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. Stwórz własną stronę internetową



Autor: Praca zbiorowa
Tłumaczenie: Rafał Jońca (rozdz. 22 – 26, 28 – 33), Marek Pałczyński (rozdz. 3, 7, 10 – 13, 15), Ewa Sławińska (wprowadzenie, rozdz. 1, 2, 4 – 6, 8, 9, 14, 16 – 21, 27)
ISBN: 83-7361-665-9
Tytuł oryginału: Creating Your Web Site with PHP
Format: B5, stron: 496
Przykłady na ftp: 55 kB

Książka „PHP. Stwórz własną stronę internetową” to podręcznik przedstawiający zasady tworzenia aplikacji internetowych z wykorzystaniem języka PHP i bazy danych MySQL. Według założeń autora ma on pomóc nawet najmniej doświadczonym programistom w zdobyciu kwalifikacji niezbędnych do rozpoczęcia kariery zawodowej w świecie tworzenia aplikacji internetowych. Każde poruszone w nim zagadnienie ilustrowane jest starannie dobranymi przykładami metod programowania, które pomogą nauczyć się tworzenia wysokiej jakości aplikacji. Niemal we wszystkich rozdziałach książki prezentowane są najbardziej praktyczne i najczęściej wykorzystywane sztuczki programistyczne (mające związek nie tylko z językiem PHP).

- Podstawowe informacje o funkcjonowaniu internetu
- Zasady działania skryptów uruchamianych po stronie serwera
- Instalacja i konfiguracja serwera Apache, platformy PHP i bazy danych MySQL
- Szczegółowy opis języka PHP
- Korzystanie z bazy danych MySQL
- Tworzenie aplikacji modułowych
- Programowanie obiektowe w języku PHP

Zawarte w tej książce wiadomości pomogą każdemu programiście stworzyć doskonałą aplikację internetową.

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

Spis treści

Wprowadzenie **13**

Część I Programowanie na potrzeby sieci — podstawy **19**

Rozdział 1. Podstawy funkcjonowania internetu **21**

Protokoły transmisji danych 21
Rodzina TCP/IP 23
Adresowanie w sieci 24
Adresy IP 24
Nazwy domen 26
Porty 28
Terminologia 30
Serwery 30
Węzły 30
Porty 31
Demony sieciowe 31
Usługodawcy 31
Hosty 32
Hosty wirtualne 32
Dostawcy usług hostingowych 32
Witryny internetowe 33
Pliki HTML 33
Strony internetowe (lub strony HTML) 33
Programowanie na potrzeby sieci 33
Sieć WWW a adresy URL 34
Protokół transmisji danych 34
Nazwa hosta 35
Numer portu 35
Ścieżka dostępu do strony 35

Rozdział 2. Interfejs CGI **37**

Czym jest CGI? 37
Tajemnice adresów URL 38
Nagłówki a metoda GET 39
GET 41
POST 41
Content-Type 41
User-Agent 42

Referer ... 42
Content-length ... 42
Cookie ... 43
Accept ... 43
Imitowanie przeglądarki poprzez telnet ... 43
Metoda POST ... 44
Kodowanie URL ... 45
Formularze oraz ich zastosowanie ... 45
Ręczne podstawianie parametrów ... 46
Korzystanie z formularza ... 46
Względne i bezwzględne ścieżki dostępu do skryptu ... 47
Formularze a metoda POST ... 48

Rozdział 3. Szczegóły działania skryptów CGI ... 51
Przesyłanie dokumentu do użytkownika ... 52
Nagłówki odpowiedzi ... 53
Przykładowy skrypt CGI ... 55
Przekazywanie danych do skryptu CGI ... 56
Zmienne środowiskowe ... 57
Przekazywanie parametrów za pomocą metody GET ... 59
Przekazywanie parametrów za pomocą metody POST ... 59
Dekodowanie danych zapisanych w formacie adresu URL ... 61
Formularze ... 64
Określanie rodzajów pól danych za pomocą znacznika <input> ... 65
Tworzenie wieloliniowych pól tekstowych za pomocą znacznika <textarea> ... 69
Tworzenie list za pomocą znacznika <select> ... 70
Przesyłanie plików ... 71
Format danych ... 71
Znacznik przesłania pliku ... 73
Czym są dane cookie i do czego służą? ... 74
Zapis danych cookie ... 76
Pobieranie danych cookie z przeglądarki ... 78
Przykładowy program obsługi danych cookie ... 78
Autoryzacja ... 80

Część II Wybór i konfiguracja narzędzi — serwer Apache ... 83

Rozdział 4. Instalowanie serwera Apache ... 85
Po co komu serwer domowy ... 85
Oprogramowanie i materiały referencyjne ... 86
Instalacja serwera Apache ... 86
Etap 1. instalacja ... 87
Etap 2. edycja pliku konfiguracyjnego serwera Apache ... 88
Etap 3. testowanie serwera Apache ... 91
Hosty wirtualne serwera Apache ... 94

Rozdział 5. Instalowanie PHP i MySQL-a ... 99
Instalowanie PHP ... 100
Konfiguracja serwera Apache pod kątem współpracy z PHP ... 101
Testowanie PHP ... 102
Instalowanie modułów dodatkowych ... 103
Instalowanie MySQL-a ... 104
Testowanie MySQL-a ... 106

Część III Podstawy języka PHP 109

Rozdział 6. Cechy szczególne języka PHP 111
Interpreter kontra kompilator 112
Zalety i wady interpretera 113
Przykładowy program w PHP 115
Korzystanie z PHP w sieci 120

Rozdział 7. Zmienne, stałe i wyrażenia 123
Zmienne 123
Typy zmiennych 124
Stosowanie zmiennych 127
Ustalenie typu zmiennej 128
Nadawanie zmiennym typów 129
Operator przypisania 129
Zmienne referencyjne 130
Referencje stałe 130
Referencje znakowe 131
Konwencje 132
string 132
int, long 133
double, float 133
bool 133
array 133
list 133
object 133
void 133
mixed 134
Stałe 134
Stałe predefiniowane 134
Definiowanie stałych 135
Sprawdzenie istnienia stałej 136
Wyrażenia 136
Wyrażenia logiczne 137
Wyrażenia tekstowe 138
Operatory 141
Operatory arytmetyczne 141
Operatory tekstowe 141
Operatory przypisania 141
Operatory inkrementacji i dekrementacji 142
Operatory bitowe 143
Operatory porównania 143
Operator identyczności 144
Operatory logiczne 145
Operator wyłączenia komunikatów 145

Rozdział 8. Przetwarzanie danych z formularzy 147
Transmisja danych — linia poleceń 147
Formularze 149
Konwersja elementów kontrolnych formularza na zmienne 150
Konwersja zmiennych środowiskowych i plików cookie 152
Konwersja list 153
Konwersja tablic 154

