCRE, оскільки він може збігтися тільки на початку рядка-суб'єкта. Однак, якщоPCRE\_DOTALLне встановлена, PCRE не може виконати цю оптимізацію, оскільки метасимвол . не збі- гається тоді із символом нового рядка/newline, і, якщо рядок-суб'єкт мі- стить newlines, патерн може збігтися із символом, що йде безпосеред- ньо після одного із символів нового рядка, замість того, щоб збігатися лише на самому початку. Наприклад, патерн(.\*) second

збігається в суб'єкті "first\nand second" (де \n це символ нового рядка) з першим захопленим підрядком "and". Для цього PCRE намагається збігатися спочатку після кожного символу нового рядка в суб'єкті.

Якщо ви використовуєте такий патерн з рядками-суб'єктами, які не міс- тять newlines, найкраща продуктивність буде досягнута установ- коюPCRE\_DOTALL, або якщо розпочати патерн з ^.\* для явного вказі- вки заякорювання. Це утримає PCRE від необхідності сканувати суб'- єкт у пошуках newline для рестарту з нього.

Уникайте створення паттернів, які містять нескінченні повторення. Вони можуть зайняти багато часу, якщо застосувати їх до рядка, який не містить збігів. Розглянемо фрагмент патерну

(a+)\*

Він може збігтися з "aaaa" 33 різними способами, і ця кількість збіль- шується дуже швидко зі збільшенням довжини рядка. (Повторення \* може збігтися 0, 1, 2, 3 або 4 рази, і для кожного випадку/case, відмін- ного від 0, повторення + можуть збігатися різну кількість разів.) Якщо залишок патерну такий, що весь збіг зазнає невдачі, PCRE має, в принципі, спробувати виконати всі можливі варіанти, і це може вима- гати величезної кількості часу.

За допомогою оптимізації можна відловити найпростіші випадки, такі як

(a+)\*b

де слідом йде літеральний символ. Перш ніж покладатися на стандар- тну процедуру пошуку збігів, PCRE перевіряє, чи є "b" далі в рядку-суб'єкті, і якщо ні, збіг негайно завершується невдачею. Однак, коли на- ступного літералу немає, ця оптимізація не може бути використана. Відчуйте різницю, порівнявши поведінку  
(a+)\*\d

з поведінкою вищенаведеного патерну. Перший видає невдачу од- разу, коли застосовується до рядка символів "a", а другий витрачає зна- чний час на пошук у рядках довжиною понад 20 символів.

Cookies

PHP прозоро підтримує HTTP cookies. Cookies це механізм збері- гання даних броузером віддаленої машини для відстеження або іден- тифікації відвідувачів, що повертаються. Ви можете встановити cookies за допомогою функцій setcookie() або setrawcookie(). Cookies є части- ною HTTP-заголовка, тому setcookie() має викликатися до будь-якого виведення даних у броузер. Це те саме обмеження, яке має функція header(). Ви можете використовувати функції буферизації виводу, щоб затримати виведення результатів роботи скрипта до того моменту, коли буде відомо, чи знадобиться встановлення cookies або інших за- головків HTTP.

Будь-які cookies, надіслані серверу броузером клієнта, будуть ав- томатично перетворені на змінні PHP, подібно до даних методів GET і POST. На цей процес впливають конфігураційні директиви register\_globals та variables\_order. Для призначення кількох значень од- нієї cookie просто додайте [] до її імені.

У PHP 4.1.0 і вище глобальний масив $\_COOKIE завжди автомати- чно заповнюється значеннями отриманих cookies. У попередніх версіях визначається масив $HTTP\_COOKIE\_VARS але тільки коли включена директива track\_vars. (Ця директива завжди включена з PHP 4.0.3.)

Cookie є невеликим пакетом інформації, переданим web-серве- ром і що зберігається на клієнтському комп'ютері. У cookie можна збе- регти корисні дані, що описують стан сеансу користувача, щоб у май- бутньому завантажити їх і відновити параметри сеансового зв'язку між сервером і клієнтом. Cookie використовуються на багатьох сайтах Ін- тернету для розширення можливостей користувача та підвищення ефективності сайту за рахунок відстеження дій та особистих уподо- бань користувача. Можливість зберігання цих відомостей відіграє клю- чову роль на сайтах електронної комерції, що підтримують персона- льне налаштування та цільову рекламу.

