helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

PHP. Praktyczne wprowadzenie



Autor: Ulrich Günther
Tłumaczenie: Piotr Bryja
ISBN: 83-7361-268-8
Tytuł oryginału: PHP Ein praktischer Einstieg
Format: B5, stron: 248

Statyczne strony WWW tworzone w języku HTML nie zawsze spełniają oczekiwania twórców i odbiorców. Coraz częściej ich miejsce zajmują dynamiczne serwisy WWW oparte na bazach danych. Jednym z najpopularniejszych narzędzi do ich tworzenia jest język PHP. Jest to łatwy do opanowania język skryptowy działający po stronie serwera, dystrybuowany na zasadzie open source. Ponieważ ma ogromne możliwości, do tworzenia dynamicznych witryn WWW i aplikacji internetowych używają go setki programistów na całym świecie. Dzięki wiadomościom zawartym w tej książce łatwiej dołączyć do ich grona i napisać samodzielnie mechanizmy, na których opierają się dynamiczne witryny internetowe.

Książka „PHP. Praktyczne wprowadzenie" ułatwia szybkie i bezproblemowe pokonanie dystansu pomiędzy statycznymi witrynami w języku HTML a dynamicznymi serwisami WWW w języku PHP. Na przykładzie prawdziwego projektu — budowy dynamicznej witryny WWW — pokazano, jak używać PHP do tworzenia wszystkich elementów takiej witryny. Autor książki, Urlich Günther, wykorzystując swoje doświadczenie dydaktyczne, przedstawia:

- Informacje o technologiach stosowanych do tworzenia witryn WWW
- Podstawowe zasady programowania w języku PHP
- Tworzenie interaktywnych formularzy
- Stosowanie funkcji i obiektów
- Tworzenie mechanizmów korzystających z baz danych
- Dynamiczne generowanie elementów graficznych
- Techniki wykrywania i usuwania błędów w skryptach

Nauczenie się zasad programowania w PHP to pierwszy krok do realizacji własnych projektów dynamicznych witryn WWW.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Słowo wstępne ***5***

Rozdział 1. A może PHP? ***7***

Przewodnik po książce 8

Pożyteczne narzędzia 9

Rozdział 2. Podstawy techniczne ***15***

Protokół HTTP (HyperText Transfer Protocol) 15

Dokumenty statyczne i dynamiczne 25

Technologie wykonywania po stronie serwera 28

Zalety PHP 32

Rozdział 3. Podstawy HTML ***35***

Praktyczne wprowadzenie do HTML 35

Strona główna witryny poświęconej zagrożonym ptakom z Nowej Zelandii 40

Rozdział 4. Wstęp do programowania: proste skrypty PHP ***53***

Skrypt PHP wyświetlający datę 53

Formularze dla danych wprowadzanych przez użytkownika 58

Przetwarzanie danych z formularza w skryptach PHP 66

Skrypty PHP combo i include 87

Rozdział 5. Efektywne programowanie z wykorzystaniem funkcji i obiektów ***95***

Programowanie oparte na funkcjach 95

Plany rozbudowy witryny 99

Programowanie obiektowe 99

Deklaracja klas dla strony WWW 104
Technika zaawansowana: obiektowe combo datek 123

Rozdział 6. PHP i bazy danych 131
Czym jest baza danych? 131
Podstawowe informacje o bazach danych dla zastosowań opartych na PHP 134
Dostęp do baz danych z poziomu skryptu PHP 147
Dostęp do bazy danych a obiekty 149

Rozdział 7. PHP — ponowne zastosowanie obiektów 169
Strona startowa dla skarbnika 169
Edycja danych o datkach 171
Lista datków 179
Grafika w PHP 186
Wskaźnik statusu kwoty datków 190

Rozdział 8. Inne pożyteczne techniki programowania 197
Wysyłanie poczty elektronicznej 197
Microsoft Office a PHP 202
Liczby i matematyka w PHP 203
Spójność bazy danych 208
Stosowanie zewnętrznych funkcji 212
Kilka słów na temat bezpieczeństwa 214
Poszukiwanie błędów podczas programowania 219

Dodatek A Instalacja i konfiguracja oprogramowania 225
Instalujemy serwer Apache i PHP 225
Instalacja MySQL 230
Instalacja aplikacji phpMyAdmin 232

Dodatek B Zasoby sieci 233
Zasoby sieci na temat PHP 233
Oprogramowanie 234
Literatura uzupełniająca z serii O'Reilly 236

Skorowidz 237

4

Wstęp do programowania: proste skrypty PHP

W rozdziale 2. Czytelnik spotkał się już z nazwą PHP. Wspomniano już, że skrypt PHP jest plikiem, który jest przechowywany na serwerze WWW. Przeglądarka może zażądać jego przysłania przez internet za pomocą zapytania HTTP.

Należy pamiętać, że dokument PHP składa się z kodu HTML ze specjalnymi znacznikami, zawierającymi kod PHP. Serwer przed wysłaniem pliku do przeglądarki przeszukuje dokument w poszukiwaniu tych znaczników. Znajdujący się pomiędzy nimi kod PHP (czyli zbiór poleceń PHP) jest przez serwer natychmiast wykonywany. Otrzymane z wykonania kodu PHP dane wyjściowe są wstawiane w miejsce znaczników PHP w wykonywanym pliku.

Bezpośrednio po przetworzeniu przez serwer danego pliku zostaje one wysłany jako odpowiedź do przeglądarki (*HTTP response*). Odpowiedź jest następnie odpowiednio interpretowana przez przeglądarkę, na ogół jako kod HTML strony WWW.

Skrypt PHP wyświetlający datę

Omówiony proces można zaprezentować za pomocą skryptu, który — tak jak inne pliki, stanowiące przykłady do niniejszej książki — znajduje się na serwerze wydawnictwa Helion (*ftp://ftp.helion.pl/przyklady/phppwp.zip*). Plik z kodem poniższego przykładu nosi nazwę *data.php*:

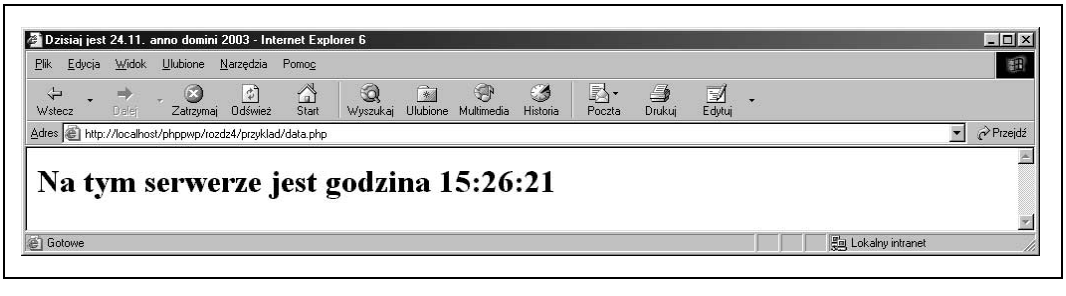
Listing 4.1. Kod pliku data.php

```
<?php
   // Ten prosty skrypt wyświetla bieżącą datę i godzinę.

    $dataiczas = getdate();
    $rok = $dataiczas["year"];
```

```
    $miesiac = $dataiczas["mon"];
    $dzien = $dataiczas["mday"];
    $godziny = $dataiczas["hours"];
    $minuty = $dataiczas["minutes"];
    $sekundy = $dataiczas["seconds"];
?>
<html>
  <head>
    <title>
      Dzisiaj jest <?php echo $dzien.".".$miesiac.". anno domini ".$rok; ?>
    </title>
  </head>
  <body>
    <h1>Na tym serwerze jest godzina
        <?php echo $godziny.":".$minuty.":".$sekundy; ?>
    </h1>
  </body>
</html>
```

Po uruchomieniu tego pliku w przeglądarce, czyli po wpisaniu odpowiedniego adresu URL w polu adresowym przeglądarki, zostanie wyświetlona strona WWW, w tytule której można przeczytać bieżącą datę. W głównym oknie przeglądarki, obok tekstu zapisanego w formacie charakterystycznym dla nagłówka pierwszego stopnia, zostanie wyświetlona aktualna godzina (zobacz rysunek 4.1).



Rysunek 4.1. Dane wyjściowe pliku data.php w oknie przeglądarki

Dobrze byłoby zapoznać się z procesem, który zachodzi przed wyświetleniem opisywanej strony: serwer otrzymuje od przeglądarki żądanie przesłania dokumentu *data.php* i po rozszerzeniu (*.php*) rozpoznaje, że danym plikiem jest plik PHP. Ładuje zatem plik ze swojego dysku twardego do pamięci i przetwarza go w interpreterze PHP. Jak już wspomniano, wykonywane są te fragmenty pliku, które znajdują się między znacznikami `<?php` i `?>`. Stanowią one właściwy kod programu.

Pierwsza część programu, która jest wykonywana wiersz po wierszu, składa się (obok znaczników i komentarza) z poniższych poleceń, z których każde jest zakończone znakiem średnika.

```
$dataiczas = getdate();
$rok = $dataiczas["year"];
$miesiac = $dataiczas["mon"];
$dzien = $dataiczas["mday"];
$godziny = $dataiczas["hours"];
$minuty = $dataiczas["minutes"];
$sekundy = $dataiczas["seconds"];
```

Czytelnikowi, który jeszcze nigdy nie pisał programów, powyższe wiersze kodu mogą się wydać podobne do równań poznanych na lekcjach matematyki. W równaniach także występowały liczne zmienne. Wiadomo również, że wyrażenie znajdujące się po lewej stronie znaku równości równało się wyrażeniu znajdującemu się po jego prawej stronie. A zmienną nazywa się symbol, mogący przyjmować różne wartości.

Jednak mimo że wiersze kodu PHP wyglądają jak równania, wcale nimi nie są. Jest to tak zwany *operator przypisania wartości*. A jego działanie jest całkiem proste.

PHP może obliczać lub. konstruować prawą stronę operatora przypisującego wartość. Wynikiem tych obliczeń, opracowań lub procesu konstruowania jest konkretna wartość lub złożona struktura danych.

Po lewej stronie znaku równości standardowo znajduje się *zmienna*. W kodzie PHP jest ona rozpoznawana po rozpoczynającym ją znaku dolara `$` (patrz: ramka *Zmienne*). Może przyjmować lub przechowywać pojedyncze wartości lub też skomplikowane struktury danych. Operator przypisuje danej zmiennej konkretną wartość lub strukturę danych, które odpowiadają wyrażeniu znajdującemu się po prawej stronie znaku równości.

Zmienna przechowuje tę wartość lub strukturę danych do momentu przypisania jej nowej wartości (lub struktury danych) albo do zakończenia wykonywania skryptu.

W opisywanym przykładzie pierwsze polecenie wykonuje operację wywołania tzw. *funkcji*. W tym przypadku jest to funkcja `getdate()`, będąca jedną z wielu funkcji wbudowanych w PHP. Funkcje zostaną dokładniej omówione nieco później. W tym momencie wystarczy zapamiętać, że funkcja w kodzie PHP — po jej wykonaniu — zawsze zastępowana jest zwracaną przez nią wartością.

Funkcja `getdate()` zwraca tzw. *tablicę asocjacyjną* (*array*) do interpretera PHP. Tablica tego typu jest przykładem nieco bardziej kompleksowej struktury danych. Za chwilę opiszemy ją nieco dokładniej. W prezentowanym poleceniu przypisującym wartość najpierw zapisano tablicę w zmiennej PHP `$dataiczas`. W ten sposób zmienna `$dataiczas` sama staje się tablicą asocjacyjną.

Tablica asocjacyjna przechowuje wartości danych każdorazowo pod tzw. kluczem (*key*). Dzięki większej liczbie kluczy można w jednej tablicy (a zatem w jednej zmiennej) przechowywać dowolną ilość danych. Klucz może składać się z dowolnego ciągu znaków (*string*). W opisywanym przypadku funkcja `getdate()` sprawia, że pod kluczem `hours` jest przechowywana liczba oznaczająca aktualną godzinę, a pod kluczem `mday` — dany dzień miesiąca (np. 23). W zasadzie sprawa jest prosta, czyż nie?

Być może Czytelnik zastanawia się, w jaki jednak sposób wydobyć konkretne wartości z tablicy asocjacyjnej? O tym za chwilę. Dla wygody interesujące wartości tablicy zapisuje się w pojedynczych zmiennych (`$rok`, `$miesiac` itd.). Dzięki temu nie trzeba później odwoływać się do asocjacyjnej składni tablicy.

Polecenia, które powodują przekopiowanie wartości tablicy do pojedynczych zmiennych, ilustrują odpowiednią składnię tablicy, np. w poleceniu:

```
$rok = $dataiczas["year"];
```

W ten sposób można wydobyć z tablicy `$dataiczas` informację o roku: należy umieścić wartość ze zbioru klucza (w naszym przypadku jest to ciąg znaków `"year"`) w nawiasie kwadratowym, bezpośrednio po nazwie tablicy. Wyrażenie to zostaje przez PHP rozpoznane jako *element* tablicy, który został zapisany pod odpowiednią wartością klucza.

Zmienne

Definiowanie zmiennych w PHP jest bardzo proste. Są one łatwo rozpoznawalne. Nazwa zmiennej rozpoczyna się zawsze znakiem dolara `$`, drugim znakiem może być litera lub znak podkreślenia (*underscore*). Nie należy używać znaków diakrytycznych. W dalszej kolejności może występować dowolna kombinacja liter i cyfr od 0 do 9. Oto przykłady kilku dozwolonych w PHP nazw zmiennych:

```
$kiwi     $_kiwi    $_0    $a
$kiwi2    $a_b_c    $a1
```

Autor zaleca, aby nazwy zmiennych rozpoczynać od litery. Pozwala to na łatwe odróżnienie ich od nazw zmiennych, które zostały z góry zdefiniowane w PHP — takie zmienne rozpoczynają się bowiem od znaku podkreślenia.

Zmiennym mogą zostać przypisane różne wartości, przy czym nie ma znaczenia, czy jest to wartość liczbowa, czy ciąg znaków (*string*). Zmienna

```
$skladnik1 = 3;
```

oraz:

```
$skladnik1 = "3"
```

są sobie równoważne. Należy jednakże uważać podczas wykorzystywania zmiennych w wyrażeniach matematycznych, np. w przypadku operacji dodawania:

```
$skladnik1 = 3; $skladnik2 = "4"; $suma = $skladnik1 + $skladnik2;
```

Powyższe wyrażenie zostanie wykonane poprawnie (`$suma` otrzyma wartość 7), jednak tak nie będzie w przypadku:

```
$skladnik1 = 3; $skladnik2 = "cztery"; $suma = $skladnik1 + $skladnik2;
```

Natomiast możliwe jest łączenie ciągów znaków za pomocą operatora kropki (`.`):

```
$wierzcholki = 3;
$liczba_kapelusz_wierzcholki = "Mój kapelusz ma ".$wierzcholki."
wierzchołki!";
```

W ten sposób zmiennej `$liczba_kapelusz_wierzcholki` zostaje przypisana wartość `"Mój kapelusz ma 3 wierzchołki!"`.