Rozdział 9. Konstrukcje składniowe języka PHP 157
Struktura kontrolna if-else 157
Posługiwanie się składnią alternatywną 158
Przedwarunkowa pętla while 159
Powarunkowa pętla do-while 160
Pętla uniwersalna for 160
Wyrażenia break oraz continue 161
Niezwykłe zastosowania wyrażeń do-while oraz break 162
Pętla foreach 163
Struktura kontrolna switch-case 164
Wyrażenie require 165
Wyrażenie include 166
Translacja i problemy z użyciem wyrażenia include 166
Wyrażenia typu _once 167

Rozdział 10. Tablice asocjacyjne 171
Dynamiczne tworzenie tablic 172
Funkcja list() 173
Listy i tablice asocjacyjne 174
Funkcja array() i tablice wielowymiarowe 174
Przetwarzanie tablic 176
Dostęp za pomocą kluczy 176
Funkcja count() 176
Łączenie tablic 177
Pośrednie przeszukiwanie tablic 178
Bezpośrednie przeszukiwanie tablic 180
Tablice i ciągi znakowe 181
Serializacja 182

Rozdział 11. Funkcje i zakres widoczności zmiennych 185
Przykład funkcji 186
Ogólna składnia definicji funkcji 188
Instrukcja return 188
Parametry domyślne 189
Przekazywanie parametrów przez referencję 190
Zmienna liczba parametrów funkcji 191
Zmienne lokalne 193
Zmienne globalne 193
Tablica $GLOBALS 194
Zmienne statyczne 196
Rekurencja 197
Funkcje zagnieżdżone 197
Warunkowa definicja funkcji 199
Przekazywanie funkcji „przez referencję” 200
Zwracanie referencji przez funkcję 201
Przykład funkcji — Dump() 202
Kilka rad dotyczących wykorzystania funkcji 203

Część IV Standardowe funkcje PHP 205

Rozdział 12. Funkcje operujące na ciągach tekstowych 207
Konkatenacja ciągów tekstowych 208
Porównywanie ciągów tekstowych — instrukcja if-else 208
Funkcja przetwarzania pojedynczych znaków 210

Funkcje usuwania znaków spacji ... 211
Funkcje podstawowe ... 212
Przetwarzanie bloków tekstu ... 213
Funkcje konwersji znaków ... 215
Funkcje zmiany wielkości liter ... 217
Ustawianie parametrów lokalizacyjnych ... 218
Funkcje formatujące ... 219
Przetwarzanie danych binarnych ... 221
Funkcje mieszające ... 223
Opróżnianie bufora wyjściowego ... 225

Rozdział 13. Wykorzystanie tablic ... 227
Sortowanie tablic ... 227
Sortowanie względem wartości — funkcje asort() i arsort() ... 227
Sortowanie względem klucza — funkcje ksort() i krsort() ... 228
Sortowanie względem klucza — funkcja uksort() ... 228
Sortowanie względem wartości — funkcja uasort() ... 229
Odwrócenie kolejności elementów tablicy — funkcja array_reverse() ... 229
Sortowanie list — funkcje sort() i rsort() ... 229
Sortowanie list — funkcja usort() ... 230
Losowy rozkład elementów listy — funkcja shuffle() ... 230
Klucze i wartości ... 230
Zastępowanie fragmentu ciągu tekstowego ... 231
Łączenie tablic ... 232
Dzielenie tablicy ... 233
Wstawianie i usuwanie elementów ... 233
Zmienne i tablice ... 235
Tworzenie list o wartościach z określonego przedziału ... 237

Rozdział 14. Funkcje matematyczne ... 239
Wbudowane stałe ... 239
Funkcje zaokrąglające ... 239
Liczby losowe ... 240
Konwersja pomiędzy różnymi systemami liczbowymi ... 242
Wartości minimalne i maksymalne ... 243
Funkcje wykładnicze ... 243
Funkcje trygonometryczne ... 244

Rozdział 15. Obsługa plików ... 245
Pliki tekstowe i binarne ... 245
Otwieranie pliku ... 246
Konstrukcja or die() ... 249
Tymczasowe pliki bez nazw ... 250
Zamykanie pliku ... 250
Odczytywanie i zapisywanie plików ... 251
Odczyt i zapis bloków danych ... 251
Odczyt i zapis pojedynczych linii tekstu ... 252
Odczyt pliku CSV ... 252
Wskaźnik pliku ... 253
Funkcje określające typy plików ... 254
Określenie typu pliku ... 254
Określanie dostępności pliku ... 255
Określanie parametrów plików ... 256
Funkcje specjalistyczne ... 257

Obsługa nazw plików ... 257
Funkcje przetwarzania całych plików ... 259
Inne funkcje ... 261
Blokowanie plików ... 262
Rodzaje blokad ... 263
Blokowanie bez wstrzymywania pracy programu ... 266
Przykład licznika wizyt ... 266

Rozdział 16. Obsługa katalogów ... 269
Manipulowanie katalogami ... 269
Obsługa rekordów ... 270
Przykład — wyświetlanie struktury katalogów ... 272

Rozdział 17. Potoki i dowiązania symboliczne ... 275
Potoki ... 275
Dowiązania symboliczne ... 277
Dowiązania twarde ... 278

Rozdział 18. Uruchamianie programów zewnętrznych ... 279

Rozdział 19. Funkcje daty i czasu ... 283
Określanie czasu w formacie timestamp ... 283
Określanie daty ... 284
Kalendarz gregoriański ... 286

Rozdział 20. Przesyłanie wiadomości e-mail w PHP ... 289
Funkcja przesyłania wiadomości e-mail ... 289
Zaawansowane przesyłanie wiadomości e-mail — tworzenie funkcji „inteligentnej” ... 290

Rozdział 21. Współpraca z siecią WWW ... 291
Ustawianie nagłówków odpowiedzi ... 291
Tworzenie nagłówków ... 291
Zakaz użycia pamięci podręcznej ... 292
Pozyskiwanie nagłówków zapytań ... 292
Obsługa cookie ... 293
Założenia teoretyczne ... 293
Ustawianie cookie ... 294
Pozyskiwanie cookie ... 296
SSI i funkcja virtual() ... 296
Imitowanie funkcji virtual() ... 297

Rozdział 22. Podstawy wyrażeń regularnych w formacie RegEx ... 299
Zacznijmy od przykładów ... 299
Przykład pierwszy ... 299
Przykład drugi ... 300
Wnioski ... 300
Terminologia ... 301
Korzystanie z wyrażeń regularnych w języku PHP ... 301
Wyszukiwanie ... 301
Zastępowanie ... 301
Język RegEx ... 302
Proste znaki ... 302
Grupy zanegowane ... 304
Kwantyfikatory powtórzeń ... 305
Znaki urojone ... 306