Внаслідок того, що cookie зазвичай зв'язуються з конкретним ко- ристувачем, вони часто зберігають унікальний ідентифікатор користу- вача (UIN). Цей ідентифікатор заноситься до бази даних на сервері і використовується як ключ для вибірки з бази всієї інформації, пов'яза- ної з цим ідентифікатором. Звичайно, збереження UIN у cookie не є обов'язковою вимогою; Ви можете зберегти будь-яку інформацію за умови, що її загальний обсяг не перевищує 4 Кбайт (4096 байт).

Компоненти cookie

В cookie зберігаються інші компоненти, за допомогою яких розробник може обмежувати використання cookie з позицій домену, шляху, тер- міну дії та безпеки. Нижче наведено опис різних компонентів cookie:

* Ім'я - ім'я cookie є обов'язковим параметром, яким програма посилається на cookie. Можна провести аналогію між ім'ям cookie та ім'ям змінної.
* Значення - Фрагмент даних, пов'язаний з ім'ям cookie. У цих даних може зберігатися будь-яка інформація - ідентифікатор корис- тувача, колір тла, поточна дата тощо.
* Термін дії - Дата, що визначає тривалість існування cookie. Як тільки поточна дата та час перевищують заданий термін дії, cookie стає недійсним та перестає використовуватися. Відпо- відно до специфікації cookie встановлювати термін дії для cookie необов'язково. Тим не менш, PHP для роботи з cookie вимага- ють, щоб термін дії встановлювався. Згідно зі специфікацією, якщо термін дії не вказаний, cookie стає недійсним наприкінці се- ансу (тобто, коли користувач залишає сайт).
* Домен - Домен, який створив cookie і може читати його значення. Якщо домен складається з декількох серверів і доступ до cookie повинен бути дозволений всім серверам, ім'я домену можна задати у формі .phprecipes.com. У цьому випадку всі потенційні до- мени третього рівня, що належать сайту PHPrecipes (наприклад, wap.phprecipes.com або news.phprecipes.com), зможуть працю- вати з cookie. З міркувань безпеки cookie можуть встановлюва- тися лише домену сервера, який намагається створити cookie. Цей компонент необов'язковий; якщо він не вказаний, за замов- чуванням використовується ім'я домену, з якого була підлога; чено значення cookie.
* Шлях - URL, з якого надається доступ до cookie. Будь-які спроби отримати доступ до cookie за межами цього шляху припиня- ються. Цей компонент необов'язковий; якщо він не заданий, за умовчанням використовується шлях до документа, який створив cookie.
* Безпека - параметр, який показує, чи допускається читання cookie у небезпечному середовищі. За промовчанням використо- вується значення FALSE.

Встановлення cookies

Установка cookies виконується за допомогою функції setcookie: setcookie

Синтаксис:

boolsetcookie (string name [, string value [, int expire [, string path [, string domain [, int secure]]]]])

* Ця функція має такі аргументи:
* name- ім'я встановлюваного cookie;
* value- значення, яке зберігається в cookie з ім'ям $name;
* expire- час у секундах з початку епохи, після якого поточний cookie стає недійсним;
* path- шлях, яким доступний cookie;
* domain- Домен, з якого доступний cookie;
* secure- директива, яка визначає, чи доступний cookie не за запи- том HTPPS. За промовчанням ця директива має значення 0, що означає можливість доступу до cookie за звичайним запитом HTTP.

При роботі з cookie необхідно враховувати важливий момент, що полягає в тому, що cookie треба обов'язково встановлювати перед відправкою в броузер будь-яких заголовків, оскільки самі cookie вста- новлюються у вигляді заголовків. Тому якщо встановити cookies після будь-якого тексту, що надсилається в броузер, виникне хибна ситуа- ція.