Dane wyjściowe jako złożony ciąg znaków

Czytelnik już wie, jak utworzyć tablicę, w jaki sposób ją skopiować oraz w jaki sposób niektóre z jej elementów zapisać w pojedynczych zmiennych. Cały ten proces zachodzi podczas wykonywania kodu i jest niewidoczny dla użytkownika. Pierwszy fragment kodu PHP nie powodował tworzenia żadnych danych wyjściowych, zatem nie zostanie uwzględniony w danych wyjściowych wysyłanych do przeglądarki klienta. Dopiero po wykonaniu pierwszej części plik PHP generuje nieco kodu HTML: znaczniki `<html>`, `<head>` i `<title>`, jak również nieco tekstu. Zostaną one później wysłane do przeglądarki bez żadnych zmian.

Następnym znacznikiem w omawianym pliku jest:

```
<?php echo $dzien.".".$miesiac.". anno domini ".$rok; ?>
```

Znacznik ten zawiera tylko jedno polecenie PHP, ale jego działanie jest bardzo szerokie. Polecenie `echo` powoduje wysłanie do przeglądarki ciągu znaków (*string*), który może się składać z:

1. jednej zmiennej. Przykładowo, ciąg znaków może stanowić zarówno `$dzien`, `$miesiac` , jak i `$rok`;
2. jawnego ciągu znaków, który w celu odróżnienia od pozostałego kodu PHP jest otoczony znakami apostrofu lub cudzysłowu:

   ```
   ' anno domini '
   ```

   lub:

   ```
   " anno domini "
   ```

   Obie wersje są poprawne.
3. wartości zwracanej przez funkcję;
4. kombinacji ciągów znaków trzech uprzednio wymienionych typów, przy czym dwa sąsiednie ciągi są za każdym razem łączone za pomocą operatora kropki.

Łatwo zgadnąć, jaka wartość tutaj występuje: oczywiście jest to złożony ciąg znaków, składający się z podciągów: `$dzien`, `"."`, `$miesiac`, `" anno domini "` i `$rok`.

Wartości zmiennych są znane z pierwszego fragmentu omawianego kodu PHP. Gdyby `$dzien` miał wartość 23, `$miesiac` wartość 2 a `$rok` wartość 2003, polecenie `echo` wstawiłoby w tym miejscu — zamiast znacznika PHP — do pliku ciąg znaków `"23.2. anno domini 2003"`.

Łatwo się domyślić, co się teraz stanie. W ramach ćwiczenia można przeanalizować kod trzeciego znacznika PHP w pliku *data.php*. Nie powinno tam być nieznanych Czytelnikowi elementów.

Co widzi przeglądarka

Warto wreszcie przekonać się, jakie dane ostatecznie serwer wysyła do przeglądarki.

Listing 4.2. Typowe dane wyjściowe HTML z pliku data.php

```
<html>
  <head>
    <title>
      Dzisiaj jest 23.2. anno domini 2003
    </title>
  </head>
  <body>
    <h1>Na tym serwerze jest godzina 14:22:37
    </h1>
  </body>
</html>
```

Jak można zauważyć, w danych przesyłanych do przeglądarki nie ma kodu PHP. I tak powinno być, gdyż przeglądarka nie służy do interpracji PHP. Kodem, który pozostał jest czysty HTML — najzupełniej zrozumiały dla przeglądarki.

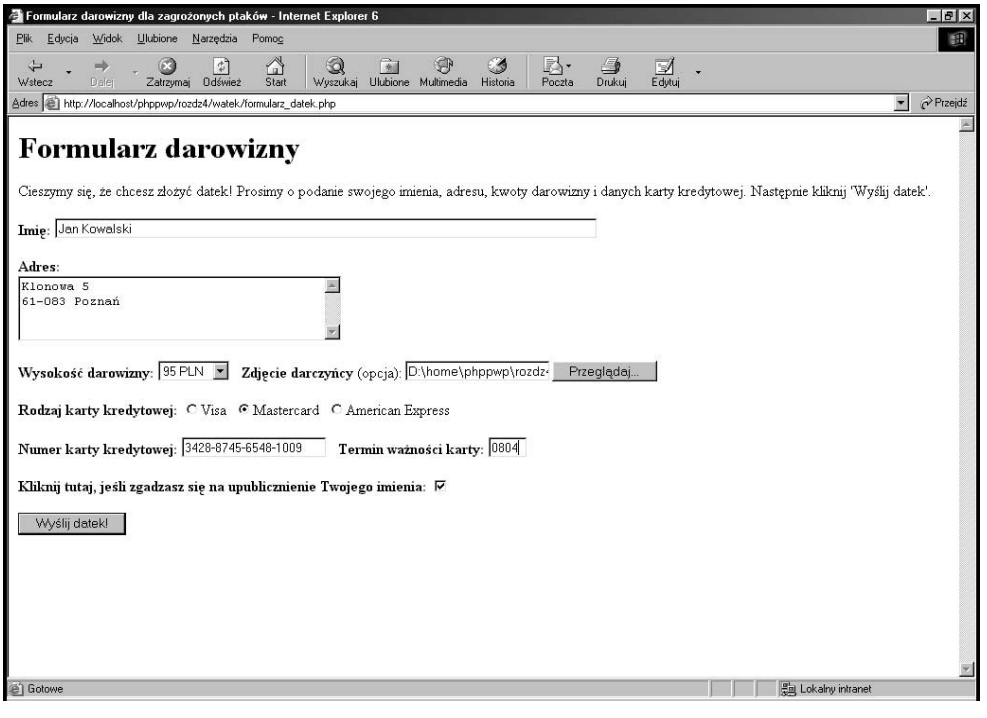
Jeśli Czytelnik wypróbuje powyższy przykładowy kod w praktyce, szybko zauważy mały wizualny mankament: brakuje wygodnego wyświetlania wartości minut i sekund w formacie dwucyfrowym. W końcowej części podrozdziału *Przetwarzanie danych z formularza w skryptach PHP* umieszczono w ramce *Zrób to sam!* małe ćwiczenie, dzięki któremu można rozwiązać ten problem.

Formularze dla danych wprowadzanych przez użytkownika

Główną przyczyną popularności języków skryptowych, takich jak PHP, jest fakt, że serwer może odpowiednio reagować na dane wprowadzone przez użytkownika przeglądarki, np. na zamówienie złożone z internetowym sklepie wysyłkowym. Najpierw użytkownik musi wypełnić formularz wyświetlany w oknie przeglądarki. Dane z tego formularza są wysyłane do serwera. Przeglądarka żąda zazwyczaj wysłania takiego formularza przez serwer. Mimo iż formularze HTML nie są elementem języka PHP, zostały jednak pomyślane głównie dla zastosowań związanych z językami skryptowymi.

Teraz należy powrócić do projektu, którego wątek będzie przewijał się przez całą książkę. Stronę główną tej witryny Czytelnik już poznał. Po kliknięciu odnośnika *Chcę złożyć datek* użytkownik będzie oczekiwał wyświetlenia strony, na której będzie mógł podać swoje dane osobowe i szczegóły dotyczące darowizny — zatem formularza.

Na rysunku 4.2 pokazano formularz służący do składania darowizny.



Rysunek 4.2. Wygląd formularza służącego do składania darowizny

Na powyższym rysunku widać pewne elementy formularza, z których cześć może być już znana użytkownikom internetu: pole edycyjne (do wpisywania imienia darczyńcy), pole tekstowe (w celu wpisania adresu), rozwijana lista wyboru wysokości darowizny, pole wyboru pliku, umożliwiające dołączanie zdjęcia szlachetnego darczyńcy, pole przełącznika typu radio, służące do określania rodzaju karty kredytowej. Zapewniono również dwa pola edycyjne przeznaczone do wpisywania numeru karty kredytowej i daty jej ważności. Wreszcie za pomocą pola wyboru darczyńca może wyrazić zgodę na publikację swojego imienia i nazwiska.

Po załadowaniu formularza do przeglądarki można zobaczyć, że jego twórca z góry założył wyrażenie takiej zgody — zatem darczyńcy chcący pozostać anonimowi muszą aktywnie wyrazić sprzeciw przez odznaczenie omawianego pola wyboru.

W celu umożliwienia sfinalizowania operacji umieszczono także przycisk zatwierdzania z napisem *Wyślij datek*.

Kod formularza darowizny

Czytelnik zapewne chciałby się przekonać, w jaki sposób utworzono opisywaną stronę w języku PHP i co też jeszcze ciekawego można znaleźć w jej kodzie.

Listing 4.3. Kod strony formularz_datek.php

```
<html>
  <head><meta http-equiv="content-type" content="text/html; charset=iso8859-2">
    <title>Formularz darowizny dla zagrożonych ptaków</title>
  </head>
  <body>
    <form name="datek" action="datek.php" method="post"
                enctype="multipart/form-data">
      <input type="hidden" name="godzina"
        value="<?php echo time();?>">
      <h1>Formularz darowizny</h1>
      Cieszymy się, że chcesz złożyć datek! Prosimy o podanie swojego
      imienia, adresu, wysokości darowizny i dane karty kredytowej.
      Następnie kliknij 'Wyślij datek'.
      <p>
      <b>Imię:</b> <input type="text" name="imiedarczyncy" size="80"><p>
      <b>Adres:</b><br>
      <textarea name="adres" rows="4" cols="40" align="top"></textarea>
      <p>
      <b>Wysokość darowizny:</b>
      <select name="kwota">
        <?php
          for($i = 5; $i < 101; $i = $i + 5) {
            echo "<option value=\"".$i."\">".$i."PLN\n";
          }
        ?>
      </select>
       
      <b>Zdjęcie darczyńcy</b> (opcja): <input name="zdjeciedarczyncy"
         type="file">
      <p>
      <b>Rodzaj karty kredytowej:</b>
      <input type="radio" name="kk_rodzaj" value="Visa">Visa
             <input type="radio" name="kk_rodzaj"
                   value="Mastercard">Mastercard
             <input type="radio" name="kk_rodzaj" value="American
                   Express">American Express
      <p>
      <b>Numer karty kredytowej:</b> <input type="text" name="kk_numer"
         size="20" maxlength="20">
       
      <b>Data ważności karty:</b> <input type="text" name="kk_data" size="4"
         maxlength="4">
      <p>
      <b>Kliknij tutaj, jeśli zgadzasz się na upublicznienie Twojego
         imienia:</b>
      <input type="checkbox" name="publiczny" checked>
      <p>
      <input type="submit" value="Wyślij datek!">
    </form>
  </body>
</html>
```

Nagłówek pliku nie powinien zawierać żadnych niespodzianek. Do znacznika `<body>` wszystko powinno być znajome. Natomiast zaraz po nim rozpoczyna się kod formularza.

Jak formularz staje się formularzem: znacznik <form>

Aby utworzyć prawdziwy formularz, trzeba otoczyć znacznikiem `<form>` wszystkie elementy służące do wprowadzania danych. Dla jednego formularza może istnieć tylko jeden

taki znacznik, może on również zawierać inne elementy strony (przykładowo, nagłówek i tekst opisu). Jednak równie dobrze można by umieścić znacznik `<h1>` poza obrębem znacznika `<form>`.

Atrybuty znacznika `<form>` określają nazwę formularza (jest to właściwość, którą można wykorzystać później tworząc połączenie z Javascript, co pozwala na dodatkowe rozszerzenie funkcjonalności formularza), do jakiego dokumentu skryptowego mają zostać przesłane dane z wypełnionego formularza i czy do tego celu zostanie wykorzystane żądanie HTTP typu `GET` czy `POST`. Z uwagi na możliwość przesłania do serwera pliku graficznego (zdjęcia darczyńcy), należy ustawić atrybut `enctype` na `multipart/form-data`.

Element ukryty

Następnym elementem jest nieco dziwna konstrukcja, z którą Czytelnik mógł się nie spotkać do tej pory.

```
<input type="hidden" name="godzina"
value="<?php echo time();?>">
```

Jest to pole tekstowe typu `hidden` (ukryte) — pozostaje celowo niewidoczne w oknie przeglądarki. W opisywanym przykładzie zostanie wykorzystane do zapamiętania czasu wyświetlania formularza. W tym celu można wykorzystać funkcję PHP `time()`, podającą liczbę sekund, które upłynęły od dnia 1 stycznia 1970, godziny 00:00:00 GTM. Wartość jest zapisywana w atrybucie `value` i zostanie przesłana do serwera pod nazwą `godzina`.

Elementy ukryte są bardzo często używane w praktyce, ponieważ umożliwiają serwerowi przechowanie części danych w przeglądarce podczas wypełniania formularza przez użytkownika i ich ponowne odczytanie po otrzymaniu danych z przesłanego do przetworzenia formularza. Dzięki temu można rozłożyć proces wpisywania danych na kilka formularzy i dopisywać do kolejnego formularza dane już przesłane przez użytkownika.

Elementy ukryte można wykorzystać również do identyfikacji i autoryzacji użytkownika, czy też do badania jego zachowań. Przykładowo, można by się dowiedzieć, ile przeciętnie czasu potrzebuje użytkownik na wypełnienie formularza. Być może warto by także wiedzieć, czy darczyńcy ofiarujący większe sumy potrzebują więcej czasu na podjęcie takiej decyzji. Sygnatura czasu może być pomocna również przy aktualizacji oprogramowania, tak aby starsze formularze były przez skrypt traktowane inaczej.

Jednakże stosowanie pól ukrytych do identyfikacji użytkownika może stać się przyczyną zagrożenia bezpieczeństwa — ten problem omówimy bardziej szczegółowo w rozdziale 7.

W opisywanym przykładzie element ukryty służy właściwie tylko do tego, aby go pokazać: w zasadzie jest to zwykły element tekstowy, posiadający z góry ustaloną i niezmienną wartość. Gdy dane z tego elementu (nazwa i wartość) trafią do serwera, będą interpretowane tak samo jak inne elementy tekstowe.

Pole tekstowe

Kolejnym elementem formularza w analizowanym przykładzie jest pole tekstowe służące do wprowadzania danych:

```
<input type="text" name="imiedarczyncy" size="80">
```

Użytkownik powinien samodzielnie wpisać wartość tego pola, zatem nie została określona domyślna wartość atrybutu `value`. Ustalono jednak szerokość pola tekstowego wyrażoną jako liczba znaków — można tego dokonać podając wartość atrybutu `size`. Warto zapamiętać, że atrybut `size` określa wielkość pola tekstowego, jaka zostanie wyświetlona w przeglądarce. Nie powoduje to żadnych ograniczeń związanych z długością wprowadzanego tekstu, którym może być, przykładowo, imię darczyńcy, składające się np. z 80 znaków.