Operator alternatywy ... 307
Nawiasy grupujące ... 307
Wydobywanie grup ... 307
Funkcje pomocnicze ... 310
Przykłady użycia wyrażeń regularnych ... 311
Nazwy plików i rozszerzenia ... 311
Nazwa pliku i katalogu ... 311
Test identyfikatora ... 311
Modyfikacja znaczników ... 312
Konwersja hiperłącza ... 312
Konwersja adresów e-mail ... 312
Pobranie wszystkich unikatowych wyrazów tekstu ... 313
Wnioski ... 314

Rozdział 23. Praca z obrazami ... 315
Uniwersalna funkcja GetImageSize() ... 316
Praca z obrazami i biblioteką GD ... 316
Przykład ... 317
Tworzenie obrazu ... 317
Pobranie atrybutów obrazu ... 318
Zapis obrazu ... 319
Korzystanie z kolorów w formacie RGB ... 320
Utworzenie nowego koloru ... 320
Pobranie najbliższego koloru ... 321
Przezroczystość ... 321
Pobranie komponentów RGB ... 322
Prymitywy graficzne ... 322
Kopiowanie obrazów ... 322
Prostokąty ... 323
Linie ... 323
Łuki ... 324
Wypełnienie dowolnego obszaru ... 324
Wieloboki ... 324
Dostęp do poszczególnych pikseli ... 325
Czcionki o stałym rozmiarze ... 325
Wczytywanie czcionek ... 325
Parametry czcionek ... 326
Nałożenie tekstu na obraz ... 326
Korzystanie z czcionek TrueType ... 327
Nałożenie tekstu na obraz ... 327
Pobranie granic tekstu ... 328
Przykład ... 328

Rozdział 24. Sterowanie interpreterem ... 331
Funkcje informacyjne ... 331
Ustawianie parametrów PHP ... 332
Parametr error_reporting ... 333
Parametr magic_quotes_gpc on/off ... 333
Parametr max_execution_time ... 334
Parametr track_vars on/off ... 334
Sterowanie błędami ... 334
Operator wyłączania błędów ... 335
Przykład zastosowania operatora @ ... 335

Wymuszanie zakończenia programu ... 336
Funkcje wyłączania skryptu ... 336
Generowanie kodu w trakcie wykonywania skryptu ... 337
Wykonywanie kodu ... 338
Generowanie funkcji ... 340
Sprawdzanie poprawności kodu ... 341
Inne funkcje ... 342

Rozdział 25. Sesje ... 343
Do czego przydają się sesje? ... 343
Sesje w praktyce ... 344
Inicjalizacja sesji ... 345
Rejestracja zmiennych ... 345
Identyfikator sesji i nazwa grupy ... 346
Nazwy grup sesji ... 346
Identyfikator sesji ... 347
Inne funkcje ... 348
Ustawienie procedur obsługi sesji ... 349
Omówienie procedur obsługi ... 349
Rejestracja procedur obsługi ... 350
Przykład — redefinicja procedur obsługi sesji ... 351
Sesje i cookies ... 353
Jawne wykorzystanie stałej SID ... 353
Niejawna zmiana łączy internetowych ... 354
Niejawna zmiana formularzy ... 355
Czy cookies powinny być stosowane w sesjach? ... 355

Rozdział 26. Praca z bazą danych MySQL ... 357
Wady stosowania plików ... 358
Podstawy MySQL-a ... 359
Łączenie z bazą danych ... 360
Obsługa błędów ... 360
Wysyłanie zapytań do bazy danych ... 361
Język SQL ... 361
Tworzenie tabeli ... 362
Usunięcie tabeli ... 365
Wstawienie rekordu ... 365
Usunięcie rekordu ... 366
Wyszukiwanie rekordu ... 366
Aktualizacja rekordu ... 366
Zliczenie rekordów spełniających wyrażenie ... 366
Pobranie unikatowych wartości rekordów ... 367
Pobranie wyników ... 367
Atrybuty wyniku ... 367
Pobranie pola wyniku ... 368
Pobranie wiersza wyniku ... 368
Pobranie informacji o wyniku ... 369
Przykład wykorzystania funkcji MySQL ... 371
Unikatowe identyfikatory w MySQL-u ... 371
Praca z tabelami ... 372

Rozdział 27. Funkcje sieciowe ... 375
Obsługa gniazd ... 375
Funkcje obsługujące DNS ... 377
Konwersja adresów IP na nazwy domen i odwrotnie ... 377
Prawidłowa konwersja adresu IP na nazwę domeny ... 378

Część V Techniki programistyczne 381

Rozdział 28. Umieszczanie plików na serwerze 383
Formularze multipart 383
Znacznik wyboru pliku 384
Umieszczanie plików na serwerze a bezpieczeństwo 384
Obsługa umieszczania plików w języku PHP 385
Proste nazwy dla elementów umieszczania plików 385
Przykład — album zdjęć 386
Złożone nazwy elementów 388
Problemy związane z nazwami złożonymi 388

Rozdział 29. Programy modułowe — tworzenie bibliotekarza 391
Wymagania 391
Bibliotekarz 392
Korzystanie z bibliotekarza 396
Automatyczne dołączanie bibliotekarza 398
Sposób pierwszy — wykorzystanie dyrektywy auto_prepend_file 398
Sposób drugi — instalacja uchwytu serwera Apache 399
Uchwyty serwera Apache 399
Przechwytywanie dostępu do nieistniejących stron 402
Powiązanie PHP z innymi rozszerzeniami 403
Rozwiązanie problemu pętli nieskończonej 404

Rozdział 30. Kod i szablon strony WWW 405
Ideologia 405
Podejście dwuwarstwowe 406
Szablon strony 407
Generator danych 408
Interakcja między generatorem danych a szablonem 409
Ograniczenia 410
Podejście trójwarstwowe 411
Szablon strony 411
Schematy modeli dwu- i trójwarstwowych 413
Interfejs 414
Jądro 414
Walidacja wprowadzanych danych 415
Zarządca szablonów 416
Tworzenie stron WWW w sposób tradycyjny 417
Rozdzielanie szablonu 418
Czym jest zarządca szablonów? 419
Opis zarządcy szablonów 420
Uchwyt serwera Apache dla zarządcy szablonów 426
Główny moduł zarządcy szablonów 426
Przechwytywanie strumienia wyjściowego 431
Stos buforów 432
Problemy związane z debugowaniem 433