Встановлення терміну придатності cookies

За замовчуванням cookies встановлюються на один сеанс роботи з броузером, проте можна задати їм більш тривалий термін існування. Це дуже зручна та корисна властивість, оскільки в цьому випадку ко- ристувачу не потрібно надавати свої дані знову при кожному відвіду- ванні сайту.

Як мовилося раніше, термін придатності встановлюється у секун- дах щодо початку епохи. У PHP існують функції time та mktime для ро- боти з датою та часом, що дозволяють переводити поточний час у кі- лькість секунд від початку епохи. Функція time просто переводить по- точний системний час у кількість секунд, що минули початку епохи:

time

*Синтаксис:* time*();*

Удосконаленим варіантом функції time є функція mktime.

*Синтаксис:*

intmktime([int hour [, int minute [, int second [, int month [, int day[, int year [, int is\_dst]]]]]]])

Аргумент is\_dst цієї функції визначає, чи ця дата потрапляє в пе- ріод літнього часу і може приймати наступні значення:

* -1 (За замовчуванням. Це означає, що властивість не задано);
* 0 (Тимчасовий інтервал не припадає на період літнього часу);
* 1 (Тимчасовий інтервал припадає на період літнього часу).

Видалення cookie

Видалити cookie легко. Для цього потрібно викликати функцію setcookie і передати їй ім'я того cookie, який підлягає видаленню:

setcookie("name");  
Інші встановлені cookie не видаляються.

Проблеми безпеки, пов'язані з cookies

Іноді cookies доводиться зберігати конфіденційні дані, і в цьому випадку розробник повинен подбати про те, щоб інформація, що збе- рігається в cookie, не була передана третім особам. Існує кілька мето- дів захисту інформації, що зберігається в cookie:

* встановлення області видимості cookies;
* шифрування;
* обмеження доступу для доменів;
* надсилання cookies за захищеним запитом.
* Найкращим рішенням є комплексне застосування всіх цих мето- дів.

Встановлення області видимості cookie

Оскільки, за замовчуванням, доступ до cookie походить з корене- вого каталогу, це може створити "дірки" в системі захисту, оскільки cookies стають доступними у будь-якому підкаталозі цього каталогу. Обмежити доступ до cookies для всіх сторінок, крім розміщених у конк- ретному каталозі, наприклад, /web, можна так:

setcookie*(*"name", $value, "/web/");

Однак і в цьому випадку, наприклад, каталоги /web/index.php, /web1/page.html тощо будуть задовольняти цьому обмеженню. Якщо таке положення також небажане, можна обмежити область видимості cookies до конкретної сторінки:

setcookie*(*"name", $value, "/web/index.php");

Однак і такий спосіб повною мірою не вирішує проблему, тому що в цьому випадку доступ до інформації, яка міститься в cookie, може отримати, наприклад, скрипт /web/index.php-script/anti\_cookie.php. Тому виникає потреба у шифруванні.

Сесії у PHP

Сесії та cookies призначені для збереження відомостей про корис- тувачів під час переходів між кількома сторінками. Під час викорис- тання сесій дані зберігаються у тимчасових файлах на сервері.

Використання сесій та cookies дуже зручне та виправдане у таких додатках як Інтернет-магазини, форуми, дошки оголошень, коли, по- перше, необхідно зберігати інформацію про користувачів протягом кі- лькох сторінок, а по-друге, своєчасно надавати користувачеві нову ін- формацію.

Протокол HTTP є протоколом "без збереження стану". Це озна- чає, що даний протокол не має вбудованого способу збереження між двома транзакціями. Т. е., коли користувач відкриває спочатку одну сторінку сайту, а потім переходить на іншу сторінку цього ж сайту, то ґрунтуючись лише на засобах, що надаються протоколом HTTP немо- жливо встановити, що обидва запити належать до одного користу- вача. Т. о. потрібний спосіб, за допомогою якого було б відстежувати інформацію про юзера протягом одного сеансу зв'язку з Web-сайтів. Одним із таких методів є керування сеансами за допомогою призначе- них для цього функцій. Для нас важливо те, що сеанс по суті є групою змінних, які, на відміну від звичайних змінних, зберігаються і після за- вершення виконання PHP-сценарію.