Aby zapewnić możliwość wprowadzania danych wielowierszowych, należy posłużyć się polem `<textarea>`. Element ten różni się nieco od zwykłego znacznika `<input>`:

```
<textarea name="adres" rows="4" cols="40" align="top"></textarea>
```

Podobnie jak `<input>` , także `<textarea>` dysponuje nazwą, która razem z wprowadzonym tekstem jest wysyłana do serwera. Wielkość pola jest określana za pomocą atrybutów `rows` (rzędy) i `cols` (kolumny). Element nie posiada atrybutu `value`. Natomiast tekst domyślny wpisuje się między znacznikiem otwierającym a zamykającym, np.

```
<textarea name="adres" rows="4" cols="40" align="top">
Tutaj proszę wpisać swój adres.
</textarea>
```

Lista rozwijana <select>

Kolejnym elementem formularza jest lista rozwijana `<select>`. W statycznym dokumencie HTML kod tego elementu wyglądałby następująco:

```
<select name="kwota">
  <option value="5">5 PLN
  <option value="10">10 PLN
  ...
  <option value="100">100 PLN
</select>
```

W prezentowanym przypadku znacznik `<select>` określa tylko nazwę, wraz z którą wybrana wartość zostanie wysłana do serwera w celu przetworzenia. Wewnątrz znacznika `<select>` znajdują się znaczniki `<option>`. Atrybut `value` znacznika `<option>` określa wartość, która zostanie wysłana do serwera, jeśli użytkownik wybierze daną opcję. Tekst występujący po znaczniku `<option>` jest tekstem, który zostanie wyświetlony na liście możliwych do wyboru wartości obok danej opcji. Chcąc ustawić pewną opcję jako domyślną, w odpowiednim znaczniku `<option>` należy umieścić atrybut `selected` bez przypisywania mu żadnej wartości, np.: `<option value="50" selected>`.

Generowanie znaczników <option> w PHP za pomocą pętli for

W opisywanym przykładzie zastosowanie PHP pozwala na ograniczenie ilości wpisywanego kodu. Warto przeanalizować fragment pliku *formularz_datki.php* zaprezentowany na listingu 4.1.

Listing 4.4. Automatyczne generowanie listy wyboru w pliku formularz_datki.php

```
<select name="kwota">
 <?php
  for($i = 5; $i < 101; $i = $i + 5) {
    echo "<option value=\"".$i."\">".$i." PLN\n";
   }
 ?>
 </select>
```

Konstrukcją ułatwiającą pracę jest tzw. pętla `for`. Programiści używają jej w przypadku, gdy jakiś proces ma zostać wykonany wielokrotnie a liczba cyklów (zwanych *iteracjami*) jest z góry określona.

W naszym przypadku procesem tym jest wyświetlenie znacznika `<option>` dla wszystkich kwot pomiędzy 5 a 100 zł, ze skokiem co 5 zł. Można tego dokonać za pomocą instrukcji przetwarzania pętli `for`. Instrukcje przetwarzania znajdują się w okrągłych nawiasach, bezpośrednio po poleceniu `for`: `($i = 5; $i < 101; $i = $i + 5)`. W opisywanym bloku można wyróżnić trzy, rozdzielone znakami średnika, części składowe.

Pierwsza z nich zawiera (w niektórych przypadkach oddzielone znakami przecinka) instrukcje, które mają zostać wykonane przed pierwszym przebiegiem pętli. Oznacza to, że pierwsza część bloku kodu służy do ustalenia warunków wstępnych (inicjalizacja). W opisywanym przykładzie zmienna `$i` otrzymuje wartość `5` i reprezentuje te kwoty, które zostaną przypisane do danych opcji. Programiści nazywają zmienną `$i` *zmienną iteracyjną pętli for*.

Druga część opisywanego fragmentu kodu zawiera warunek, który musi zostać spełniony na początku każdego cyklu, aby pętla mogła zostać ponownie wykonana. Założono, że wartość zmiennej `$i` nie może przekroczyć `100`, zatem warunkiem wykonania następnej iteracji jest, że `$i` ma być mniejsze niż `101`. W przeciwnym razie pętla nie zostanie wykonana.

Trzecia część instrukcji przetwarzania pętli zostaje wykonana przed zakończeniem każdego cyklu i ma na celu sprawdzenie warunku zawartego w drugiej części. Ten blok kodu składa się z instrukcji, które przeważnie służą do zmiany wartości zmiennej iteracyjnej. W opisywanym przykładzie jest podobnie: kwota zawarta w zmiennej iteracyjnej `$i` jest każdorazowo powiększana o `5`. W tym celu dodaje się `5` do poprzedniej wartości `$i` a wynik ponownie zapisuje się w zmiennej `$i`.

Polecenia, które mają zostać wykonane w każdej iteracji, można umieścić w trzeciej części polecenia w nawiasie okrągłym. Przyjęło się jednak umieszczać je w nawiasie klamrowym. Polecenia te są wykonywane zaraz po instrukcjach zawartych w nawiasie okrągłym.

W prezentowanym przypadku jest to tylko jedno polecenie, które zajmuje dwa wiersze — jest to możliwe, gdyż ciągi znaków w PHP mogą być dłuższe niż jeden wiersz.

```
echo "<option
          value=\"".$i."\">".$i." PLN\n";
```

Zrozumienie części składowych polecenia `echo` nie powinno sprawić Czytelnikowi problemów. Ciąg znaków, który zostanie wyświetlony na stronie, składa się z pięciu podciągów:

```
echo "<option value=\""
$i
"\">"
$i
" PLN\n";
```

Elementem jeszcze nieznanym Czytelnikowi może być konstrukcja `\"`. Jest to tzw. sekwencja unikowa (ang. *escape sequence)*, którą tworzy się za pomocą znaku lewego ukośnika. Rola sekwencji unikowych jest bardzo prosta do zrozumienia: ciąg znaków jest ograniczany z obu stron za pomocą cudzysłowu, zatem drugi znak cudzysłowu występujący w ciągu znaków jest interpretowany jako znak zamykający ten ciąg. W ten sposób, jeśli elementem danego ciągu znaków jest cudzysłów, należy go oznaczyć, tak aby nie został zinterpretowany jako zakończenie danego ciągu znaków. W tym celu przed takim cudzysłowem wstawia się znak lewego ukośnika — stanowi to informację, że dany znak należy do właściwego ciągu znaków.

Drugi i czwarty z powyższych ciągów znaków są kopią zmiennej `$i`. Należy zwrócić uwagę na to, że w instrukcjach przetwarzania pętli zapisano w zmiennej `$i` najpierw wartość numeryczną, którą następnie wykorzystano jako ciąg znaków. To niewidoczne przejście z wartości numerycznej do odpowiadającego jej ciągu znaków (tzw. niejawna konwersja typu zmiennych) jest jedną z bardzo wartościowych cech PHP.

W piątym ciągu znaków znajdują się dwa elementy wymagające krótkiego komentarza: `PLN` jest zwykłym tekstem, oznaczającym polską walutę, natomiast `\n` oznacza koniec wiersza. Zabieg ten nie ma wprawdzie wpływu na poprawność działania kodu, jednakże znacznie ułatwia wyszukiwanie ewentualnych błędów w kodzie źródłowym strony. Gdyby kod wszystkich opcji został zapisany w jednym wierszu, to po wybraniu w oknie przeglądarki opcji *Pokaż źródło* szybkie odnalezienie właściwej opcji wśród dwudziestu byłoby utrudnione.

Przesyłanie pliku do serwera

Następnym po elemencie `<select>` jest pole, umożliwiające użytkownikowi przesłanie (ang. *upload*) swojego zdjęcia do serwera. Właściwie wszystko jest oczywiste:

```
<input name="zdjeciedarczyncy" type="file">
```

Ważnym jest pamiętanie o pewnym szczególe. Jak już wcześniej wspomniano, aby umożliwić przekazywanie plików do serwera za pomocą formularza, należy ustalić wartość atrybutu `enctype` znacznika `form` na `multipart/form-data`. Komplikuje to nieco obsługę pliku graficznego po stronie serwera. Problemem tym zajmiemy się jednak później.

Przycisk opcji

Kolejnym krokiem jest określenie przez użytkownika typu karty kredytowej. Można by to zrobić za pomocą znacznika `<select>`, tak jak to miało miejsce w przypadku wskazania kwoty darowizny. Alternatywnym rozwiązaniem jest zastosowany tutaj przycisk opcji (tzw. przycisk typu radio — *radio button*):

Listing 4.5. Przyciski opcji w pliku formularz_datki.php

```
<input type="radio" name="kk_rodzaj" value="Visa">Visa
 <input type="radio" name="kk_rodzaj" value="Mastercard">Mastercard
 <input type="radio" name="kk_rodzaj" value="American Express">American
Express
```

Przyciski opcji swoją angielską nazwę zawdzięczają przyciskom, które są znane ze starych odbiorników radiowych: po wciśnięciu przycisku należącego do danej grupy każdy uprzednio wciśnięty powraca do pozycji wyjściowej. W przypadku przycisków opcji jest identycznie: po kliknięciu jednego z nich następuje jego zaznaczenie z równoczesnym odznaczaniem wszystkich pozostałych należących do tej samej grupy.

Każdy z przycisków w danej grupie jest samodzielnym elementem oznaczonym osobnym znacznikiem `<input>`. Łączenie znaczników w grupy odbywa się za pomocą atrybutu `name`: przyciski mające tą samą nazwę należą do jednej grupy. W opisywanym przypadku są to trzy przyciski, przypisane do grupy `kk_rodzaj`. Atrybut `value` służy do określania wartości, która po zaznaczeniu danego przycisku zostanie wysłana do serwera jako element wypełnionego formularza. Znak ` ` (niełamliwa spacja) pozwala na zachowanie pewnego minimalnego odstępu pomiędzy przyciskami opcji.

Kolejny element służący do wprowadzania danych, pole służące do wpisywania numeru karty kredytowej, powinien być jasny dla Czytelnika. Omówić tutaj należy jedynie atrybut `maxlength`, określający maksymalną liczbę znaków, jaką użytkownik może wpisać w pole. Parametr ten stanowi jednak tylko wskazówkę dla przeglądarki — nie należy się spodziewać, że w skrypcie ostatecznie przetwarzającym formularz pod nazwą `kk_numer` zostanie zwrócony rzeczywiście ciąg składający się z maksymalnie 20 znaków. Podobnie ma się sprawa z elementem służącym do podawania terminu ważności karty.

Pole wyboru — być albo nie być?

Ostatni element omawianego formularza ma kształt kratki i służy do określenia, czy darczyńca decyduje się na publikację swojego imienia. Jest to pole wyboru, tzw. *checkbox*:

```
<input type="checkbox" name="publiczny" checked>
```

Pola wyboru są samodzielnymi elementami, których w przeciwieństwie do przycisków opcji nie łączy się w grupy. Natomiast oba elementy umożliwiają wysyłanie do serwera pewnych wartości za pomocą atrybutu `value`. Jeśli pole wyboru nie zostanie zaznaczone, żadna wartość nie jest wysyłana. Atrybut `value` może zostać pominięty — jeśli pole zostało zaznaczone, przeglądarka przesyła ciąg znaków `on`.

Atrybut `checked` sprawia, że pole wyboru przy ładowaniu formularza zostaje od razu zaznaczone.

Ostatnim elementem formularza jest przycisk `submit`. Kliknięcie tego przycisku powoduje wysłanie formularza do serwera w celu przetworzenia:

```
<input type="submit" value="Wyślij datek!">
```

Atrybut `value` w tym przypadku oznacza opis przycisku.

To właściwie prawie wszystkie ważne elementy formularzy udostępniane przez język HTML i umożliwiające wprowadzanie danych. Tabela 4.1 zawiera zestawienie ich najważniejszych cech.

W ten sposób można przygotować formularz służący do przesyłania danych do serwera. Warto teraz zapoznać się ze sposobem przetwarzania takich przesłanych danych przez serwer.

Zrób to sam!

Wcześniej omówione elementy formularza umożliwiają rozbudowę prezentowanego formularza, np. o pole tekstowe, przeznaczone do wpisywania nazwiska, znajdującego się na karcie kredytowej. Poza tym można zachęcić ofiarodawcę do składania wielokrotnych datków, umieszczając na formularzu listę wyboru, pozwalającą darczyńcy na określenie częstotliwości składania datków: tylko raz (co 0 dni), co tydzień (co 7 dni), co miesiąc (co 30 dni), co kwartał (co 90 dni) lub co rok (co 365 dni).

Rozwiązanie tego problemu można znaleźć (podobnie jak w przypadku innych plików opisywanych w tym rozdziale) w dostępnym on-line pliku *formularz_datki_rozszerzony.php*.

Przetwarzanie danych z formularza w skryptach PHP

Po wypełnieniu formularza przez użytkownika i po kliknięciu przycisku zatwierdzania (*submit*) wprowadzone dane są przesłane do skryptu podanego w atrybucie `action` znacznika `form`. Wartość atrybutu `action` jest interpretowana przez przeglądarkę jako adres URL (adres sieci WWW). W razie niepodania żadnego protokołu i adresu serwera, tak jak zachodzi w przypadku opisywanego pliku, domyślnie jest wykorzystywany protokół HTTP a skrypt PHP jest poszukiwany na tym samym serwerze, na którym znajduje się dokument zawierający formularz.

Tabela 4.1. Ważniejsze znaczniki HTML służące do tworzenia formularza

Znacznik	Ważniejsze atrybuty	Znaczenie	Uwagi
`<input type="text">`	`name`, `id`, `value`, `size`, `maxlength`	jednowierszowy element służący do wprowadzania tekstu	
`<input type="hidden">`	`name`, `id`, `value`	pole ukryte	Wartość zostaje ustalona za pomocą atrybutu `value` przez serwer lub Javascript.
`<textarea>`	`name`, `id`, `rows`, `cols`	pole tekstowe	Wartość domyślna jest ustawiana między znacznikiem otwierającym a zamykającym.
`<select>`	`name`, `id`	lista wyboru	Zawiera znacznik `<option>` z poszczególnymi opcjami wyboru.
`<option>`	`value`, `selected`	pojedyncze opcje wyboru, elementy listy wyboru	Wyświetlana w oknie wartość jest podawana po znaczniku.
`<input type="file">`	`name`, `id`	pole wyboru pliku przeznaczonego do przesłania do serwera	Nie można ustalić wartości domyślnej; znacznik `form` wymaga podania `enctype="multipart/form-data"`.
`<input type="radio">`	`name`, `id`, `value`, `checked`	przełącznik	Przełączniki (*radio buttons*) są grupowane za pomocą wspólnego dla wszystkich elementów atrybutu `name`.
`<input type="checkbox">`	`name`, `id`, `value`, `checked`	pole wyboru	W przypadku niezaznaczonego pola wyboru do serwera nie jest wysyłana żadna wartość.
`<input type="submit">`	`value`	przycisk wysyłający formularz	atrybut `value` służy do opisania przycisku.

Formularz z obcego źródła — uwaga, pułapka!

Dzięki stosowaniu formatu URL w atrybucie `action` można wysyłać dane do skryptu, który znajduje się na innym serwerze. W niektórych przypadkach jest to bardzo pożądane — przykładowo, można umieścić na swojej stronie formularz znanej wyszukiwarki, który po wypełnieniu przez użytkownika zostanie do niej wysłany.

Istnieje jednak pewien problem, o którym powinien pamiętać każdy programista PHP (lub ASP, JSP, CGI itd.): nie można zagwarantować, że otrzymane przez skrypt dane będą pochodziły z prawidłowego formularza. Tym samym istnieje ryzyko zagrożenia bezpieczeństwa serwera.