Rozdział 31. Programowanie obiektowe w PHP 435
Klasy i obiekty 436
Właściwości obiektów 436
Metody 437
Klasa tabeli bazy danych MySQL 438
Dostęp do właściwości obiektu 440
Inicjalizacja obiektów — konstruktory 441

Destruktory ... 442
Dziedziczenie ... 442
Polimorfizm ... 444
Klasa tabeli bazy danych MySQL ... 445
Kopiowanie obiektów ... 454
Referencje i interfejsy ... 455
Zwracanie referencji do obiektu ... 456
Zwracanie interfejsu ... 457

Rozdział 32. Szablony listów e-mail ... 461
Miniaturowy zarządca szablonów ... 461
Wysyłanie i kodowanie listów ... 463
Przykład ... 465

Rozdział 33. Różne rady i wskazówki ... 469
Współdzielenie czasu ... 469
Korzystanie z przekierowania do samego siebie ... 472
Zabronienie buforowania stron przez przeglądarki ... 474
Kilka słów na temat opcji formularzy ... 475

Dodatki ... 477

Skorowidz ... 479

Rozdział 9.

Konstrukcje składniowe języka PHP

W niniejszym rozdziale omówimy konstrukcje składniowe języka PHP. O niektórych z nich, na przykład o wyrażeniu `if`, była już okazja wspomnieć wcześniej. Tym razem jednak zapoznamy się z pełnym wachlarzem konstrukcji. Nie jest ich zbyt wiele, co na pewno należy zaliczyć do plusów języka PHP. Jak pokazuje praktyka, im bardziej zwięzła składnia języka programowania, tym łatwiej się nim posługiwać. PHP jest doskonałym przykładem prawdziwości tej teorii.

Struktura kontrolna if-else

Zaczynamy od omówienia najprostszej konstrukcji, mianowicie wyrażenia warunkowego. Jego składnia wygląda tak:

```
if(wyrażenie_logiczne)
  wyrażenie_1;
else
  wyrażenie_2;
```

Konstrukcja ta funkcjonuje tak, że gdy spełniony jest warunek `wyrazenie_logiczne`, to następuje wykonanie wyrażenia `wyrazenie_1`. Gdy wspomniany warunek nie zostanie spełniony, wykonywane jest wyrażenie `wyrazenie_2`. Podobnie jak ma to miejsce w innych językach programowania, konstrukcję `else` można opuścić. Wówczas w przypadku niespełnienia warunku nie będzie podejmowana żadna akcja.

Oto przykład:

```
  if($a>=1&&$b<=10) echo "Wszystko w porządku";
else echo "Nieprawidłowa wartość!";
```

Gdy zachodzi potrzeba zagnieżdżenia kilku wyrażeń w obrębie jednego, używa się do tego celu nawiasów klamrowych, jak w poniższym przykładzie:

```
if($a>$b) { print "a jest większe od b"; $c=$b; }
elseif($a==$b) { print "a jest równe b"; $c=$a: }
else {print "a jest mniejsze od b"; $c=$b; }
```

Uwaga — w powyższym skrypcie nie błędu! Zamiast `else if` można równie dobrze pisać `elseif` (czyli bez spacji pomiędzy tymi dwoma poleceniami), ale prawdopodobnie większość programistów uzna pierwszy sposób zapisu za bardziej czytelny.

Konstrukcja `if-else` ma także składnię alternatywną:

```
if(wyrażenie_logiczne):
  wyrażenia;
elseif(inne_wyrażenie_logiczne):
  inne_wyrażenia;
else:
  wyrażenia_alternatywne;
endif
```

Zwróćmy uwagę na dwukropek (`:`)! Jeśli zapomnimy umieścić go w skrypcie, wygenerowany zostanie komunikat o błędzie. Jak poprzednio, bloki `elseif` oraz `else` można pominąć.

Posługiwanie się składnią alternatywną

W poprzednich rozdziałach wielokrotnie rozpatrywaliśmy przykłady umieszczania kodu HTML w skryptach. W tym celu należy po prostu zamknąć skrypt znakiem ?>, wstawić pożądany fragment kodu HTML, po czym kontynuować skrypt po uprzednim zasygnalizowaniu jego początku znakiem <?.

Co prawda taki sposób zapisu nie wygląda zbyt elegancko, ale przy odrobinie wysiłku można to zmienić. Wystarczy sięgnąć po składnię alternatywną oraz inne konstrukcje języka.

Często zachodzi potrzeba umieszczenia kodu programu w obrębie dokumentu HTML, rzadziej dzieje się odwrotnie. Tak jest wygodniej, na przykład, dla projektanta grafiki strony, który jest odpowiedzialny za wygląd końcowego dzieła, a nie za pisanie skryptów. Dlatego właśnie powinno się dążyć do oddzielania kodu HTML od skryptu. Można to osiągnąć przez zapis kodu w odrębnym pliku, który będzie w odpowiednim momencie dołączany do skryptu w rezultacie użycia wyrażenia `include` (opis w dalszej części rozdziału). Do zagadnienia tego jeszcze powrócimy.

Poniżej przedstawiamy alternatywny sposób zapisu skryptu powitalnego zaprezentowanego w poprzednim rozdziale. Korzystamy z konstrukcji `if-else` (listing 9.1).

Listing 9.1. *Konstrukcja if-else. Alternatywny sposób zapisu*

```
  <?if(@$go) :?>
Cześć, <?=$name?>!
  <?else:?>
<form action=<?=$REQUEST_URI?> method=post>
Twoje imię: <input type=text name=name><br>
<input type=submit name=go value="Prześlij!">
  <?endif?>
```

Czytelnicy zapewne zgodzą się z tym, że skrypt w takiej postaci jest zrozumiały nawet dla osoby, która nie zna PHP, ale, na przykład, orientuje się w kodzie HTML.

Przedwarunkowa pętla while

Ta konstrukcja to kolejny spadek po języku C. Jej zadanie polega na wielokrotnym wykonywaniu wyrażeń zagnieżdżonych w pętli tak długo, jak długo spełniony będzie warunek wyrażony w sprawdzonym uprzednio wyrażeniu logicznym nagłówka pętli. W tym czasie konstrukcja wykonuje kolejne iteracje, ale natychmiast przerywa pracę, gdy tylko wyrażenie logiczne zwróci wartość `false`. Pętla ma następującą składnię:

```
while(wyrażenie_logiczne)
  wyrażenie;
```

Znaczenie powyższej definicji jest oczywiste; nazwa `wyrażenie_logiczne` mówi sama za siebie, natomiast `wyrażenie` to pojedyncze lub złożone wyrażenie wchodzące w skład pętli (oczywiście w celu niedopuszczenia do powstania pętli nieskończonej konieczne są pewne manipulacje mające na celu zmianę wartości wyrażenia — można w tym celu posłużyć się licznikiem). Jeśli wartością zwracaną przez wyrażenie będzie od początku `false`, to pętla nigdy nie zostanie wykonana. Oto przykład:

```
$i=1; $p=1;
while($i<32) {
  echo $p," ";
  $p=$p*2; // można by napisać $p*=2
  $i=$i+1; // można by napisać $i+=1 lub nawet $i++
}
```

Skrypt wyświetla liczby stanowiące wynik działania podnoszącego 2 do potęgi z zakresu od 1 do 31. Gdy zmiennej `$i` przypiszemy wartość `32`, żadna operacja nie zostanie wykonana.