Працюючи з сесіями розрізняють такі етапи:

* відкриття сесії
* реєстрація змінних сесії та їх використання
* закриття сесії

Найпростіший спосіб відкриття сесії полягає у використанні функції session\_start, яка викликається на початку PHP-сценарію:

session\_start

Синтаксис:

session\_start();

Ця функція перевіряє, чи існує ідентифікатор сесії, і, якщо ні, ство- рює його. Якщо ідентифікатор поточної сесії вже існує, завантажу- ються зареєстровані змінні сесії.

Реєстрація змінних сесії

Взагалі, реєстрація змінних сесії здійснюється за допомогою фун- кції session\_register, проте, починаючи з версії PHP 4.2.0, практика ре- єстрування сеансових змінних зазнала деяких змін, що вносить у це питання деяку плутанину. Справа в тому, що функція session\_register застосовується лише до глобальних змінних і вимагає, щоб параметр register\_globals був увімкнений. Починаючи з цієї версії мови, сеансові змінні завжди реєструються в асоціативних масивах $HTTP\_SESSION\_VARS та $\_SESSION. Т.ч., сесії коректно відкривати присвоєнням значення глобальної змінної:

**$\_SESSION*[*'username'] = "username";**

// або  
$HTTP\_SESSION\_VARS*[*'username'] = "username";

Надалі під час виконання лабораторних робіт рекомендується ви- користовувати змінну $\_SESSION.

Таким чином:

* якщо ви використовуєте функцію session\_register, то має бути увімкнено параметр register\_globals
* **при відключеному register\_globals ініціалізація сеансових змінних відбувається шляхом надання значень елементам асоціативних масивів**

Для коректної роботи програми необхідно перевіряти, чи встанов- лені змінні сесії.

При використанні асоціативних масивів $HTTP\_SESSION\_VARS і $\_SESSION необхідно безпосередньо перевіряти елементи цих маси- вів, наприклад, так:

if(isset($\_SESSION['username']))

Закриття сесії

Після завершення роботи із сесією спочатку потрібно розреєстру- вати всі змінні сесії, а потім викликати функцію session\_destroy:

session\_destroy

Синтаксис:

session\_destroy();

Способи розреєстрації сеансових змінних різняться залежно від цього, як вони зареєстровані. У разі використання масиву $\_SESSION розреєстрація здійснюється викликом функції unset.

unset

Синтаксис:

unset($\_SESSION["username"]);

Виконання роботи

Опис завдань, що вирішуються серверним сценарієм.

Серверний сценарій (PHP-скрипт) реалізує міні-калькулятор для обробки даних, отриманих із завантажених користувачем текстових файлів. Основні завдання, які вирішуються цим сценарієм, такі:

1. Завантаження і обробка файлів

Користувач завантажує два текстових файли, кожен з яких містить список чисел, розділених символом «;». Сценарій зчитує ці файли, розбиває їх на окремі числові значення та перетворює їх у числовий формат. Цей процес гарантує, що дані будуть готові для обчислень, та запобігає помилкам, пов'язаним з неправильним форматом введених даних.

1. Перевірка коректності формату файлів

Скрипт перевіряє, чи обидва завантажені файли містять однакову кількість чисел. Це критично для правильності подальших обчислень, оскільки кожна операція виконується над відповідними парами чисел. Якщо файли містять різну кількість чисел, виводиться повідомлення про помилку, і обчислення не проводяться.

1. Виконання математичних операцій над парами чисел

Користувач може обрати одну з наступних операцій: додавання, віднімання, множення, ділення та піднесення до степеня. Сценарій виконує вибрану операцію над кожною парою чисел, отриманою з обох файлів, і виводить результати для кожної пари.

1. Обробка можливих помилок у процесі обчислень

Під час обчислень сценарій перевіряє можливі помилки, наприклад, ділення на нуль. У разі виникнення такої ситуації для відповідної операції виводиться повідомлення про помилку, замість числового результату. Це допомагає забезпечити надійність та коректність роботи калькулятора.