Nie można zakładać, że wszyscy odwiedzający daną witrynę użytkownicy internetu mają dobre intencje. Programiści często zapominają o hakerach, chcących pozyskać numery kart kredytowych ofiarodawców lub wykorzystać dany serwer jako platformę do rozproszonego ataku typu „denial-of-service" na inne serwery. Użytkownikami internetu są różne osoby o bardzo różnych motywach działania, zatem zawsze przy pisaniu skryptów należy zachować ostrożność.

Skrypt, który przyjął nieprawidłowe dane, może zachować się w sposób odmienny od życzeń swojego twórcy, na przykład może:

- wypełnić bazę danych sprzecznymi danymi i ją zniszczyć;
- pozwolić osobom nieupoważnionym na uzyskanie dostępu do bazy, na odczytanie, zmianę lub usunięcie danych. Cały ten proceder jest właściwie niewidoczny;
- rozsyłać — wykorzystując dany serwer — pocztę elektroniczną, zawierającą dane systemowe lub inne poufne pliki czy dane;
- uruchamiać na serwerze programy lub — w pewnych okolicznościach —instalować nowe;
- zastąpić całkowicie lub częściowo daną stronę WWW nowymi, wybranymi przez hakera treściami;
- unieruchomić serwer;
- zamknąć właścicielowi witryny dostęp do własnych danych.

Nie są to problemy typowe jedynie dla PHP. Dotyczą bowiem wszystkich technologii WWW, które opierają się na interakcji danych wprowadzanych przez użytkownika z procesami przebiegającymi w serwerze. Podobne zagrożenia dotyczą także CGI, Active Server Pages, Java Server Pages, mod_perl, serwletów itd.

Liczba serwerów, których zabezpieczenia zostały złamane dzięki wykorzystaniu luk w skryptach, jest znaczna. Wiele z tych ataków dotyczy technologii CGI, gdyż niektóre jej dystrybucje posiadają pewne luki w systemie zabezpieczeń. Nawet niezbyt uzdolniony haker wyszuka adresy internetowe i serwery zawierające te (często niestosowane przez właściwych użytkowników) skrypty. To samo dotyczy licznych skryptów, które są oferowane bezpłatnie do pobrania.

Jak powstają takie luki w zabezpieczeniach systemów, których nie wykrywają nawet specjaliści? Istnieją dwie przyczyny takiego stanu rzeczy: Po pierwsze, głównym celem programisty jest uruchomienie nowego programu. Dlatego przeważnie zwraca uwagę tylko na to, aby program poprawnie wykonywał to, do czego został stworzony. Kwestia, że program nie powinien pracować w nieprzewidziany sposób pozostaje drugoplanowa. Drugim powodem jest duży stopień skomplikowania nowoczesnych języków programowania: niektóre polecenia mogą w pewnych trudnych do określenia sytuacjach powodować zupełnie inne zachowanie systemu, niż życzyłby sobie tego programista.

Nie należy się jednak zniechęcać do samodzielnego programowania w PHP. Wprost przeciwnie: pamiętając o tym ostrzeżeniu, będzie można zadawać pytania dotyczące bezpieczeństwa i zwracać uwagę na zagrożenia i stosować środki zapobiegawcze. Należy starać się unikać typowych błędów, często popełnianych przez początkujących programistów i tworzyć właściwie zabezpieczone skrypty.

Warto wspomnieć, że kolekcję często powtarzających się pytań dotyczących bezpieczeństwa serwerów i skryptów można znaleźć w *WWW-Security FAQ* autorstwa Lincolna Steina i Johna Stewarta pod adresem *http://www.w3.org/Security/faq/www-security-faq.html*.

Przesyłanie danych do skryptu i ich kontrola

Użytkownik wypełnił zatem formularz i wysłał go z przeglądarki do serwera. W normalnym przypadku powinny teraz nastąpić dwa zdarzenia: przesłane dane powinny zostać zapisane w bazie danych a do przeglądarki powinien zostać odesłany komunikat, przeznaczony dla użytkownika i potwierdzający dokonanie wpisu.

Niezależnie od tego, czy dane pochodzące z formularza zostaną zapisane w bazie danych, czy też zostaną wykorzystane do wygenerowania strony WWW, trzeba je mieć do dyspozycji. Sposób zapisywania danych w bazie danych zostanie omówiony w rozdziale 6. Najpierw Czytelnik się dowie, w jaki sposób na poziomie skryptu uzyskać dostęp do danych przesłanych przez przeglądarkę, sprawdzić ich prawidłowość oraz jak wysłać do użytkownika komunikat zwrotny dotyczący wpisanych przez niego danych lub informujący o błędnych wpisach. Na rysunku 4.3 pokazano przykładowy komunikat zwrotny wysłany przez omawiany skrypt do (fikcyjnego) użytkownika formularza służącego do składania datków.

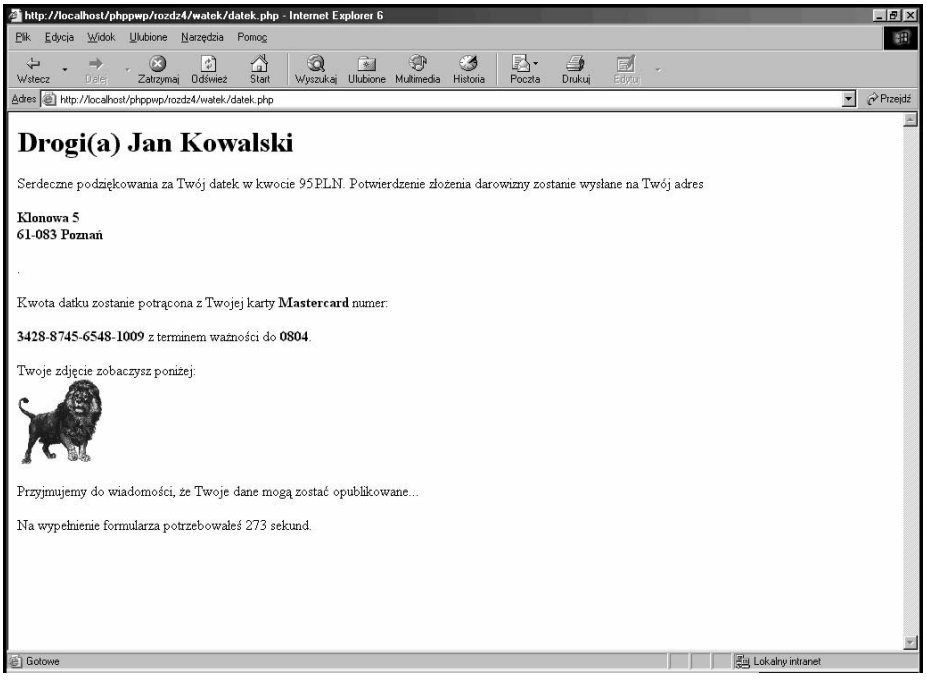
Teraz należy omówić kod zawarty w pliku *datek.php*.

Listing 4.6. Kod programu przetwarzającego dane z formularza w pliku datek.php

```
<?php

  // Ten skrypt przyjmuje dane z formularza.
  // Dane są sprawdzane i następuje wyświetlenie potwierdzenia dla użytkownika.
  $hack = false;        // ta zmienna typu boolean wskazuje, czy
                        // nie mamy do czynienia z działaniem hakera

  // Czytanie danych z $_POST
```



Rysunek 4.3. Potwierdzenie wysłane przez skrypt datek.php

```
  $imiedarczyncy = $_POST["imiedarczyncy"];
  $adres = $_POST["adres"];
  $kwota = $_POST["kwota"];
  $kk_rodzaj = $_POST["kk_rodzaj"];
  $kk_numer = $_POST["kk_numer"];
  $kk_data = $_POST["kk_data"];
  $publiczny = $_POST["publiczny"];
  $godzina = $_POST["godzina"];

  // Weryfikacja danych
  // $imiedarczyncy może być dowolnym ciągiem znaków
  // nie może być tylko ciągiem pustym
  if ($imiedarczyncy == "") {
$hack = true; $pole = "Imię";
  }
  // $adres może być również dowolnym niepustym ciągiem znaków
  if ($adres == "") {
$hack = true; $pole = "Adres";
  }
  // Kwota musi być liczbą całkowitą z zakresu 5 - 100
  if (!preg_match("/^\d*[05]$/", $kwota)) { // nie jest liczbą całkowitą?
$hack = true; $pole = "Kwota";
  }
  if (($kwota < 5) || ($kwota > 100)) {
    $hack = true; $pole = "Kwota";
       }
  // Akceptowane karty to Visa, Mastercard, American Express
  switch ($kk_rodzaj) {
         case "Visa": break;
         case "Mastercard": break;
```

```
        case "American Express": break;
        default:
                $hack = true; $pole = "Rodzaj karty kredytowej";
}
// Numer karty powinien składać się z czterech grup,
// z których każda powinna składać się z 4 cyfr
// Grupy powinny być połączone spacją, łącznikiem lub napisane w jednym ciągu
if (!preg_match("/^(\d{4}[\s\-]?){4}$/", $kk_numer)) {
    $hack = true; $pole = "Numer karty";
}

// Termin ważności karty powinien składać się z 4 cyfr,
// przy czym pierwsze dwie powinny pochodzić z przedziału od 1 do 12
// a trzecia musi równać się 0 (chyba że termin ważności karty
// upływa po roku 2009
// Czwarta cyfra musi pochodzić z przedziału 2 - 9

if (!preg_match("/^(\d{2})0[2-9]$/", $kk_data, $match)) {
    $hack = true; $pole = "Termin ważności karty";
}
else
{
    if (($match[1] < 1) || ($match[1] > 12)) {
         $hack = true; $pole = " Termin ważności karty";
        }
}
// Pole wyboru  może być albo puste, albo zaznaczone (wartość 'on').
if (($publiczny != "") && ($publiczny!= "on")) {
     $hack = true; $pole = "Publikacja danych ofiarodawcy";
}
// Czas wypełnienia formularza musi być liczbą całkowitą
if (!preg_match("/^\d+$/", $godzina)) {
    $hack = true; $pole = " Czas wypełnienia formularza ";
}
   // Czy załączono zdjęcie?
   if ($_FILES["zdjeciedarczyncy"]["size"] > 0) {
      $zdjecie = true;
      preg_match("/(\.\w+)$/",
         $_FILES["zdjeciedarczyncy"]["name"],$match);
      $typ = $match[1];
      // akceptujemy tylko rozszerzenia plików graficznych
      if (in_array(
          strtolower($typ),
          array(".gif",".bmp",".jpg",".png",".jpeg"))) {
          $nazwapliku = uniqid("").$typ;
          $sciezka = preg_replace("/\/[^\/]+$/","",
                     $_SERVER["SCRIPT_FILENAME"])
                     ."/Obrazki/";
          copy($_FILES["zdjeciedarczyncy"]["tmp_name"],
               $sciezka.$nazwapliku);
  }
}
else
{
  $zdjecie = false;
}


// Przy nieprawidłowych danych zwróć komunikat o błędzie:
if ($hack) { ?>
   <html>
     <body>
       <h1>Błędne dane</h1>
```

```
          Twój wpis w polu <b><?php echo $pole; ?></b>
          jest niepoprawny.
        </body>
      </html>
<?php  exit(); // zakończ skrypt!
}

// W tym miejscu dołączymy później kod wpisujący dane do bazy danych
// Wyświetlenie potwierdzenia:
?>
  <html>
    <body>
      <h1>Drogi(a) <?php echo $imiedarczyncy; ?></h1>
      Serdeczne podziękowania za Twój datek w kwocie  <?php echo $kwota ?> PLN.
      Potwierdzenie złożenia darowizny zostanie wysłane na Twój adres <p>
      <b><?php echo preg_replace("/\r?\n/","<br>",$adres); ?></b><p>.<p>
      Kwota datku zostanie potrącona z Twojej karty <b>
      <?php echo $kk_rodzaj; ?></b> numer:<p>
      <b><?php echo $kk_numer; ?></b> z terminem ważności do
      <b><?php echo $kk_data; ?></b>.<p>
          <?php
          if ($zdjecie) { ?>
              Twoje zdjęcie zobaczysz poniżej:<br>
              <img src="<?php echo "Obrazki/".$nazwapliku; ?>"><p>
          <?php
       } ?>
      Przyjmujemy do wiadomości, że Twoje dane
      <?php
        if ($publiczny == "") {
            echo "nie";
        }
      ?>
      mogą zostać opublikowane...<p>
      Na wypełnienie formularza potrzebowałeś <?php echo (time() - $godzina); ?>
      sekund.
    </body>
</html>
```

Jak widać, kod programu jest stosunkowo obszerny. Z tego też powodu będziemy omawiać go stopniowo. Pierwszą wykonaną czynnością jest zdefiniowanie zmiennej `$hack`, której przypisano boolowską wartość logiczną `false`. W ten sposób ustalono, że na początku wykonywania skryptu nie ma przesłanek, aby stwierdzić (umyślne lub nieumyślne) wprowadzenie przez użytkownika nieprawidłowych danych. Bezpośrednio po wykryciu błędnych wpisów wartość zmiennej `$hack` zostanie zmieniona na `true`, aby w ten sposób wskazać pojawienie się problemu. Jeśli po sprawdzeniu wszystkich danych pochodzących z formularza zmienna `$hack` nadal będzie posiadała wartość `false`, będzie to oznaczać, że nie wystąpił żaden rozpoznawalny błąd.

Proste dane formularza, które przesłano do skryptu za pomocą zapytania HTTP metodą `POST`, znajdują się w globalnej tablicy asocjacyjnej noszącej nazwę `$_POST`. Tablica ta jest udostępniana przez PHP automatycznie[1]. W pierwszym kroku należy skopiować zapisane w niej dane do zmiennych, które cechuje większa łatwość obsługi:

[1] Wersje PHP poniżej 4.10 nie posiadają jeszcze `$_POST`. Osoby używające PHP w wersji o numerze niższym niż 4.10 mogą korzystać z ekwiwalentnej tablicy noszącej nazwę `$HTTP_POST_VARS`, która jednak jest dostępna przy odpowiedniej (domyślnej) konfiguracji (`track_vars on` w pliku konfiguracyjnym *php.ini*).

```
$imiedarczyncy = $_POST["imiedarczyncy"];
$adres = $_POST["adres"];
$kwota = $_POST["kwota"];
$kk_rodzaj = $_POST["kk_rodzaj"];
$kk_numer = $_POST["kk_numer"];
$kk_data = $_POST["kk_data"];
$publiczny = $_POST["publiczny"];
$godzina = $_POST["godzina"];
```

Wartości kluczy tablicy asocjacyjnej odpowiadają tutaj nazwom elementów służących do wprowadzania danych formularza. Nazwy te zdefiniowano atrybutem `name`.

Być może Czytelnik zauważył brak danych dotyczących fakultatywnego pliku ze zdjęciem. Otóż pliki, które zostały wysłane do serwera, są obsługiwane przez PHP nieco inaczej. Dokładniejsze informacje na ten temat zostaną podane nieco później. Najpierw warto przyjrzeć się nieco bliżej danym wejściowym. Ma to swoje dobre uzasadnienie: po pierwsze, umożliwi to wysłanie do użytkownika odpowiedniego komunikatu w przypadku odnalezienia błędnego wpisu, po drugie, utrudni to działanie potencjalnym hakerom, ponieważ nieprawidłowe wpisy zostaną wychwycone i odrzucone.