Podobnie jak wyrażenie `if`, także pętla `while` ma składnię alternatywną, która ułatwia łączenie skryptów z kodem HTML:

```
while(wyrażenie_logiczne):
  wyrażenia;
endwhile;
```

Powarunkowa pętla do-while

W przeciwieństwie do pętli `while`, pętla `do-while` sprawdza wyrażenie logiczne *po* wykonaniu każdej iteracji, a nie przed. Z tego wniosek, że pętla ta musi być wykonana przynajmniej raz. Składnia wyrażenia jest następująca:

```
do {
  wyrażenia;
} while(wyrażenie_logiczne);
```

Po każdej iteracji sprawdzana jest prawdziwość wyrażenia logicznego — jeśli warunek jest spełniony, pętla wykonywana jest ponownie. W przeciwnym razie następuje przerwanie procesu.

Tego rodzaju pętla nie ma składni alternatywnej, co wynika najprawdopodobniej stąd, że ma ona nieczęste zastosowanie w skryptach.

Pętla uniwersalna for

Określenie „uniwersalna" nie wzięło się z powietrza — pętla `for` w PHP pozwala programistom tworzyć konstrukcje znacznie mniej trywialne w porównaniu z jej odpowiednikami z języków Pascal oraz C (tam pętla `for` służy jedynie do przechodzenia przez kolejne wartości zliczane przez licznik). Składnia pętli wygląda następująco:

```
for(wyrażenia_inicjujące; warunek_pętli; wyrażenia_po_iteracji)
  ciało_pętli;
```

Sprawdźmy teraz, jak to działa. W momencie uruchomienia pętli wykonywane są wyrażenia określone jako `wyrażenia_inicjujące` (w kolejności od lewej do prawej). Oddziela się je przecinkami, jak w poniższym przykładzie:

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

Dalej rozpoczyna się iteracja. Najpierw sprawdzany jest warunek pętli (jak w konstrukcji `while`) i, w przypadku jego spełnienia (wartość `true` (prawda)), następuje wykonanie iteracji. Jeśli warunek nie zostanie spełniony, interpreter zakończy pętlę. Przykład:

```
// dodaje kolejną kropkę
for($i=0,$j=0,$k="Test"; $i<10; .....) $k.=".";
```

Załóżmy, że wykonana została jedna iteracja. W takim razie powinno dojść do wykonania wyrażeń po iteracji, których format jest identyczny jak wyrażeń inicjujących. Przykład:

```
for($i=0,$j=0,$k="Punkty"; $i<100; $j++,$i+=$j) $k=$k.".";
```

W tym miejscu należy wspomnieć, że powyższy przykład (jak również dowolny inny) można zaimplementować także z wykorzystaniem pętli `while`. Taka konstrukcja nie będzie jednak wyglądać równie dobrze jak pętla `for`. Oto przykład:

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

Cóż, to już chyba wszystko… Nie, chwileczkę! Spróbuj jeszcze zgadnąć, ile kropek zostanie dodanych do zmiennej `$k` po zakończeniu wykonywania pętli.

Pętla `for` — podobnie jak większość pozostałych konstrukcji — ma składnię alternatywną:

```
for(wyrażenia_inicjujące; warunek_pętli; wyrażenia_po_iteracji):
  wyrażenia;
endfor;
```

Wyrażenia break oraz continue

Kontynuujemy dyskusję na temat pętli. Czasem zachodzi potrzeba przerwania bieżącej iteracji (na przykład gdy zostanie spełniony określony warunek). W tym celu zaprojektowano wyrażenie `break`, które natychmiast kończy pracę pętli. Wyrażenie to może przyjmować parametr opcjonalny, służący do definiowania liczby zagnieżdżonych pętli, które należałoby przerwać. Domyślną wartością tego parametru jest `1` i wskazuje ona na pętlę bieżącą. Od czasu do czasu używa się jednak i innych wartości:

```
for($i=0; $i<10; $i++) {
  for($j=0; $j<10; $j++) {
    if($A[$i]==$A[$j]) break(2);
  }
}
if ($i<10) echo "W tablicy \$A znaleziono pasujące elementy!";
```

W powyższym przykładzie wyrażenie `break` przerywa zarówno drugą, jak i pierwszą pętlę, ponieważ jego parametr ma wartość `2`.



Opisywaną tutaj funkcję wprowadzono po raz pierwszy w czwartej wersji PHP. Autor niniejszej książki przyznaje, że nigdy dotąd nie pracował z językiem o tak przejrzystej składni — podziękowania dla twórców PHP.

Wyrażenie `break` przydaje się zwłaszcza wtedy, gdy trzeba coś wyszukać. W momencie spełnienia warunku wyszukiwania przez którąkolwiek z iteracji następuje przerwanie wykonywania pętli. Oto przykład pętli wyszukującej pierwszy zerowy element w tablicy `$A`:

```
for($i=0; $i<count($A); $i++)
  if($A[$i]==0) break;
if ($i<count($A)) echo "Znaleziono element zerowy: i=$i";
```

Nowością jest tutaj standardowa funkcja `count()`; zwraca ona liczbę elementów w tablicy.

Kolejne wyrażenie, `continue`, ma pewną cechę wspólną z wyrażeniem `break` — może być używane tylko „w kombinacji" z pętlą. Jego zadanie polega na natychmiastowym zakończeniu bieżącej iteracji i wykonaniu kolejnej (ale — co oczywiste — tylko w przypadku

spełnienia warunku postawionego w pętli przedwarunkowej). Kolejne podobieństwo do wyrażenia `break` polega na możliwości wskazania poziomu zagnieżdżenia pętli, do którego wykonywane będzie wyrażenie `continue`.