1. Генерація звіту з результатами обчислень

Після виконання всіх обчислень сценарій генерує HTML-документ, у якому відображаються результати кожної операції над парами чисел. У звіті також зазначаються дата і час генерації, а внизу документу виводиться інформація про виконавця. Цей формат забезпечує зручне відображення та зберігання результатів роботи.

Короткий опис алгоритму роботи сценарію.

1. Завантаження файлів і перевірка даних

* Користувач завантажує два текстових файли, які сервер приймає через форму за допомогою полів file1 і file2.
* Сценарій перевіряє, чи завантажені обидва файли та обрана операція. У разі відсутності цих даних виводиться повідомлення про помилку.

1. Зчитування чисел із файлів

* Сценарій зчитує вміст кожного файлу та розбиває його на масив чисел, використовуючи символ «;» як роздільник.
* Отримані значення конвертуються в числовий формат для подальшої обробки.

1. Перевірка відповідності кількості чисел

* Після зчитування даних з файлів перевіряється, чи обидва файли містять однакову кількість чисел. Це гарантує можливість проведення операцій над кожною парою чисел.
* Якщо кількість чисел не збігається, сценарій припиняє роботу і виводить повідомлення про помилку.

1. Виконання обраної операції над парами чисел

* Залежно від вибору користувача, виконується одна з операцій (додавання, віднімання, множення, ділення або піднесення до степеня) для кожної пари чисел.
* Якщо обрано операцію ділення, сценарій додатково перевіряє, чи не дорівнює друге число нулю, щоб уникнути помилки ділення на нуль. У разі ділення на нуль виводиться відповідне повідомлення.

1. Генерація HTML-документа з результатами

* Сценарій створює HTML-документ, у якому відображаються результати обчислень для кожної пари чисел.
* У документі виводяться дата та час генерації, а також дані про виконавця (ПІБ і група).

Алгоритм сценарію забезпечує обробку файлів, виконання обчислень над числовими даними та коректне відображення результатів у HTML-документі.

Опис використовуваних вхідних даних і функцій

**Вхідні дані**

1. **Файли з числами:**

* Користувач завантажує два текстових файли (file1 і file2). Кожен з файлів містить список чисел, розділених крапкою з комою (;).
* Важливо, щоб обидва файли мали однакову кількість чисел. Якщо кількість чисел не співпадає, виводиться повідомлення про помилку.

1. **Операція:**

* Користувач вибирає одну з п'яти операцій: додавання, віднімання, множення, ділення або піднесення до степеня. Цей вибір передається через форму як значення змінної operation.
* У разі, якщо операція не вибрана або її значення не відповідає жодній з опцій, встановлюється відповідне повідомлення про помилку.

**Функції**

1. **loadNumbersFromFile($filePath)** – функція виконує завантаження чисел з файлу.

*Параметри:*

* $filePath: шлях до тимчасового файлу, який завантажив користувач.

*Логіка роботи:*

* Функція зчитує весь вміст файлу за допомогою file\_get\_contents.
* Якщо файл порожній, функція повертає порожній масив, щоб уникнути подальших помилок обробки.
* Якщо файл містить текст, він розділяється на частини за допомогою explode(";"), а отримані значення перетворюються у числа за допомогою array\_map('floatval', $numbers).

**Обробка вхідних даних:**

1. **Перевірка на наявність файлів і вибір операції:**

* Якщо користувач не завантажив обидва файли або не вибрав операцію, програма завершується з повідомленням про помилку.

1. **Перевірка на порожні файли:**

* Якщо один або обидва файли виявляються порожніми, програма виводить відповідне повідомлення і не продовжує обчислення.

1. **Перевірка на однакову кількість чисел:**

* Якщо кількість чисел у файлах не збігається, виводиться повідомлення про помилку, а обчислення не виконуються.

**Вихідні дані**

1. **Результати обчислень**:

* На сторінці виводиться заголовок, що містить назву виконуваної операції та результат для кожної пари чисел.
* В разі виникнення помилки, користувач отримує повідомлення з деталями проблеми.

1. **Дата і підпис**:

* Виводиться час та дата генерації результату роботи, а також підпис користувача.