Kontrola imienia i nazwiska

Dokładna kontrola danych wejściowych zależy w głównej mierze od oczekiwań programisty. W niektórych przypadkach jest to bardzo proste, np. ponieważ istnieje tak wiele dozwolonych wpisów, że można dopuścić prawie wszystkie. W przypadku imion i adresów dane mogą zawierać właściwie dowolne znaki. Dlatego należy się tylko upewnić, że przesłany ciąg znaków, który powinien zawierać imię i nazwisko darczyńcy, nie jest pusty:

```
if ($imiedarczyncy == "") {
                $hack = true; $pole = "Imię";
   }
```

Takiej samej kontroli należy poddać także adres. Konstrukcją użytą podczas kontroli jest tzw. instrukcja warunkowa `if`. W PHP składa się ona z zarezerwowanego słowa `if`, warunku (umieszczanego w nawiasie okrągłym) oraz jednego lub wielu poleceń w nawiasie klamrowym. Jeśli warunek zostanie spełniony, nastąpi wykonanie poleceń umieszczonych w nawiasie klamrowym. W przeciwnym przypadku PHP je pominie.

W opisywanym konkretnym przypadku oznacza to, że aby warunek został spełniony, wartość zmiennej `$imiedarczyncy` (a zatem dane wysłane do serwera z elementu formularza o nazwie `imiedarczyncy`) musiałaby być równa pustemu ciągowi znaków, który jest reprezentowany za pomocą dwóch apostrofów (`''`) lub dwóch znaków (`""`) cudzysłowu. Należy zwrócić uwagę, że w PHP równość dwóch wyrażeń jest wyrażana za pomocą podwójnego znaku równości (`==`), w przeciwieństwie do operacji przypisywania wartości, która jest reprezentowana przez pojedynczy znak równości (`=`). Przykład polecenia przypisującego wartość pokazano w powyższym fragmencie kodu, jest to polecenie umieszczone w nawiasie klamrowym (`{}`).

Warunki i operatory porównania

Pierwszy przypadek zastosowania warunku opisano podczas omawiania pętli `for` w poprzednim podrozdziale. Tam także znalazł się warunek, którego spełnienie decydowało o ponownym wykonaniu pętli. W PHP warunki mogą występować w dowolnych miejscach, nie tylko w instrukcjach `if` i `for`.

Należy zatem zapoznać się ze sposobem formułowania warunków. Warunkiem nazywa się takie wyrażenie, którego wartość jest oceniana przez PHP albo jako `true` (*prawda*), albo jako `false` (*fałsz*). Poza tym jako `false` PHP interpretuje puste ciągi znaków, wartości numeryczne `0`, jak również puste tablice. Wszystkie inne liczby, ciągi znaków i tablice są oceniane jako `true`.

Ponieważ zarówno wartości `true`, jak i `false` mogą być przechowywane w zmiennych PHP, warunek może składać się tylko z samej zmiennej. W kolejnym wierszu zawsze zostanie wyświetlony komunikat „Witam”:

```
$x = true; if ($x) { echo "Witam!"; }
```

Wartości te mogą zostać również zwrócone przez funkcję, tak więc cała funkcja może także reprezentować warunek. Przykładem może być np. funkcja `preg_match()`, którą Czytelnik pozna w dalszej części tego rozdziału.

Bardzo często spotykany jest trzeci rodzaj warunków: porównania. W prezentowanym przypadku (np. podczas kontroli imienia ofiarodawcy), porównuje się pewne wyrażenie (przesłane imię i nazwisko ofiarodawcy) z innym wyrażeniem (pusty ciąg znaków `""`). W tym celu użyto operatora `==`. Istnieją jednak jeszcze inne operatory, które można zastosować do tego typu porównań. Pierwszy zaprezentowano już podczas omawiania kodu formularza: był to operator „mniejszy niż” (`<`).

Poniżej znajduje się zestawienie kilku innych operatorów porównań.

Operator	Nazwa	Znaczenie (z przykładowymi zmiennymi)
`!=`	nierówny	`$a != $b` jest `true`, wtedy gdy wartość zmiennej `$a` nie jest równa wartości zmiennej `$b`.
`<`	mniejszy niż	`$a < $b` jest `true`, wtedy gdy wartość zmiennej `$a` jest mniejsza niż wartość zmiennej `$b`.
`>`	większy niż	`$a > $b` jest `true`, wtedy gdy wartość zmiennej `$a` jest większa niż wartość zmiennej `$b`.
`<=`	mniejszy lub równy	`$a <= $b` jest `true`, wtedy gdy wartość zmiennej `$a` jest mniejsza lub równa wartości zmiennej `$b`.
`>=`	większy lub równy	`$a >= $b` jest `true`, wtedy gdy wartość zmiennej `$a` jest większa lub równa wartości zmiennej `$b`.

Powyższe operatory można stosować zarówno w odniesieniu do wartości numerycznych, jak i ciągów znaków. Wydawałoby się, że w przypadku operatora „nierówny" można to jeszcze wyobrazić, jednak w przypadku pozostałych operatorów sytuacja wydaje się być niejasna. Tymczasem ciągi znaków są zamieniane na liczby, które składają się z kodów numerycznych odpowiadających danym znakom. Praktyczne zastosowanie tego zagadnienia wygląda jednak znacznie prościej: za pomocą operatorów można sortować ciągi znaków w porządku alfabetycznym. Ciąg znaków `$a` jest zamieniany na mniejszą wartość niż ciąg `$b`, zatem `$a` w alfabetycznym porządku pojawia się przed `$b`. Ważna jest tutaj m.in. wielkość znaków. Poniższe wyrażenia są w PHP prawdziwe: `"duży" < "mały"`, `"Duży" < "duży"`, `"Noga" < "Nóżka"`.

Kontrola liczb całkowitych

Aby sprawdzić kwotę darowizny, trzeba się cofnąć nieco wstecz. Także tutaj stosuje się instrukcję `if`, jednak należy się upewnić, że w przypadku danych wysłanych jako ciąg znaków rzeczywiście przesłano wartość równą liczbie całkowitej, która jest podzielna przez 5. Ponadto trzeba się upewnić, że liczba ta znajduje się w przedziale 5 – 100. Aby sprawdzić, że daną liczbą jest liczba całkowita podzielna przez 5, można zastosować tzw. *wyrażenie regularne*.

Listing 4.7. Kontrola kwoty darowizny za pomocą wyrażenia regularnego w pliku datek.php

```
if (!preg_match("/^\d*[05]$/", $kwota)) { // nie jest liczbą całkowitą?
                $hack = true; $pole = "Kwota";
        }
```

Warto przyjrzeć się nieco bliżej tej zaszyfrowanej konstrukcji. Warunek instrukcji `if` brzmi następująco: `!preg_match("/^\d*[05]$/", $kwota)`. Wykrzyknik znajdujący się na początku pełni funkcję znaku negacji. Oznacza to, że całe wyrażenie będzie miało wartość `true`, jeśli wyrażenie znajdujące się po znaku wykrzyknika będzie miało wartość `false` i odwrotnie. `preg_match()` jest funkcją wbudowaną w PHP, umożliwiającą porównywanie wzorców za pomocą wyrażeń regularnych kompatybilnych z Perl. Ten tajemniczo wyglądający ciąg znaków, stanowiący pierwszy argument funkcji `preg_match()`, jest takim właśnie wyrażeniem regularnym.

Wyrażenia regularne są stosowane do identyfikacji wzorców ciągów znaków w innych ciągach znaków. W opisywanym przypadku trzeba rozpoznać, czy ciąg znaków przekazany do zmiennej `$kwota` składa się wyłącznie z cyfr. Jak już wspomniano, wyrażenia regularne w funkcjach typu `preg_` mają tę samą składnię jak te w języku Perl (język ten często jest stosowany w skryptach CGI).

Omówienie kompletnej składni wyrażeń regularnych w Perlu znacznie wykracza poza ramy niniejszej książki. Dobrze napisany i przystępny podręcznik omawiający wyrażenia regularne w Perlu można znaleźć w ofercie wydawnictwa O'Reilly *Perl. Wprowadzenie* autorstwa Randala L. Schwartz i Toma Phoenix (w Polsce wydana przez wydawnictwo Helion). Wiele tego typu podręczników dostępnych jest również w internecie. Programista

aplikacji internetowych musi być przyzwyczajony do stosowania wyrażeń regularnych i powinien dobrze znać ich składnię, ponieważ w ten sposób można stosunkowo prosto wykonywać kompleksowe operacje na ciągach znaków. Dla celów niniejszej książki wystarczy, że Czytelnik zrozumie wyrażenia z prezentowanego skryptu.

Wszystkie wyrażenia regularne w Perlu (a co za tym idzie, również w funkcjach typu `preg_...`) rozpoczynają się i kończą identycznymi znakami granicznymi (są tzw. ograniczniki, ang. *delimiter*). W większości przypadków programiści Perla i PHP stosują ukośnik lewy (`/`) lub znak `#`. W omawianym przypadku trzeba się dowiedzieć, czy ciąg znaków składa się wyłącznie z cyfr. Zatem cały ciąg, od samego początku, musi odpowiadać podanemu wzorcowi, co wyraża się znakiem `^` umieszczonym na jego początku. W wyrażeniach regularnych cyfry są reprezentowane przez `/d`. Gwiazdka (`*`) oznacza, że poprzedzający ją znak może wystąpić jeden lub więcej razy. W opisywanym przypadku jest to wiele cyfr lub też żadna. Następnie trzeba żądać znaku, który będzie równoważny jednemu ze znaków umieszczonych w nawiasie kwadratowym, tzn. `0` lub `5`. Znak ten ma znajdować się równocześnie na końcu ciągu znaków. Aby to wymusić, wyrażenie regularne należy zakończyć znakiem dolara (`$`). Oznacza on, że w tym miejscu wzorca powinien znajdować się koniec ciągu znaków.

W ten sposób można się upewnić, że przesłana zmienna ma wartość ciągu znaków, składającego się wyłącznie z cyfr oraz że jego wartość numeryczna podzielna jest przez 5. W jaki jednak sposób można zagwarantować, że wysłana kwota nie będzie wynosić podejrzane 100 000 zł lub 0 zł? Sprawę tę można ponownie rozwiązać za pomocą instrukcji `if`:

```
if (($kwota < 5) || ($kwota > 100)) {
                $hack = true; $pole = "Kwota";
        }
```

Tutaj warunek instrukcji został rozbity na dwa warunki. Jeśli zostanie spełniony chociaż jeden z nich, całe wyrażenie otrzymuje wartość `true`. Efekt ten osiąga się korzystając z operatora logicznego `LUB`, który w PHP jest oznaczany dwoma pionowymi kreskami (`||`). Jeśli zmienna `$hack` po wykonaniu tej instrukcji nadal posiada wartość `false`, oznacza to, że ciąg znaków w `$kwota` odpowiada jednej z opcji w prezentowanej liście wyboru `<select>`.

Kontrola opcji wielokrotnego wyboru

Czytelnik już poznał chyba całkiem dobrze instrukcję `if`. Rodzaj karty kredytowej można by sprawdzić zagnieżdżając kilka instrukcji `if`:

```
if ($kk_rodzaj != "Visa") {
 if ($kk_rodzaj != "Mastercard") {
   if ($kk_rodzaj != "American Express") {
    $hack = true; $pole="Rodzaj karty";
   }
 }
}
```

Mając do sprawdzenia jedynie trzy opcje można jeszcze skorzystać z tej metody. Jednak jeśli trzeba sprawdzić większą ich liczbę, zagnieżdżone instrukcje `if` stają się mało przejrzyste — szczególnie wtedy, gdy dla wielu pojedynczych opcji ma zostać wykonana

konkretna sekwencja poleceń. W opisywanym przypadku mogłaby to być np. kontrola numeru karty w powiązaniu z jej rodzajem.

Dlatego w celu sprawdzenia tego rodzaju danych programiści preferują instrukcję `switch-case`:

Listing 4.8. Kontrola stałych opcji za pomocą konstrukcji switch-case w pliku datek.php

```
switch ($kk_rodzaj) {
       case "Visa": break;
       case "Mastercard": break;
       case "American Express": break;
       default:
         $hack = true; $pole = "Rodzaj karty kredytowej";
        }
```

W przypadku instrukcji `switch` istnieje jedno wyrażenie (w tym przypadku jest to `$kk_rodzaj`), które jest sprawdzane pod kątem różnych wariantów. Każdy z nich oznaczany jest jako `case`. Wartość, którą przyjmuje wyrażenie, jest podawana — dla każdego z wariantów — po słowie kluczowym `case`. Po znaku dwukropka umieszcza się polecenia PHP, które powinny zostać wykonane w danym przypadku. W prezentowanym fragmencie kodu są to jedynie polecenia `break`, powodujące opuszczenie instrukcji `switch`. Jeśli zachodzi możliwość, że nie zajdzie żaden z podanych w `case` wariantów, można określić także wariant domyślny (`default`). W prezentowanym przypadku tylko on wymaga krótkiego acz dosadnego komentarza: ktoś przesłał podejrzaną wartość, omijając formularz!

Kontrola numerów karty kredytowej i terminu ważności karty

Numery kart kredytowych można obsłużyć wykorzystując ponownie wyrażenia regularne. Spoglądając na numer karty kredytowej można stwierdzić, że składa się on z szesnastu cyfr, ułożonych w czterocyfrowych grupach. Pomiędzy grupami może znajdować się łącznik, spacja lub też grupy mogą być nierozdzielone. Ostatni przypadek zachodzi wówczas, gdy w podanym numerze karty każda z czterocyfrowych grup znajduje się bezpośrednio za poprzedzającą lub gdy jest to ostatnia z czterocyfrowych grup, kończąca numer karty. Przykładowe numery kart kredytowych w tym formacie to np. 7895-7777-3365-2178, 3033 2896 7847 2391 lub 7645889372694218. Potrzebne wyrażenie regularne może zatem wyglądać następująco:

Listing 4.9. Kontrola numeru karty kredytowej w pliku datek.php

```
if (!preg_match("/^(\d{4}[\s\-]?){4}$/", $kk_numer)) {
                $hack = true; $pole = "Numer karty";
       }
```

Ograniczniki, znak początku i końca `^` oraz `$` zostały już opisane: informują one, że znajdujący się pomiędzy nimi wzorzec dotyczy całego poddawanego kontroli ciągu znaków. Natomiast poniższa składnia wzorca wymaga komentarza: `(\d{4}[\s\-]?){4}`. W nawiasie okrągłym znajduje się wzorzec podwyrażenia: `(\d{4}[\s\-]?)`. Po nim, w nawiasie

klamrowym występuje „mnożnik": `{4}`. Mnożnik oznacza, że poprzedzający go wzorzec (a zatem podwzorzec zawarty w nawiasie okrągłym) ma wystąpić czterokrotnie w ciągu znaków `$kk_numer`.

Wspomniany podwzorzec dotyczy całej grupy cyfr lub jednej z czterocyfrowych grup z ewentualnym znakiem rozdzielającym. `\d` oznacza tutaj ponownie dowolną cyfrę. Liczba `4` znajdująca się w nawiasie klamrowym po `\d` — co oczywiste — to mnożnik, który informuje, że poprzedzający wzorzec (zatem `\d`) musi pasować czterokrotnie. Innymi słowy: `\d{4}` oznacza ciąg czterech dowolnych cyfr.