Jedną z najważniejszych zalet wyrażenia `continue` jest możliwość znacznego zredukowania liczby klamer, co wpływa na poprawę czytelności skryptu. Przydaje się to w pętlach filtrujących, gdzie wybiera się wyłącznie obiekty ściśle odpowiadające określonym warunkom. Oto przykład pętli, która przypisuje zera do elementów tablicy spełniających specyficzne warunki:

```
for($i=0; $i<count($A); $i++) {
  if(!warunek1($A[$i])) continue;
  ...
  if(!warunekN($A[$i])) continue;
  $A[$i]=0;
}
```



Poprawne korzystanie z wyrażeń `break` oraz `continue` może pozytywnie wpłynąć na czytelność skryptu, zmniejszając przy okazji ogólną liczbę bloków `else`. Oczywiście przedstawione tutaj przykłady ich użycia nie są najwyższych lotów, ale pozwalają się zorientować, że prędzej czy później każdy programista dojdzie do wniosku, że nie potrafi się bez nich obejść.

Niezwykłe zastosowania wyrażeń do-while oraz break

Stosowanie wyrażenia `break` daje ciekawy efekt uboczny, przydatny zwłaszcza wówczas, gdy trzeba ominąć w skrypcie „niepotrzebne" elementy (nawiasem mówiąc, efekt ten da się również wykorzystać w języku C). Taka potrzeba omijania zdarza się zadziwiająco często. Rozważmy następujący przykład (listing 9.2).

Listing 9.2. *Przykładowy skrypt do przetwarzania formularzy*

```
...
$WasError=0; // Wskaźnik błędu: wartość inna niż zero oznacza błąd
// Gdy zostanie kliknięty przycisk potwierdzenia (o nazwie $doSubmit)...
if(@$doSubmit) do {
  // Sprawdzanie danych wejściowych
  if(nieprawidłowa_nazwa_użytkownika) { $WasError=1; break; }
  ...
  if(nieprawidłowe_dane) { $WasError=1; break; }
  ...
  // Dane są prawidłowe, przetwórz je
  wykonaj działania;
  wyświetl rezultaty;
  zakończ skrypt;
} while(0);
...
  // Wyświetl formularz, z którego użytkownik skorzysta w celu uruchomienia skryptu.
  // Możliwe, że należy wyświetlić komunikat o błędzie, jeśli $WasError!=0
```

Jest to typowy sposób pisania skryptów dialogowych. Skrypt uruchamiany jest bez żadnych parametrów i użytkownik widzi formularz pytający o jego imię, hasło i inne tego typu informacje. Po kliknięciu przycisku następuje wykonanie operacji zaprogramowanych w skrypcie. Polegają one na sprawdzeniu, czy kliknięto przycisk `doSubmit`, a także czy podano nazwę i hasło. Jeśli dane te są nieprawidłowe, formularz wyświetlany jest ponownie (z komunikatem o błędzie podświetlonym na czerwono). Gdy zaś wszystko jest w porządku, skrypt kończy swoje działanie i wyświetla rezultaty.

Jak widać, przedstawioną tutaj logikę dość łatwo zaadaptować do konkretnej sytuacji, o ile tylko znamy metodę przerywania wykonywania bloku sprawdzającego i ponownego wyświetlania formularza. Osiągamy to przez użycie konstrukcji:

```
if(warunek) do {...} while(0);
```

Oczywiście pętla `do-while` wykonywana jest tylko raz, jako że jej warunek jest zawsze fałszywy. Takiej „zwyrodniałej" pętli można jednak użyć w sytuacji, gdy zajdzie potrzeba zakończenia jej wyrażeniem `break`.

Niektórzy Czytelnicy mogą w tym momencie stwierdzić, że w podobnych przypadkach bardziej odpowiednie byłoby użycie funkcji i wyrażenia `return`, lecz to jest właśnie przykład sytuacji, w której korzystanie z funkcji nie jest zbyt wygodne. Chodzi o to, że aby pozyskać zmienną globalną (w rodzaju elementu formularza) z funkcji, trzeba wykonać dodatkowe operacje. Jest to oczywiście wadą PHP i wkrótce przedyskutujemy ją szerzej.

Pętla foreach

Ten rodzaj pętli zaprojektowano specjalnie z myślą o przeglądaniu poszczególnych elementów tablicy. Ma ona następującą składnię:

```
foreach(tablica as $klucz=>$wartość)
  wyrażenia;
```

W powyższym przypadku wyrażenia są wykonywane kolejno dla każdego elementu tablicy, a bieżąca para `klucz=>wartość` przypisywana jest do zmiennych `$klucz` oraz `$wartość`. Rozważmy teraz inny przykład (listing 9.3), w którym do wyświetlenia wszystkich zmiennych globalnych posługujemy się konstrukcją `foreach`.

Listing 9.3. *Wyświetlanie zmiennych globalnych*

```
<?
foreach($GLOBALS as $k=>$v)
echo "<b>$k</b> => <tt>$v</tt><br>\n";
?>
```

Pętla `foreach` ma składnię alternatywną, przydatną zwłaszcza wtedy, gdy nie potrzebujemy wartości klucza bieżącego elementu. Ten alternatywny format zapisu przedstawia się następująco:

```
foreach(tablica as $wartość)
  wyrażenia;
```

W tym przypadku dostępna jest wyłącznie *wartość* elementu, natomiast klucz — nie. Jest to użyteczne rozwiązanie, na przykład, podczas pracy z tablicami list.



Pętla `foreach` nie korzysta z prawdziwej tablicy, lecz z jej kopii. Oznacza to, że wszelkie zmiany wprowadzone w tablicy nie będą odzwierciedlane w pętli. Można jednak użyć rezultatu wykonania funkcji jako tablicy. W takim wypadku funkcję wywołuje się tylko raz, przed rozpoczęciem pętli, a wszelkie następujące później manipulacje są już wykonywane na kopii zwróconej wartości.

W następnym rozdziale przyjrzymy się nieco uważniej zagadnieniom dotyczącym tablic asocjacyjnych.

Struktura kontrolna switch-case

W wielu wypadkach wygodniej jest posłużyć się specjalną konstrukcją `switch-case` w miejsce rozbudowanych wyrażeń `if-else`:

```
switch(wyrażenie) {
  case wartość1: wyrażenia1; [break;]
  case wartość2: wyrażenia2; [break;]
  ...
  case wartośćN: wyrażeniaN; [break;]
  [default: domyślne_wyrażenia; [break;]]
}
```

Konstrukcja `switch-case` działa w sposób następujący: najpierw szacowana jest wartość wyrażenia (załóżmy, że jest to wartość `V`), a następnie wyszukiwana jest linia rozpoczynająca się od `case V:`. Po jej odnalezieniu wykonywane są kolejne wyrażenia (w tym czasie wszystkie pojawiające się dalej wyrażenia typu `case` są ignorowane, w przeciwieństwie do pozostałych fragmentów kodu). Gdy pożądana linia nie zostanie odnaleziona, wykonywane są wyrażenia pojawiające się po wyrażeniu `default` (o ile jakieś istnieją).