Лістинг вихідного коду з коментарями

Код сценарію, що обробляє форму калькулятора подано в Додатку А.

Результати роботи

Результати роботи сценарію подано в таблиці 1.

Таблиця 1 – результат роботи сценарію з різними вхідними данними

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вхідні значення 1 файлу** | **Вхідні значення 2 файлу** | **Операція** | **Очікуваний результат** | **Обробка помилок** |
| 1;2;3 | 4;5;6 | Додавання | 1+4=5, 2+5=7, 3+6=9 | - |
| 10;20;30 | 5;5;5 | Віднімання | 10-5=5, 20-5=15, 30-5=25 | - |
| 2;3;4 | 5;6;7 | Множення | 25=10, 36=18, 4\*7=28 | - |
| 9;12;15 | 3;4;5 | Ділення | 9/3=3, 12/4=3, 15/5=3 | - |
| 2;3;4 | 1;2;3 | Піднесення до степеня | 2^1=2, 3^2=9, 4^3=64 | - |
| 2;0;4 | 2;2;0 | Ділення | 2/2=1, 0/2=0, Помилка: ділення на нуль | Помилка оброблена при діленні на нуль |
| 10;20;30 |  | Додавання | Помилка: один або обидва файли порожні | Виведення помилки для порожнього другого файлу |
|  | 4;5;6 | Віднімання | Помилка: один або обидва файли порожні | Виведення помилки для порожнього першого файлу |
| 1;2 | 3;4;5 | Множення | Помилка: кількість чисел у файлах не збігається | Виведення помилки через різну кількість чисел |
| abc;def | 1;2;3 | Додавання | Помилка: неможливо перетворити текст у числа | Програма повинна ігнорувати неправильні дані |
|  |  | Множення | Помилка: відсутні необхідні файли або не вибрано операцію | Пропущені обов'язкові вхідні дані |
| 2;3; | ;5;6 | Піднесення до степеня | Помилка: один або обидва файли порожні | Пропуск порожніх значень, обробка порожніх файлів |

висновок

В рамках виконання роботи було поставлено завдання розробити серверний сценарій на мові PHP, що реалізує "Міні-калькулятор" для обробки даних із завантажених файлів та виконання математичних операцій над парами чисел. Основною метою було здобуття навичок роботи з PHP, обробки файлів на сервері та валідації вхідних даних.

У процесі виконання роботи було:

* Досліджено синтаксис PHP для серверного програмування та роботу з формами, що дозволило надійно обробляти дані від користувача;
* Ознайомлено з типами даних у PHP, зокрема числовими та рядковими, що дозволило коректно обробляти вміст файлів із числами;
* Розроблено функцію для завантаження та перетворення чисел із файлів, а також перевірку на порожні та некоректні дані;
* Реалізовано логіку виконання математичних операцій, таких як додавання, віднімання, множення, ділення та піднесення до степеня, з обробкою можливих помилок (наприклад, ділення на нуль);
* Створено систему виведення повідомлень про помилки, що допомагає користувачу швидко зрозуміти причину некоректної роботи скрипта (різна кількість чисел, порожні файли, некоректний формат);
* Виведено результати у форматі HTML-документа із зазначенням обраної операції, дати виконання та ПІБ виконавця, що покращує зручність користування скриптом.

В результаті роботи створено ефективний "Міні-калькулятор", який обробляє дані з файлів та виконує вибрані математичні операції, демонструючи засвоєні навички програмування на PHP. Робота дозволила розширити знання з PHP, вивчити обробку файлів на сервері, забезпечення валідації та опрацювання помилок у веб-додатках.