Następnie w nawiasie kwadratowym znajduje się „klasa": `[\s\-]`. Klasa podaje pewną liczbę znaków lub wzorców, które w tym miejscu obowiązkowo muszą wystąpić. W opisywanym przypadku jest to albo spacja[2], albo łącznik, który tutaj musi zostać poprzedzony lewym ukośnikiem. W wyrażeniach regularnych łączniki mają inne znaczenie. Znak zapytania, który występuje po nawiasie kwadratowym, informuje, że znak z danej klasy musi wystąpić jeden raz lub wcale.

Zrób to sam!

Powyższe wyrażenie regularne, służące do kontroli numeru karty kredytowej ma pewien drobny mankament: akceptowane są mianowicie numery kart, których ostatnim znakiem jest łącznik lub spacja. Można wpisać np. numer `1111222233334444-`. Oczywiście nie jest to prawidłowy numer karty kredytowej, ale opisywane wyrażenie kontrolne nie jest w stanie tego wychwycić. Warto zatem uzupełnić to wyrażenie regularne, tak aby numer karty kredytowej musiał się definitywnie kończyć cyfrą. Rozwiązanie problemu znajduje się w komentarzu umieszczonym w pliku *datek.php*, dostępnym on-line.

Mówiąc o wyrażeniach regularnych można zauważyć, że w kodzie pliku *datek.php* znajduje się jeszcze jedno wyrażenie, służące co kontroli terminu ważności karty kredytowej:

Listing 4.10. Kontrola terminu ważności karty kredytowej w pliku datek.php

```
if (!preg_match("/^(\d{2})0[2-9]$/", $kk_data, $match)) {
    $hack = true; $pole = "Termin ważności karty";
}
else
{
  if (($match[1] < 1) || ($match[1] > 12)) {
       $hack = true; $pole = " Termin ważności karty";
  }
}
```

Pożądaną wartością jest ciąg znaków, którego pierwsze dwie cyfry oznaczają miesiąc i znajdują się w zakresie między 01 a 12. Trzecia i czwarta cyfra oznacza rok. W przypadku pierwszych dwóch cyfr (oznaczających miesiąc) muszą one odpowiadać podwyrażeniu

[2] Znaki puste odpowiadające `\s` to także znak tabulatora i końca wiersza.

`(\d{2})`. Nawias okrągły pełnią w tym przypadku jeszcze jedną funkcję dodatkową: częściowy ciąg znaków, odpowiadający temu podwyrażeniu, zostanie zapisany w tablicy `$match`, która jest trzecim argumentem funkcji `preg_match()`. Do tej kwestii powrócimy za chwilę. Obecnie nie ma w obiegu kart, których termin ważności upływa po roku 2009, zatem można założyć, że trzecią cyfrą musi być `0`. Czwarta cyfra musi znajdować się w przedziale 2 a 9 (można również wpisać rok wydania tej książki). Łącznik wyznacza tutaj zakres.

Poza tym zastosowano klauzulę `else`, będącą rozszerzeniem instrukcji `if`. Klauzula `else` pojawia się zaraz po instrukcji `if`. Polecenia znajdujące się w nawiasie klamrowym umieszczonym po słowie kluczowym `else` są wykonywane tylko wtedy, gdy warunek instrukcji `if` jest fałszywy. W opisywanym przykładzie jest to przypadek, gdy `$kk_data` składa się z prawidłowych 4 cyfr. Następnie trzeba upewnić się, że pierwsze dwie cyfry należą do przedziału od 01 do 12.

Kontrola pól wyboru

Kontrola pól wyboru jest stosunkowo prosta: przesłane dane muszą się składać z pustego ciągu znaków lub z ciągu `"on"` , w przeciwnym razie należy przypuszczać, że dane pochodzą z podejrzanego źródła.

Listing 4.11. Kontrola pól wyboru w pliku datek.php

```
if (($publiczny != "") && ($publiczny!= "on")) {
     $hack = true; $pole = "Publikacja danych ofiarodawcy";
}
```

Chcąc wyrazić, że coś „nie jest równe" stosuje się operator nierówności `!=`. Aby wartość zmiennej `$publiczny` została uznana za podejrzaną, muszą zostać spełnione oba warunki jednocześnie. W tym celu połączono je operatorem `I` (`&&`).

Kontrola czasu generowania formularza

Kontroli czasu generowania formularza dokonuje się również za pomocą wyrażenia regularnego:

Listing 4.12. Kontrola czasu generowania formularza w pliku datek.php

```
if (!preg_match("/^\d+$/", $godzina)) {
    $hack = true; $pole = " Czas wypełnienia formularza ";
}
```

W ten sposób zakończono sprawdzanie zwykłych pól służących do wprowadzania danych. Teraz trzeba tylko zapewnić przetworzenie opcjonalnego pliku graficznego.

Przetwarzanie pliku przesłanego do serwera

Teraz trzeba zwiększyć czujność.

- Po pierwsze, pliku może wcale nie być (i nie będzie to przecież błąd!). Można to rozpoznać po wielkości pliku równej 0.
- Po drugie, plik musi zostać zapisany we wcześniej zdefiniowanym miejscu. Aby zapobiec nadpisaniu znajdującemu się w danej lokalizacji pliku o tej samej nazwie, trzeba zapewnić unikalną nazwę dla kopiowanego pliku. Poza tym należy rozpoznać format pliku.

Listing 4.13. Sprawdzenie i zapisanie zdjęcia ofiarodawcy w pliku datek.php

```
if ($_FILES["zdjeciedarczyncy"]["size"] > 0) {
    $zdjecie = true;
    preg_match("/(\.\w+)$/",
         $_FILES["zdjeciedarczyncy"]["name"],$match);
    $typ = $match[1];
    // akceptujemy tylko rozszerzenia plików graficznych
    if (in_array(
          strtolower($typ),
          array(".gif",".bmp",".jpg",".png",".jpeg"))) {
        $nazwapliku = uniqid("").$typ;
        $sciezka =
            preg_replace("/\/[^\/]+$/","",
                         $_SERVER["SCRIPT_FILENAME"])
                         ."/Obrazki/";
         copy($_FILES["zdjeciedarczyncy"]["tmp_name"],
              $sciezka.$nazwapliku);
    }
}
else
{
    $zdjecie = false;
}
```

Tablica asocjacyjna `$_FILES`[3] jest tablicą dwuwymiarową. Oznacza to, że wyrażenie `$_FILES["zdjeciedarczyncy"]` także stanowi tablicę asocjacyjną. Element tej tablicy, dostępny pod wartością indeksową `"size"`, zawiera wielkość przesłanego pliku w bajtach. Jeśli wielkość ta wynosi 0, oznacza to, że nie przesłano żadnego pliku. W tym przypadku należy ustawić zmienną `$zdjecie` na boolowską wartość logiczną `false`. Jeśli natomiast plik jest dostępny, wartość ta zostanie ustawiona na `true`. Będzie to pomocne przy generowaniu strony WWW potwierdzającej złożenie datku.

Określanie rozszerzenia pliku

Po uzyskaniu pewności co do faktu przesłania pliku trzeba odszukać rozszerzenie pliku oryginalnego, który został wysłany z przeglądarki klienta. Informacja o pliku graficznym jest potrzebna, gdyż przesłany plik graficzny będzie wyświetlany w przeglądarkach. Aby przeglądarka mogła we właściwy sposób wyświetlić obraz z pliku graficznego, musi wcześniej otrzymać informację o jego formacie. Określenie rozszerzenia pliku jest możliwe przy zastosowaniu `$_FILES["zdjeciedarczyncy"]["type"]` i instrukcji `switch`,

[3] Także `$_FILES` pojawia się w PHP 4.1.0 — we wcześniejszych wersjach tablica ta nosi nazwę `$HTTP_POST_FILES`.

która reaguje w zależności od typu pliku. Proponowany sposób wykorzystania wyrażenia regularnego jest jednak nieco bardziej elegancki, tym bardziej, że informacja o rodzaju pliku jest dla przeglądarki bardzo ważna.

Zastosowane wyrażenie regularne poszukuje znaku kropki na końcu oryginalnej nazwy pliku, po którym występuje jeden lub wiele znaków czyli słów, którymi mogą być litery, cyfry i znak podkreślenia (_). Rozszerzenia plików zawierają się w tej grupie. Kropka została oznaczona przez poprzedzający ją lewy ukośnik. Znaki-słowa w wyrażeniach regularnych są oznaczane ciągiem znaków `\w`. Znak `+` oznacza, że poprzedzający go ciąg znaków lub znak musi wystąpić przynajmniej raz. Rozszerzenie nazwy pliku musi zatem składać się z przynajmniej jednej litery. Częściowy ciąg znaków, oznaczający rozszerzenie, zostaje zapisany w tablicy `$match` a następnie przekopiowany do wygodniejszej zmiennej `$typ`.

Skrypt wgrywający plik na serwer — uwaga, pułapka!

Teraz trzeba pamiętać o bezpieczeństwie. Zapisywanie na serwerze pliku, który został wysłany przez użytkownika sieci WWW wymaga rozwagi. Nazwę, względnie adres URL zapisanego pliku trzeba udostępnić przeglądarce w znaczniku `<img>`, aby umożliwić wyświetlenie obrazu na stronie WWW. Istnieje jednak pewien problem: skąd można uzyskać pewność, że użytkownik przesłał plik graficzny?

W zasadzie użytkownik mógłby przesłać skrypt PHP, który zostałby wywołany przez adres URL podany w znaczniku `<img>`. Za pomocą takiego skryptu i zawartych w nim różnych funkcji (np. omówionych w rozdziale 8. funkcji zewnętrznych) można spowodować wiele problemów. Przykładowo, w ten sposób haker mógłby wykonywać na danym serwerze dowolne polecenia — wykorzystując do tego celu jakąkolwiek przeglądarkę.

Sposób traktowania pliku znajdującego się na twardym dysku serwera zależy m.in. od rozszerzenia pliku. Jeśli rozszerzenie wskazuje na plik graficzny, serwer nie będzie go interpretował jako skryptu. Zatem trzeba się upewnić, że zapisanie danego pliku na dysk nastąpi tylko w przypadku, gdy `$typ` jest znanym rozszerzeniem pliku graficznego.

Można tego dokonać za pomocą instrukcji `if`. Jej warunek wykorzystuje wbudowaną funkcję PHP `in_array()`, która ma na celu sprawdzenie, czy rozszerzenie zawarte w `$typ` znajduje się w tablicy z akceptowanymi rozszerzeniami. Aby oszczędzić sobie pracy związanej z wypisaniem wszystkich możliwych kombinacji wielkich i małych liter, można dokonać konwersji zapisu zmiennej `$typ` na zapis uwzględniający tylko małe litery. Ułatwi to wyszukiwanie w ciągu znaków.

Jeśli rozszerzenie pozwala na zaakceptowanie pliku, zostanie ono użyte w kolejnym wierszu do wygenerowania nazwy pliku, pod którą przesłany plik zostanie zapisany na serwerze. Główną część nazwy pliku stworzy PHP. Funkcja `uniqid()`, wykorzystując czas systemowy, generuje unikalny ciąg znaków, który składa się z cyfr i małych liter oraz ma długość 13 znaków, np. `3c6d829638ae5`. Do wygenerowanej nazwy należy dodać wcześniej określone rozszerzenie i w ten sposób tworzy się nową nazwę pliku, czyli np. `3c6d829638ae5.jpg`.

Uwaga!

Aby móc zapisać plik, trzeba znać absolutną ścieżkę dostępu do katalogu docelowego. Ważnym jest absolutnie jednoznaczne określenie miejsca, do którego dany plik zostanie skopiowany poleceniem `copy`.

Z uwagi na fakt, że plik ten będzie wysyłany ponownie do przeglądarki, warto zapisać go w podkatalogu znajdującym się w katalogu ze skryptem, np. w *obrazki/*. Katalog ten musi być zapisywalny z poziomu serwera, tzn. najpierw trzeba określić odpowiednie prawa dostępu.

Dla systemów z rodziny MS Windows można zdefiniować ścieżkę w pliku *datek.php* w następujący sposób:

```
$sciezka="C:/Documents and Settings/Administrator/Moje dokumenty/ora/
phppw/Rodzial4/WatekGlowny/obrazki/";
```

System Windows, interpretując ścieżkę do danego katalogu, zaakceptuje także prawe ukośniki zamiast zwykłych lewych ukośników. Jeśli trzeba będzie przesunąć skrypt do innego katalogu lub umieścić do na serwerze sieci WWW, tego rodzaju ścieżka nie umożliwi prawidłowego działania skryptu.

Lepszym rozwiązaniem jest zażądanie od PHP określenia ścieżki absolutnej do danego skryptu. Jest ona dostępna (łącznie z nazwą skryptu) pod indeksem `SCRIPT_FILENAME` w globalnej tablicy asocjacyjnej `$_SERVER`. Tablica ta jest udostępniana skryptowi przez PHP i zawiera dane dotyczące otoczenia zapytania HTTP, dzięki któremu wywołano skrypt. Należą tutaj obok ścieżki i adresu URL m.in. także adres IP klienta.

Teraz trzeba usunąć ze ścieżki nazwę skryptu. W ten sposób można otrzymać ścieżkę do katalogu ze skryptem. Dokonuje się tego za pomocą funkcji `preg_replace()`:

```
$sciezka =
  preg_replace("/\/[^\/]+$/","",
    $_SERVER["SCRIPT_FILENAME"])."/obrazki/";
```

Pierwszym parametrem tej funkcji jest wyrażenie regularne, ustalające wzorzec ciągu znaków, który należy odnaleźć. Czytelnik już poznał podobne działanie funkcji `preg_match()`. Drugim parametrem jest ciąg znaków, który ma zastąpić znaleziony wzorzec. Trzecim parametrem jest ciąg znaków, w którym ma dojść do wymiany.

W opisywanym przypadku w ścieżce należy odszukać znaki prawego ukośnika (`\/`), po którym może występować dowolna liczba znaków, które *nie* mogą być prawymi ukośnikami. Wyraża to się za pomocą `[^\/]` — znak `^` oznacza tutaj, że występujące po nim znaki w klasie *nie* mogą wystąpić. Wzorzec ten musi sięgać do samego końca klasy. W ten sposób można objąć nazwę pliku ze skryptem razem ze stojącym przed nim ukośnikiem pochodzącym ze ścieżki, zatem */datek.php*. Jest on zastępowany pustym ciągiem znaków, czyli w rzeczywistości jest usuwany.

Do pozostałej części ścieżki można teraz dołączyć katalog *obrazki/*. Trzeba jednak dołączyć ten usunięty wcześniej ukośnik. Dokonuje się tego za pomocą kropki, którą już wcześniej zastosowano do łączenia dwóch ciągów znaków.