Zwróćmy uwagę na obecność wyrażeń `break` (zapisano je w nawiasach, co oznacza, że są opcjonalne). Wstawia się je po każdej grupie wyrażeń (`wyrażenia1`, `wyrażenia2` itd.) z wyjątkiem ostatniej — dodanie wyrażenia `break` poniżej ostatniej linii jest możliwe, ale nie ma większego sensu. W przypadku zaistnienia warunku `V=wartość1` mogą być wykonane wszystkie wyrażenia, a nie tylko te z grupy pierwszej (`wyrażenia1`).

Składnia alternatywna dla konstrukcji `switch-case` prezentuje się następująco:

```
switch(wyrażenie):
  case wartość1: wyrażenia1; [break;]
  ...
  case wartośćN: wyrażeniaN; [break;]
  [default: domyślne_wyrażenia; [break;]]
endswitch;
```

Wyrażenie require

Wyrażenie `require` umożliwia dokonanie podziału tekstu skryptu na części, z których każda tworzy oddzielny plik. Składnia wyrażenia wygląda następująco:

```
require nazwa_pliku;
```

Rozpoczynając pracę (ale nie wykonanie programu!), interpreter zastępuje wyrażenie plikiem o nazwie `nazwa_pliku` (może to być plik zawierający skrypt PHP ograniczony znacznikami <? i ?>). Co w tym wszystkim najważniejsze, operacja zastępowania wyrażenia plikiem wykonywana jest *tylko jeden raz* (czyli zupełnie inaczej niż ma to miejsce w przypadku wyrażenia `include` opisywanego w dalszej części rozdziału) przed rozpoczęciem wykonywania programu. To bardzo wygodne rozwiązanie, szczególnie gdy zachodzi potrzeba dołączenia do skryptu różnych nagłówków i (lub) stopek HTML. Przykłady znajdziemy na listingach 9.4, 9.5 i 9.6.

Listing 9.4. *Plik naglowek.htm*

```
<html>
<head><title>Temat!</title></head>
<body bgcolor=yellow>
```

Listing 9.5. *Plik stopka.htm*

```
&copy;Moja firma, 1999.
</body></html>
```

Listing 9.6. *Plik skrypt.php*

```
<?
require "naglowek.htm";
... skrypt wyświetlający dokument
require "stopka.htm";
?>
```

Jest to o wiele lepsze rozwiązanie od zwykłego przetykania wyrażeń skryptu fragmentami kodu HTML. Umożliwia, na przykład, wygodną zmianę wyglądu strony osobom, które korzystają z usług naszego programu. Z drugiej strony, nie jest to programowanie w dobrym stylu — w końcu mamy tu do czynienia z aż trzema różnymi plikami! Tymczasem, jak wspomniano wcześniej, im mniej plików składa się na cały program, tym łatwiej nim operować (szczególnie z punktu widzenia osoby, która nie potrafi posługiwać się PHP, czyli na przykład projektanta graficznego).

Przy okazji lektury rozdziału 30. poznamy technikę rozdzielania kodu i szablonów, która będzie znakomitym lekarstwem na tego typu problemy.

Wyrażenie include

Wyrażenie `include` jest niemal identyczne z wyrażeniem `require`, z jednym wszakże wyjątkiem. Otóż dołączany plik staje się częścią skryptu nie przed jego wykonaniem, lecz właśnie w czasie wykonywania.

Jak to różnica? Już wyjaśniamy. Załóżmy, że mamy dziesięć plików tekstowych, nazwanych odpowiednio *plik0.php*, *plik1.php*, *plik2.php* i tak dalej, aż do *plik9.php*. Każdy z tych plików zawiera cyfrę z zakresu od 0 do 9. Uruchamiamy więc następujący program:

```
for($i=0; $i<10; $i++) {
  include "plik$i.php";
}
```

i w rezultacie jego wykonania otrzymujemy zestaw dziesięciu cyfr: `"0123456789"`. Widać z tego, że podczas wykonywania pętli każdy z plików tekstowych został jednokrotnie dołączony do skryptu (spróbujmy teraz zastąpić wyrażenie `include` wyrażeniem `require`, a następnie porównajmy rezultaty).

Być może Czytelnik zwrócił uwagę na to, że klamry wydają się tu zbędne. Ale to pozory. Gdy tylko je usuniemy, zgłoszony zostanie dziwny komunikat o błędzie (lub, co gorsza, program zacznie funkcjonować nieprawidłowo bez żadnej widocznej przyczyny). Dlaczego tak się dzieje? Ponieważ wyrażenie `include` nie jest bynajmniej wyrażeniem sensu stricte. Aby to zrozumieć, wyobraźmy sobie, że za każdym razem, gdy interpreter napotka wyrażenie `include`, po prostu dołącza do skryptu zawartość odpowiedniego pliku. Lecz co się stanie, jeśli ów plik będzie zawierać więcej niż jedno wyrażenie? Tylko pierwsze z nich zostanie wykonane podczas odtwarzania pętli, natomiast pozostałe zostaną uwzględnione dopiero *po* zakończeniu pętli. Reguła jest zatem taka: jeśli tylko używamy wyrażenia `include` w obrębie inne konstrukcji, to powinniśmy otoczyć je klamrami.



Istnieje możliwość, że w nowszych wersjach PHP sytuacja ulegnie poprawie, ale raczej trudno uznać to za pewnik.

Translacja i problemy z użyciem wyrażenia include

Jak już wiemy, skrypt PHP jest przed wykonaniem przekształcany na jego wewnętrzną reprezentację. Po zakończeniu tego procesu w pamięci komputera znajduje się już na wpół „upieczony” program, bez komentarzy, niektórych nazw zmiennych, formatowania itd. Potem następuje interpretacja, czyli wykonanie tej wewnętrznej reprezentacji skryptu. Wiemy jednak, że mogą istnieć takie fragmenty programu, których nie da się poddać translacji z wyprzedzeniem — jest ona wykonywana później, gdy interpreter napotka takie fragmenty podczas wykonywania skryptu.

Jednym z takich „trudnych" fragmentów kodu jest wyrażenie `include`. Gdy tylko interpreter napotka je w skrypcie, zatrzymuje jego wykonanie i wywołuje translator, którego zadaniem jest przetworzenie zawartości dołączanego pliku. Zmniejsza to szybkość działania programu, szczególnie gdy mamy do czynienia z naprawdę dużym plikiem. Wniosek z tego taki, że gdy pracujemy nad rozległym i bardzo złożonym skryptem, powinniśmy unikać wyrażenia `include`, zastępując je — gdzie to tylko możliwe — wyrażeniem `require`.