*Додаток А*Машинний лістинг файлу **calculator\_for\_files.php**

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="UTF-8">

<title>Міні-калькулятор з обробкою файлів</title>

</head>

<body>

<?php

date\_default\_timezone\_set('Europe/Kiev');

$errormsg = "";

if (!isset($\_FILES['file1'], $\_FILES['file2'], $\_POST['operation'])) {

$errormsg .= "Помилка: відсутні необхідні файли або не вибрано операцію.<br>";

exit;

}

$file1Path = $\_FILES['file1']['tmp\_name'];

$file2Path = $\_FILES['file2']['tmp\_name'];

$operation = $\_POST['operation'];

// Встановлення назви операції для відображення

switch ($operation) {

case 'addition':

$operationName = 'Додавання';

break;

case 'subtraction':

$operationName = 'Віднімання';

break;

case 'multiplication':

$operationName = 'Множення';

break;

case 'division':

$operationName = 'Ділення';

break;

case 'exponentiation':

$operationName = 'Піднесення до степеня';

break;

default:

$operationName = 'Невідома операція';

}

// Функція для завантаження чисел з файлу

function loadNumbersFromFile($filePath)

{

$content = file\_get\_contents($filePath);

if (trim($content) === '') {

return []; // Повертаємо порожній масив, якщо файл порожній

}

$numbers = explode(";", trim($content));

return array\_map('floatval', $numbers);

}

// Завантаження чисел з файлів

$numbers1 = loadNumbersFromFile($file1Path);

$numbers2 = loadNumbersFromFile($file2Path);

// Перевірка на порожні файли

if (empty($numbers1) || empty($numbers2)) {

$errormsg .= "Помилка: один або обидва файли порожні. Будь ласка, завантажте файли з числами.<br>";

}

// Перевірка на однакову кількість чисел у файлах

if (count($numbers1) !== count($numbers2)) {

$errormsg .= "Помилка: кількість чисел у файлах не збігається.<br>";

}

if (!empty($errormsg)) {

echo "<h2>Відбулася помилка під час виконання:</h2>";

echo "$errormsg";

} else {

// Якщо всі перевірки пройдено, виконуємо обчислення

echo "<h2>Результати обчислень:</h2>";

echo "<h3>Операція, яка виконувалась: $operationName</h3>";

// Обробка кожної пари чисел

foreach ($numbers1 as $index => $number1) {

$number2 = $numbers2[$index];

$operationResult = '';

// Виконання обраної операції

switch ($operation) {

case 'addition':

$operationResult = $number1 + $number2;

break;

case 'subtraction':

$operationResult = $number1 - $number2;

break;

case 'multiplication':

$operationResult = $number1 \* $number2;

break;

case 'division':

$operationResult = ($number2 != 0) ? ($number1 / $number2) : "Помилка: ділення на нуль";

break;

case 'exponentiation':

$operationResult = pow($number1, $number2);

break;

default:

$operationResult = "Невідома операція";

}

echo "Результат для пари чисел $number1 і $number2: <u>$operationResult</u><br>";

}

}

?>

<footer>

<hr>

<p>Дата генерації: <?php echo date('H:i:s d.m.Y'); ?><br>

Виконавець: Зайченко Ярослав Ігорович, 632п</p>

</footer>

</body>

</html>

*Додаток Б*Машинний лістинг файлу **miniCalc\_mainform.html**

<!DOCTYPE html>

<html lang="uk">

<head>

<meta charset="UTF-8">

<title>Міні-калькулятор з файлами</title>

</head>

<body>

<h2>Міні-калькулятор з файлами</h2>

<form action="calculator\_for\_files.php" method="POST" enctype="multipart/form-data" target="\_blank">

<label for="file1">Файл 1:</label><br>

<input type="file" id="file1" name="file1" accept=".txt" required><br><br>

<label for="file2">Файл 2:</label><br>

<input type="file" id="file2" name="file2" accept=".txt" required><br><br>

<label>Оберіть операцію:</label><br>

<input type="radio" id="addition" name="operation" value="addition" required>

<label for="addition">Додавання</label><br>

<input type="radio" id="subtraction" name="operation" value="subtraction">

<label for="subtraction">Віднімання</label><br>

<input type="radio" id="multiplication" name="operation" value="multiplication">

<label for="multiplication">Множення</label><br>

<input type="radio" id="division" name="operation" value="division">

<label for="division">Ділення</label><br>

<input type="radio" id="exponentiation" name="operation" value="exponentiation">

<label for="exponentiation">Піднесення до степеня</label><br><br>

<input type="submit" value="Обчислити">

<input type="reset" value="Скасувати">

</form>

</body>

</html>