W kolejnym wierszu następuje kopiowanie pliku z jego tymczasowego miejsca przechowania do katalogu docelowego. W tym momencie ważnym jest, że PHP samodzielnie określa nazwę pliku oraz akceptuje tylko wybrane rozszerzenia. Aby zrozumieć przyczyny tego stanu rzeczy, warto przeanalizować poniższy przykład. Można założyć, że odrzucono by rozszerzenie plików skryptowych *.php*, nie wygenerowano by własnej nazwy pliku i zamiast tego przyjęto by nazwę pliku przesłaną przez przeglądarkę.

W ten sposób potencjalny haker mógłby wpisać ciąg znaków `"../index.html"` jako nazwę pliku. Plik ten zostałby zapisany w systemie Windows np. jako:

```
C:/Documents and Settings/Administrator/Moje dokumenty/ora/
phpbuch/Rodzial4/WatekGlowny/obrazki/../index.html
```

czyli:

```
C:/Documents and Settings/Administrator/Moje dokumenty/ora/
phpbuch/Rodzial4/WatekGlowny/index.html
```

W ten sposób powstałby problem — jest to bowiem nazwa pliku startowego. Zatem oryginalny plik startowy mógłby zostać nadpisany plikiem przesłanym do serwera. Nawet wykorzystując wyrażenia regularne nie można zapewnić pełnego bezpieczeństwa. Przykładowo, zastosowanym wyrażeniem regularnym nie byłoby `"/(\..+)$/"` lecz `"/(\.\w+)$/"`. Dzięki znakowi kropki, znajdującemu się przed znakiem plusa szukano by dowolnych znaków, ale niekoniecznie znaków-słów. Dlatego zawsze istnieje możliwość otrzymania nowego pliku startowego o nazwie */../../index.html*, co również spowodowałoby efekt opisany powyżej. Czujność jest zatem wskazana.

Żądając celowo znaków-słów następujących po znaku kropki można uniknąć tego problemu — nazwy plików, jak ta powyższa, zostają w ten sposób zredukowane do nieszkodliwego *.html*.

Następnie można by przeprowadzić kontrolę wielkości pliku. Kto chciałby przesłać zdjęcia w pliku o wielkości 10 MB lub więcej? Zdjęcia przeznaczone do publikacji w internecie nie powinny przekraczać kilkunastu kB.

W obecnej wersji opisywanego skryptu nie trzeba sprawdzać prostych danych wejściowych, gdyż wadliwe dane doprowadzą co najwyżej do wyświetlenia błędnej strony w przeglądarce. To się zmieni po rozszerzeniu skryptu o funkcję zapisu danych w bazie. Z tego powodu dobrą praktyką jest, aby zawsze dokładnie sprawdzać dane wejściowe pod kątem ich zawartości.

Zachowanie skryptu w przypadku nieprawidłowych danych wejściowych

W ten sposób można zakończyć sprawdzanie danych wprowadzonych przez użytkownika i wiadomo już, czy są one prawidłowe czy też nie. Teraz trzeba zapewnić, aby w razie nieprawidłowości wyświetlić komunikat o błędzie i zakończyć wykonywanie skryptu:

Listing 4.14. Wyświetlanie komunikatu o błędzie w przypadku nieprawidłowych danych w pliku datek.php

```
if ($hack) { ?>
    <html>
      <body>
       <h1>Błędne dane</h1>
       Wpis w polu <b><?php echo $pole; ?></b>
       jest niepoprawny.
      </body>
    </html>

<?php
    exit(); // zakończ skrypt!
}
```

Decyzję podejmuje się na podstawie wartości zmiennej `$hack` i za pomocą instrukcji `if`. Opisywana instrukcja `if` wyróżnia się tym, że zawiera nie tylko polecenia PHP, ale także w dużej mierze kod HTML, który jest wyświetlany tylko wtedy, gdy zmienna `$hack` ma wartość `true`. W tym celu należy opuścić PHP w obrębie nawiasu klamrowego instrukcji `if`, wykorzystując zamykający znacznik `?>` i ponownie powrócić do PHP, aby wyświetlić wartość zmiennej `$pole`. Dopiero na krótko przed końcem instrukcji `if` następuje powrót do PHP za pomocą znacznika `<?php`. Zadaniem `exit()` jest zakończenie skryptu po wyświetleniu kodu HTML, aby nieprawidłowe dane nie spowodowały dalszych szkód.

Zachowanie skryptu po otrzymaniu prawidłowych danych wejściowych

Jeśli skrypt działa nadal, to dane pochodzące z formularza powinny być prawidłowe. Teraz można by zapisać otrzymane dane w bazie. Ale jak już wspomniano, to zagadnienie zostanie omówione w jednym z dalszych rozdziałów. Teraz należy skoncentrować się na wygenerowaniu strony z potwierdzeniem otrzymania datku:

Listing 4.15. Generowanie strony potwierdzenia w pliku datek.php

```
?>
  <html>
    <body>
      <h1>Drogi(a) <?php echo $imiedarczyncy; ?></h1>
      Serdeczne podziękowania za Twój datek w kwocie  <?php echo $kwota ?> zł.
      Potwierdzenie złożenia darowizny zostanie wysłane na Twój adres <p>
      <b><?php echo preg_replace("/\r?\n/","<br>",$adres); ?></b><p>
      .<p>
      Kwota datku zostanie potrącona z Twojej karty
      <b><?php echo $kk_rodzaj; ?></b> numer:<p>
      <b><?php echo $kk_numer; ?></b> z terminem ważności do
      <b><?php echo $kk_data; ?></b>.<p>
               <?php
               if ($zdjecie) { ?>
                   Twoje zdjęcie zobaczysz poniżej:<br>
                   <img src="<?php echo "Obrazki/".$nazwapliku; ?>"><p>
               <?php
          } ?>
      Przyjmujemy do wiadomości, że Twoich danych
      <?php
        if ($publiczny == "") {
            echo "nie";
```

```
            }
          ?>
        możemy opublikować...<p>

        Na wypełnienie formularza potrzebowałeś <?php echo (time() - $godzina); ?>
        sekund.
      </body>
    </html>
```

Znacznik `?>` na początku tego fragmentu skryptu pozwala na opuszczenie PHP, ponieważ potwierdzenie będzie się składało w przeważającej części z kodu HTML. W odpowiednich miejscach wstawiono właściwe wartości wykorzystując małe fragmenty kodu PHP. Właściwie wszystkie zastosowane techniki powinny być znajome Czytelnikowi. Interesującym miejscem jest atrybut `src` w znaczniku `<img>`, za pomocą którego ustalono adres URL zdjęcia. Tutaj trzeba podać ścieżkę dostępu do pliku graficznego, gdyż plik ze zdjęciem znajduje się teraz w podkatalogu katalogu, gdzie umieszczono opisywany skrypt PHP. Wystarczy zatem umieścić ciąg znaków `"Obrazki/"`, który wraz z nazwą plik graficznego zostanie zinterpretowany jako relatywny adres URL.

> *Zrób to sam!*
>
> Teraz można powrócić na moment do omówionego wcześniej skryptu *data.php*. Jak już wspomniano, ma on małą wadę: wartości oznaczające sekundy i minuty są zwracane przez funkcję w tablicy asocjacyjnej jako liczby. Oznacza to, że w naszym obecnym skrypcie godzina może mieć postać np. 12:5:5, jeśli w danym momencie jest pięć minut i pięć sekund po dwunastej. Dzięki odpowiedniemu wyrażeniu regularnemu można tę sytuację rozpoznać i poprawić. Rozwiązanie tego problemu znajduje się w pliku *data_formatowana.php* w przykładach do tego rozdziału.

Cudzysłów — mały mankament czy duży problem?

Przed rozpoczęciem tworzenia kolejnych skryptów należy przyjrzeć się drobnej, trudno dostrzegalnej usterce. Niekiedy sposób skonfigurowania PHP może uniemożliwić zauważenie tej wady. Ma to jednak swoje konsekwencje, które mogą Czytelnikowi przeszkadzać podczas wykonywania ćwiczeń zaprezentowanych w późniejszych rozdziałach.

Przykładowo, jako imię darczyńcy może zostać podany ciąg znaków zawierający apostrof, np. **O'Reilly**. Jeśli po wprowadzeniu takiego imienia zostanie ono wyświetlone z lewym ukośnikiem jako *O\'Reilly*, oznacza to, że zaistniał problem. W pliku konfiguracyjnym PHP istnieje bowiem parametr `magic_quotes_gpc`, który z reguły jest ustawiony na `On`. Jego zadaniem jest poprzedzenie lewym ukośnikiem wszystkich apostrofów, cudzysłowów oraz prawych ukośników, znajdujących się w ciągach znaków będących elementami tablicy `$_POST`.

Aby zrozumieć powód takiego zachowania, trzeba wiedzieć, że PHP często jest stosowane w połączeniu z bazami danych SQL.

Aby móc zapisać ciąg znaków w bazie SQL, trzeba, wykorzystując odpowiednie polecenia SQL, otoczyć go znakami apostrofu lub cudzysłowu. W przeciwnym przypadku SQL nie zinterpretuje ciągu znaków jako *string*, lecz jako polecenie lub inny element składni SQL. Dla SQL ciąg znaków wygląda w następujący sposób.

```
"To jest ciąg znaków"
```

Teraz należy zastanowić się, co się stanie po umieszczeniu w nim cudzysłowu:

```
"To niestety "nie" jest ciąg znaków"
```

Otwierający znak cudzysłowu przed `"nie"` po prostu zakończy ciąg znaków. Kolejne znaki zostałyby zinterpretowane jako składnia SQL, co nie byłoby korzystne. Jeśli ciąg znaków ma zostać przesłany przez przeglądarkę, to stwarza się potencjalnemu hakerowi możliwość kontrolowania bazy danych poprzez umieszczone w nim polecenia SQL. Do tego tematu powrócimy jeszcze później.

Tego zagrożenia można uniknąć w prosty sposób: przed każdym znakiem cudzysłowu umieszcza się lewy ukośnik:

```
"To jest znowu \"ciąg\" znaków"
```

`magic_quotes_gpc` w pliku *php.ini* służy do tego, aby zautomatyzować ten proces dla danych pochodzących od przeglądarki. Jak Czytelnik pamięta z analizy przypadku z imieniem darczyńcy, może wywołać to pewne problemy. W PHP, HTML i Javascript ciągi znaków są ograniczane albo znakami apostrofu, albo cudzysłowu, zatem istnieje duże prawdopodobieństwo wystąpienia błędu. Zatem problem ten musi zostać rozwiązany.

Jeśli `magic_quotes_gpc` pozostaje włączone, trzeba umieszczać dowolne ciągi znaków między znakami cudzysłowu. W przeciwnym przypadku trzeba dodać lewy ukośnik, wykorzystując wbudowaną w PHP funkcję `addslashes()`:

```
$ciag_z_ukosnikami = addslashes ($ciag_bez_ukosnikow);
```

A niepotrzebne ukośniki (jak ma to miejsce w opisywanym przypadku z imieniem darczyńcy) można usunąć za pomocą funkcji `stripslashes()`:

```
$ciag_bez_ukosnikow = stripslashes ($ciag_z_ukosnikami);
```

Ważnym jest, aby znać szczegóły związane z konfiguracją PHP. Czytelnik, który konfiguracji nie przeprowadzał samodzielnie, może sprawdzić wartość `magic_quotes_gpc` za pomocą funkcji `get_magic_quotes_gpc()`. Zwrócenie wartości `1` oznacza, że `magic_quotes_gpc` jest ustawione na `On`.

Po uzyskaniu tej informacji należy się zastanowić, czy dany ciąg znaków może zawierać ukośniki, czy też nie. Przy włączonym parametrze `magic_quotes_gpc` opisywany skrypt mógłby wyglądać w następujący sposób.

Listing 4.16. Usuwanie ukośników za pomocą stripslashes()

```
<html>
  <body>
    <h1>Drogi(a) <?php echo stripslashes($imiedarczyncy); ?></h1>
```

```
Serdeczne podziękowania za Twój datek w kwocie <?php echo $kwota ?> zł.
Potwierdzenie złożenia darowizny zostanie wysłane na Twój adres <p>
<b><?php echo stripslashes (preg_replace("/\r?\n/","<br>",$adres));
?></b><p>.<p>
```

Na potrzeby kolejnych rozdziałów przyjęto, że `magic_quotes_gpc` pozostaje włączone. Osoby, które mają inną konfigurację, będą musiały zmienić skrypty w odpowiedni sposób.

Skrypty PHP combo i include

Czytelnik z pewnością oglądał już interaktywne strony WWW o ciekawej właściwości: w przypadku wysłania formularza z brakującym lub błędnym wpisem, dany formularz wyświetlany był ponownie, a zawartość prawidłowo wypełnionych pól nie była usuwana. Z PHP efekt ten można osiągnąć w prosty sposób. Zalecane są dwie strategie: po pierwsze nie tworzy się dwóch wersji formularza, lecz formularz wyjściowy i formularz korekty jest generowany przy użyciu tego samego kodu. Po drugie, dobrym pomysłem jest oddzielne przechowywanie kodu prezentacyjnego (HTML) i kodu przetwarzającego (sprawdzenie i przetworzenie danych wejściowych oraz kierowanie całym procesem). Ułatwia to poszukiwanie błędów i aktualizację plików.

Techniki, które zostaną w tym celu zaprezentowane, to *comboscript* i *includes*. *comboscript* opiera się na prostym pomyśle: do tej pory przeglądarka przyjmowała plik z formularzem a jego zawartość była wysyłana do innego pliku (skryptu), którego zadaniem było przetworzenie otrzymanych danych.

W skrypcie combo oba te pliki zostały połączone w jedną całość: w razie żądania pliku z poziomu przeglądarki bez wysyłania w zapytaniu danych z formularza, zostaje zwrócony czysty formularz. Jeśli zapytanie o skrypt zawiera również dane z formularza, skrypt sprawdza, czy dane są poprawne. Jeśli tak jest, przetwarza plik w odpowiedni sposób i zwraca komunikat potwierdzający. W przeciwnym przypadku następuje zwrócenie formularza z danymi wpisanymi uprzednio przez użytkownika.