Powyższa rada zyskuje na znaczeniu, jako że mamy w perspektywie możliwość pojawienia się któregoś dnia kompilatora PHP. Cechowałby się on zdolnością przechowywania przetłumaczonego kodu w plikach wykonawczych (tak jak ma to miejsce w programach tworzonych w języku Perl). Wówczas użycie wyrażenia `include` powodowałoby zamieszanie, bo kompilator nie „wiedziałby", gdzie szukać żądanych plików i ostatecznie nie mogłyby one trafić do plików wykonawczych.

Które z wyrażeń jest zatem lepsze: `require` czy `include`? Gdy jesteśmy pewni, że dany plik powinien być dołączony tylko raz i w jednym konkretnym miejscu, skorzystajmy z wyrażenia `require`. W innym wypadku korzystniejsze może być użycie wyrażenia `include`.

Wyrażenia typu _once

Wyrażeń `include` i `require` używa się najczęściej w rozległych i złożonych skryptach. Z tego względu odnajdywanie wszystkich miejsc wstawienia tego samego pliku może sprawiać nie lada trudności (nie wspominając już o wynikających z tego błędach).

Żeby mieć lepsze wyobrażenie na temat problemu, z którym mamy do czynienia, rozważymy taką oto historię. Pewnego dnia programista Grzegorz napisał kilka użytecznych funkcji do obsługi plików programu Excel. Następnie podjął decyzję, że zintegruje je z biblioteką o nazwie *xllib.php* (listing 9.7).

Listing 9.7. *Biblioteka xllib.php*

```
<?
function LoadXlDocument($filename) {...}
function SaveXlDocument($filename,$doc) {...}
?>
```

Programista Antek postanowił stworzyć podobne funkcje dla programu Word, co zaowocowało powstaniem kolejnej biblioteki, o nazwie *wlib.php*. Ponieważ programy Excel i Word mają ze sobą wiele wspólnego, Antek postanowił skorzystać z biblioteki *xllib.php*, odwołując się do niej za pomocą wyrażenia `require` (listing 9.8).

Nie minęło wiele czasu i obydwie biblioteki zyskały sobie taką popularność, że zaczęli z nich korzystać inni programiści tworzący na potrzeby sieci. Przy tym nikt nie zawracał sobie głowy konstrukcją bibliotek; po prostu dołączali je do skryptów za pomocą wyrażenia `require`.

Listing 9.8. *Biblioteka wlib.php*

```
<?
require "xllib.php";
function LoadWDocument($filename) {...}
function SaveWDocument($filename,$doc) {...}
?>
```

Pewnego pięknego dnia, bliżej nikomu nieznany programista postanowił popracować zarówno z dokumentami Excela, jak i Worda. Nie namyślając się wiele, dołączył do skryptu obydwie biblioteki (listing 9.9).

Listing 9.9. *Dołączanie bibliotek xllib.php i wlib.php*

```
<?
require "wlib.php";
require "xllib.php";
$wd=LoadWDocument("dokument.doc");
$xd=LoadXlDocument("dokument.xls");
?>
```

Wyobraźmy sobie zdziwienie na twarzy tego programisty, gdy po uruchomieniu skryptu jego oczom ukazał się komunikat o błędzie. Komunikat ten głosił, że funkcja `LoadXlDocument()` w pliku *`xllib.php`* została zdefiniowana więcej niż raz!

W czym tkwi problem? Łatwo to ustalić, gdy prześledzi się sposób, w jaki translator PHP „tworzy" kod z listingu 9.9. Spójrzmy:

```
//require "wlib.php";
  //require "xllib.php";
    function LoadXlDocument($filename) {...}
    function SaveXlDocument($filename,$doc) {...}
  function LoadWDocument($filename) {...}
  function SaveWDocument($filename,$doc) {...}
//require "xllib.php";
  function LoadXLDocument($filename) {...}
  function SaveXLDocument($filename,$doc) {...}
$wd=LoadWDocument("dokument.doc");
$wd=LoadXlDocument("dokument.xls");
```

Jak widać, plik *`xllib.php`* został dołączony dwukrotnie: najpierw — pośrednio — przez bibliotekę *`wlib.php`*, a potem już bezpośrednio przez sam program. Dlatego właśnie translator zgłosił błąd, gdy napotkał linię oznaczoną w powyższym skrypcie pogrubioną czcionką.

Oczywiście autor nieszczęsnego skryptu mógłby wcześniej przeanalizować plik źródłowy biblioteki *`wlib.php`* i wówczas na pewno odkryłby, że nie powinien dopuścić do dwukrotnego dołączenia pliku *`xllib.php`*. Nie takiego jednak rozwiązania oczekujemy. Gdy dochodzi do pośredniego, zagnieżdżonego dołączania plików, jedynym sposobem na rozwikłanie problemu wydaje się modyfikacja bibliotek. Ale tego nie chcemy. Co więc powinniśmy zrobić?

Cóż, po tak długim wprowadzeniu (ale chyba nie za długim?), czas poznać opinię twórców PHP. Otóż oferują oni bardzo eleganckie rozwiązanie wszystkich naszych bolączek: wyrażenia `include_once` oraz `require_once`.

Wyrażenie `require_once` działa niemal identycznie jak wyrażenie `require` z jednym tylko, lecz zasadniczym, wyjątkiem. Gdy okaże się, że żądany plik został już wcześniej dołączony do skryptu, wyrażenie nie robi nic. Oczywiście takie rozwiązanie wymaga od interpretera PHP przechowywania pełnych nazw wszystkich dołączonych plików. I tak też się dzieje.

Podobnie rzecz się ma z wyrażeniem `include_once` z tą różnicą, że działa ono w fazie wykonywania programu, a nie w fazie jego translacji.



Jak już wspomniano, w PHP dostępna jest wewnętrzna tablica, w której przechowywane są pełne nazwy wszystkich dołączonych plików. Zawartość tej właśnie tablicy sprawdzana jest przez wyrażenia `include_once` oraz `require_once`. Co więcej, *dodawanie* wartości do tej tablicy jest także realizowane przez wyrażenia `require` i `include`. I właśnie dlatego po dołączeniu pliku do skryptu, na przykład przez wyrażenie `require`, żądanie ponownego dołączenia tego samego pliku przez wyrażenie `require_once` jest po prostu ignorowane.

Wyrażeń z przyrostkiem `_once` starajmy się używać wszędzie, gdzie to tylko możliwe, natomiast unikajmy jak ognia wyrażeń `require` oraz `include`. W ten sposób łatwiej nam będzie dzielić złożone skrypty na mniejsze i względnie niezależne fragmenty.