Aby móc automatycznie wypełnić formularz, trzeba go nieco rozszerzyć. Rozszerzenie to zostało uwzględnione w pliku *formularz_datek_wypelnienie.php* w przykładach dostępnych on-line:

Listing 4.17. Kod pliku formularz_datek_wypelnienie.php

```
<html>
  <head><meta http-equiv="content-type" content="text/html; charset=iso8859-2">
    <title>Formularz darowizny dla zagrożonych ptaków</title>
  </head>
  <body>
    <form name="datek" action="datek_combo.php" method="post"
      enctype="multipart/form-data">
      <input type="hidden" name="godzina"
        value="<?php echo time();?>">
      <h1>Formularz darowizny</h1>
      <?php
        if ($hack) {
```

```
?>
Niestety nie możemy przetworzyć Twojego formularza, ponieważ nie został
wypełniony prawidłowo. Proszę wypełnić prawidłowo pole <b><?php echo
$pole; ?></b> i proszę sprawdzić także inne wpisy, łącznie z plikiem
zdjęcia, jeśli chcesz je załączyć.
<?php
  }
  else
  {
?>
    Cieszymy się, że chcesz złożyć datek! Prosimy o podanie swojego
    imienia, adresu, wysokości darowizny i dane karty kredytowej.
<?php
  }
?>
Następnie kliknij 'Wyślij datek'.
<p>
<b>Imię i nazwisko:</b> <input type="text" name="imiedarczyncy" size="80"
                         value="<?php
                         echo htmlspecialchars(
                         stripslashes($imiedarczyncy)); ?>"><p>
<b>Adres:</b><br>
<textarea name="adres" rows="4" cols="40" align="top"><?php
echo stripslashes($adres); ?></textarea><p>
<b>Wysokość darowizny:</b>
<select name="kwota">
  <?php
    for($i = 5; $i < 101; $i = $i + 5) {
      echo "<option value=\"".$i."\"";
      if ($kwota == $i) {
        echo " selected";
      }
      echo ">".$i." PLN\n";
    }
  ?>
</select>
 
<b>Zdjęcie darczyńcy</b> (opcja): <input name="zdjeciedarczyncy"
   type="file">
<p>
<b>Rodzaj karty kredytowej:</b> <input type="radio" name="kk_rodzaj"
   value="Visa" <?php
  if ($kk_rodzaj == "Visa") { echo "checked"; } ?>>Visa
   <input type="radio" name="kk_rodzaj" value="Mastercard" <?php
  if ($kk_rodzaj == "Mastercard") { echo "checked"; } ?>>Mastercard
   <input type="radio" name="kk_rodzaj" value="American Express" <?php
  if ($kk_rodzaj == "American Express") { echo "checked"; } ?>>American
      Express
<p>
<b>Numer karty kredytowej:</b>
<input type="text" name="kk_numer" size="20" maxlength="20"
   value="<?php echo $kk_numer; ?>">
 
<b>Termin ważności karty kredytowej:</b>
<input type="text" name="kk_data" size="4" maxlength="4"
   value="<?php echo $kk_data; ?>">
<p>
<b>Kliknij tutaj, jeśli zgadzasz się na upublicznienie Twoich danych::</b>
<input type="checkbox" name="publiczny" <?php
  if ((!$hack) || ($publiczny == "on")) { echo "checked"; }?>>
```

```
        <p>
        <input type="submit" value="Wyślij datek!">
      </form>
    </body>
  </html>
```

Przeglądając powyższy skrypt warto zwrócić uwagę na kilka nowych elementów. Pierwszym zadaniem instrukcji `if` jest wyświetlenie nieco innego tekstu wprowadzającego, jeśli wartość zmiennej `$hack` jest `true`. Dobrze się składa, że PHP nieznanym zmiennym logicznym przypisuje wartość `false`. Dlatego przy pierwszym załadowanie formularza nie trzeba się martwić o to, czy do zmiennej `$hack` została przypisana jakaś wartość czy nie.

Wypełnianie wstępne pól formularza

Kolejną nowością jest fakt, że wszystkim polom służącym do wprowadzania tekstu została przypisana wartość domyślna. Treść atrybutu `value` ustawiono na wartość odpowiedniej zmiennej wykorzystując znacznik PHP i polecenie `echo`. W tym przypadku także za korzystny należy uznać fakt, że PHP traktuje nieznane zmienne jak puste ciągi znaków. Treść `<textarea>` zostaje wpisana — także za pomocą polecenia `echo` — po prostu pomiędzy znacznik `<textarea>` a `</textarea>`.

Dla kwoty darowizny sprawa jest nieco bardziej skomplikowana. Tutaj trzeba w pętli `for` za pomocą instrukcji `if` wybrać odpowiednią opcję, której następnie przypisuje się atrybut `selected`. Jeśli zmienna `$kwota` nie została ustawiona, przeglądarka domyślnie wybierze pierwszą opcję z listy. Jak za chwilę Czytelnik się przekona, w skrypcie *combo* można ofiarodawcy z góry zaproponować pewną kwotę, ustawiając zmienną `$kwota` na określoną wartość domyślną.

Pewnym mankamentem może być fakt, że pole służące do wyboru pliku ze zdjęciem do przesłania do serwera nie może zostać z góry wypełnione żadną wartością. Wszystkie nowsze przeglądarki zabraniają tego, gdyż byłaby to ogromna luka w zabezpieczeniach. Języki skryptowe, np. Javascript umożliwiają (a nawet jest to zalecane) wysyłanie formularzy automatycznie do serwera.

Jeśli element formularza typu `file` mógłby zostać opatrzony wartością domyślną, można by z góry określić, który z plików, znajdujących się na komputerze użytkownika, ma zostać przesłany do serwera. Użytkownikowi, który odwiedza daną stronę WWW, można by podsunąć tak wypełniony element formularza i potajemnie pobrać plik z jego komputera na serwer. Oczywiście nie jest to wskazane, dlatego trzeba będzie ponownie poprosić ofiarodawcę o wybranie pliku ze zdjęciem.

Nieco bardziej skomplikowanym alternatywnym rozwiązaniem powyższego problemu byłoby, w przypadku już dostępnego pliku, wpisanie w pole ukryte formularza nazwy, pod którą jest przechowywany tymczasowo na serwerze i użycie tej nazwy podczas ponownego przetwarzania danych z formularza. Po przeczytaniu do końca tego rozdziału, warto pomyśleć nad tym rozwiązaniem. Jakie potencjalne luki w bezpieczeństwie systemu otwierają się w ten sposób?

W przypadku przycisków opcji, określających rodzaj karty kredytowej trudno uniknąć w tak łatwy sposób przypisania każdemu przyciskowi własnej instrukcji `if`, która w razie potrzeby może nadać znacznikowi `<input>` atrybut `checked`. W przypadku formularzy z dużą liczbą przycisków opcji rozwiązanie to jest nieco czasochłonne. Pewną alternatywą byłoby dynamiczne generowanie przycisków z wartościami pochodzącym z tablicy. Mogłoby to wyglądać np. tak:

```
<b>Rodzaj karty kredytowej:</b>
<?php
$rodzaje_kart = array("Visa", "Mastercard", "American Express");
  foreach($rodzaje_kart as $kk_rodzaj) {
?>
    <input type="radio" name="kk_rodzaj" value="<?php
                echo $kk_rodzaj."\"";
                if($kk_rodzaj == $rodzajkarty){
                  echo " checked";
                }
                echo ">".$kk_rodzaj;
  }
?>
```

Na początku zatem można zdefiniować tablicę `$rodzaje_kart` i następnie za pomocą pętli `foreach` w każdej iteracji generuje się dopasowane do potrzeb przyciski opcji, wykorzystując jeden z elementów z tablicy `$rodzaje_kart`, zmienną `$kk_rodzaj`, polecenie `echo` i instrukcję `if`. Dla trzech przycisków jest to może nieco pracochłonne, ale od sześciu sztuk sposób ten jest już opłacalny. Odpowiedni plik skryptu combo to *datek_combo_dynamiczny.php*, który poza nazwą niczym się nie różni od pliku *datek_combo.php*.

Pole wyboru można zaznaczyć lub odznaczyć za pomocą atrybutu `checked`, który ustala się instrukcją `if` w zależności od wartości zmiennych `$hack` i `$publiczny`. To właściwie wszystkie główne zmiany w opisywanym formularzu do składania datków. Warto jeszcze wskazać, że atrybut `action` znacznika `<form>` teraz wskazuje na plik *datek_combo.php*.

Skrypt combo w pliku datek_combo.php

Nowy skrypt oczywiście pochodzi od pliku *datek.php*. Tak więc część kodu nie zmieniła się wcale:

Listing 4.18. Kod skryptu combo w pliku datek_combo.php

```
<?php
// Ten skrypt sprawdza, czy istnieją jakieś dane wejściowe
// wysłane z przeglądarki. Jeśli nie, wyświetlany jest formularz.
// Jeśli tak, dane pochodzące z formularza są sprawdzane
// i użytkownik otrzymuje komunikat potwierdzający. W przypadku
// nieprawidłowych danych następuje zwrócenie wypełnionej wersji formularza.

    if (sizeof($_POST) == 0) {
        $kwota = 50; // proponowana kwota datku
        include("formularz_datek_wypelnienie.php");
        exit(); // zakończ skrypt!
    }
```

```
$hack = false;  // ta zmienna logiczna sprawdza, czy nie mamy do
                   czynienia z hakerem
// Pobieranie danych z tablicy $_POST

$imiedarczyncy = $_POST["imiedarczyncy"];
$adres = $_POST["adres"];
$kwota = $_POST["kwota"];
$kk_rodzaj = $_POST["kk_rodzaj"];
$kk_numer = $_POST["kk_numer"];
$kk_data = $_POST["kk_data"];
$publiczny = $_POST["publiczny"];
$godzina = $_POST["godzina"];

// Weryfikacja danych
// $imiedarczyncy może być dowolnym ciągiem znaków
// nie może być tylko ciągiem pustym
if ($imiedarczyncy == "") {
    $hack = true; $pole = "Imię i nazwisko";
}
// $adres może być również dowolnym niepustym ciągiem znaków
if ($adres == "") {
    $hack = true; $pole = "Adres";
}
// Kwota musi być liczba całkowitą pomiędzy 5 a 100
if (!preg_match("/^\d*[05]$/", $kwota)) { // nie jest liczba całkowitą?
    $hack = true; $pole = "Kwota";
}
if (($kwota < 5) || ($kwota > 100)) {
    $hack = true; $pole = "Kwota";
}
// Akceptowane rodzaje kart kredytowych to Visa, Mastercard, American
   Express
switch ($kk_rodzaj) {
    case "Visa": break;
    case "Mastercard": break;
    case "American Express": break;
    default:
        $hack = true; $pole = "Rodzaj karty kredytowej";
}
// Numer kary powinien składać się z czterech grup,
// z których każda powinna składać się z 4 cyfr
// Grupy powinny być połączone spacją, łącznikiem lub napisane
// w jednym ciągu
if (!preg_match("/^(\d{4}[\s\-]?){4}$/", $kk_numer)) {
    $hack = true; $pole = "Numer karty";
}
// Termin ważności karty powinien składać się z 4 cyfr, przy czym
// pierwsze dwie powinny leżeć w przedziale od 1 do 12 a trzecia musi
// równać się 0 (chyba że termin ważności karty upływa po roku 2009
// Czwarta cyfra musi leżeć pomiędzy 2 a 9

if (!preg_match("/^(\d{2})0[2-9]$/", $kk_data, $match)) {
    $hack = true; $pole = "Termin ważności karty";
}
else
{
    if (($match[1] < 1) || ($match[1] > 12)) {
        $hack = true; $pole = "Termin ważności karty";
    }
}
// Pole wyboru  może być albo puste, albo zaznaczone (wartość 'on').
```

```
  if (($publiczny != "") && ($publiczny != "on")) {
      $hack = true; $pole = "Publikacja danych ofiarodawcy";
  }
  // Czas wypełnienia formularza musi być liczbą całkowitą
  if (!preg_match("/^\d+$/", $godzina)) {
      $hack = true; $pole = "Czas wypełnienia formularza ";
  }
// Czy załączono zdjęcie?
   if ($_FILES["zdjeciedarczyncy"]["size"] > 0) {
       $zdjecie = true;
       preg_match("/(\.\w+)$/",
           $_FILES["zdjeciedarczyncy"]["name"],$match);
    $typ = $match[1];
    // akceptujemy tylko rozszerzenia plików graficznych
    if (in_array(
            strtolower($typ),
            array(".gif",".bmp",".jpg",".png",".jpeg"))) {
       $nazwapliku = uniqid("").$typ;
       $sciezka =
          preg_replace("/\/[^\/]+$/","",
                       $_SERVER["SCRIPT_FILENAME"])
                        ."/Obrazki/";
        copy($_FILES["zdjeciedarczyncy"]["tmp_name"],
             $sciezka.$nazwapliku);
    }
 }
 else
 {
    $zdjecie = false;
 }

// Przy nieprawidłowych danych zwróć komunikat o błędzie:
if ($hack) {
  include("formularz_datek_wypelnienie.php");
  exit(); // zakończ skrypt!
}

// W tym miejscu dołączymy później kod wpisujący dane do bazy danych
// Wyświetlenie potwierdzenia:
?>
  <html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
    </head>
    <body>
      <h1>Drogi(a) <?php echo stripslashes($imiedarczyncy); ?></h1>
          Serdeczne podziękowania za Twój datek w kwocie
          <?php echo $kwota ?> PLN.
          Potwierdzenie złożenia darowizny zostanie wysłane na Twój adres <p>
      <b><?php echo stripslashes(preg_replace("/\r?\n/","<br>",$adres));
      ?></b><p>
      .<p>
      Kwota datku zostanie potrącona z Twojej karty
      <b><?php echo $kk_rodzaj; ?></b> numer:<p>
      <b><?php echo $kk_numer; ?></b> z terminem ważności do
      <b><?php echo $kk_data; ?></b>.<p>
      <?php
        if ($zdjecie) { ?>
            Twoje zdjęcie zobaczysz poniżej:<br>
        <img src="<?php echo "obrazki/".$nazwapliku; ?>"><p>
      <?php
      } ?>
```

```
        Przyjmujemy do wiadomości, że
        <?php
          if ($publiczny == "") {
              echo " Twoich danych nie ";
          }
           else
          {
              echo " Twoje dane ";
          }
        ?>
        możemy opublikować...<p>
        Na wypełnienie formularza potrzebowałeś <?php echo (time()
        - $godzina);  ?>
        sekund.
      </body>
    </html>
```

Największe zmiany zaszły w początkowej części skryptu. Za pomocą funkcji `sizeof()` można sprawdzić, czy `$_POST` zawiera jakieś dane. Jeśli plik *datek_combo.php* zostaje wywołany z przeglądarki bezpośrednio lub jako odnośnik, żadne dane nie są wysyłane z formularza do serwera, czyli tablica `$_POST` jest pusta. Oznacza to, że jej wielkość wynosi `0`. W tym przypadku należy wyświetlić tylko formularz, razem z domyślną kwotą datku:

```
if (sizeof($_POST) == 0) {
    $kwota = 50; // proponowana kwota datku
    include("formularz_datek_wypelnienie.php");
    exit(); // zakończ skrypt!
}
```

Sam formularz dołączany jest do skryptu za pomocą instrukcji `include`. PHP zmienia tutaj tryb na HTML, tak więc dołączony plik jest traktowany jako plik HTML, do momentu pojawienia się pierwszego znacznika PHP. Poza formularzem nie trzeba niczego wysyłać do przeglądarki. Z tego też powodu po instrukcji `include` należy zakończyć wykonywanie skryptu poleceniem `exit()`.

Kolejna zmiana w skrypcie polega na usunięciu komunikatu o błędzie i zastąpieniu go kolejną instrukcją `include`:

```
if ($hack) {
    include("formularz_datek_wypelnienie.php");
    exit(); // zakończ skrypt!
}
```

Cała reszta pozostaje niezmieniona.

Zrób to sam!

Warto spróbować przechować komunikat potwierdzenia w zewnętrznym pliku, który następnie może być włączony do skryptu za pomocą instrukcji `include`. Warto pomyśleć także nad innymi zastosowaniami polecenia `include`. Może hojniejszym ofiarodawcom warto wyświetlać nieco bardziej atrakcyjną stronę potwierdzenia? Przykładowe rozwiązanie znajdują się w pliku *datek_combo_rozszerzony.php* w przykładach do tego rozdziału, dostępnych on-line.