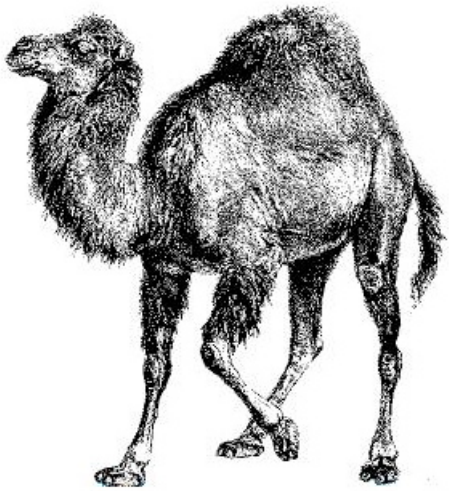
Język programowania PERL – wprowadzenie.

Prezentacja przygotowana przez: **Łukasza Nowakowskiego**
zaprezentowana na seminarium dyplomowym PWSZ Tarnów, dn. 08.05.2006r.



SPIS TREŚCI:

1. Co to takiego Perl?

Rysunek obok dobrze ilustruje istotę PERLa. Jest to bowiem połączenie elementów języków takich jak C, awk, sed, grep i Bourne shell. Perl jest doskonałym narzędziem do obróbki tekstu i plików binarnych, pobierania z nich informacji i generowania komunikatów na ich podstawie. Stąd też wzięła się jego nazwa, będąca skrótem od **Practical Extraction and Report Language (Praktyczny Język Pobierania Danych i Raportowania)**. Najczęściej stosowany do pisania skryptów CGI. Jego autorem jest Larry Wall wspomagany oczywiście przez ogromną liczbę innych programistów.

Perl należy do tych języków programowania, których można się nauczyć szybko. Perl nie wymaga deklarowania typów zmiennych przed ich użyciem. Wystarczy „po prostu” napisać, co ma być zrobione. Warto więc już na początku zapamiętać, ze Perl nie jest najlepszy do wszystkiego – w szczególności nie należy rozwiązywać przy jego pomocy skomplikowanych problemów wymagających użycia złożonych struktur danych. Łatwo natomiast przetwarzać dane binarne.

Kalendarium - historia Perla.

- 1986 - początki pracy nad nowym językiem, twórcą jest Larry Wall
- 1987 - ogłoszony zostaje perl-1.0,
- 1988 - perl-2.0,
- 1989 - perl 3.0,
- 1991 - perl-4.0; pierwsza edycja książki Perl-Programowanie (z wielbłądem),
- 1994 - perl-5.0 (pierwsza wersja języka z obiektami),
- 2002 - perl 5.8,
- 02.02.2006 – **perl 5.8.8**, jednocześnie rozwijany jest Perl 6, który będzie działał używając maszyny wirtualnej Parrot

2. Cechy Perla:

- nazywany **językiem zarządzania systemem**, gdyż może zastąpić dotychczasowe skrypty shella,

- w języku angielskim istnieje inne złośliwe rozwinięcie skrótu Perl: **"Pathologically Eclectic Rubbish Lister".** Wynika to z użycia wielu znaków przestankowych w składni języka,

- **kompilatory** Perla są **bezpłatne** i dostępne dla wielu systemów operacyjnych. Większość skryptów jest przenośna. Sam Perl to wolne oprogramowanie, dostępne pod licencjami GPL i artystyczną,

- stosunkowo **łatwy w użyciu** oraz **wydajny**, przypłacając to może nieco elegancją i czytelnością,

- Perl **nie jest językiem kompilowanym**, jednak jest szybszy od większości języków interpretowanych. Perl jest językiem skryptowym, tzn. tworzone w nim programy są po prostu plikami tekstowymi, które następnie są wykonywane przez interpreter Perla,

- oprócz programów wykonywanych "wiersz po wierszu" pozwala on konstruować skomplikowane struktury danych i programować w stylu obiektowym,

- hasło przewodnie „**Każde zadanie można wykonać na więcej niż jeden sposób**”. Powoduje to, iż programista może opracować swój własny koncept myślenia i własny styl programowania.

- w celu rozróżnienia język Perl pisze się dużą, zaś nazwę programu, małą litera,

- w zastosowaniach www Perl **wykorzystywany** jest **po stronie serwera**.

3. Wymagania.

Perl 5 jest dostępny na prawie wszystkie systemy Unixowe, wchodzi w skład praktycznie wszystkich dystrybucji Linuxa, można go używać pod Dos-em (bardzo okrojona wersja), w systemie Windows, Macintosh System 7, Novell Netware i wielu innych. Skrypty te są na ogół przenośne między różnymi systemami, o ile nie wykorzystują cech konkretnego systemu lub nie korzystają z programów na danym systemie.

Instalacja

Aby móc uruchamiać skrypty Perla, trzeba mieć go zainstalowanego. Ponieważ jest on dostępny za darmo, pobranie go z sieci nie stanowi żadnego problemu. Na stronie http://www.perl.com/download.csp znajdują się informacje o najnowszych wersjach Perla dla każdej platformy. Do uruchamiania skryptów Perla w systemie Windows polecam interpretator Active-Perl. http://www.activestate.com/ASPN/Downloads/ActivePerl/. Ponadto na stronie http://www.kt.agh.edu.pl/other/perl/faq/ znajdują się odpowiedzi na najczęściej zadawane pytania na temat Perla (FAQ).

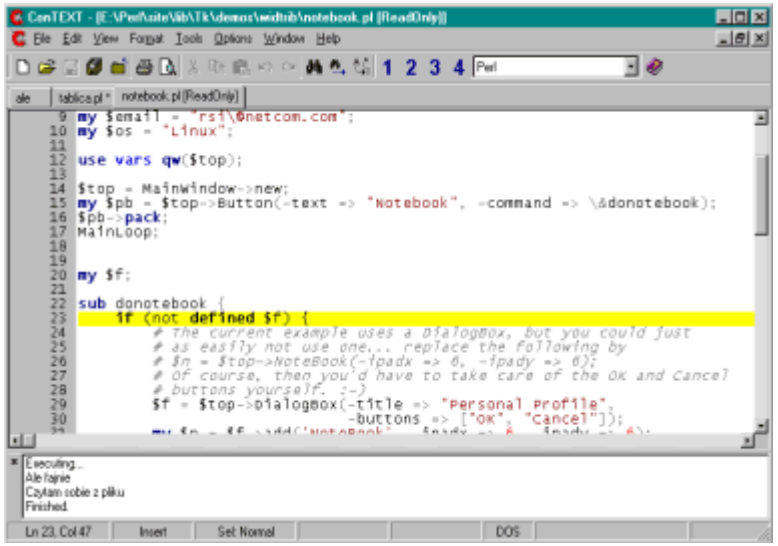
Edytory

Do pisania w Perlu wystarczy zwykły edytor tekstowy (np. notatnik), ale dużo łatwiej pisze się w edytorze z podświetlaną składnią.

Jednym z nich jest **EditPlus - Text Editor**. Nie dość, że działa w nim kolorowanie składni Perla, PHP, ASP, Javascript, HTML to jest w pełni konfigurowalny. Niestety posiada jeden mankament. Wersja Shareware działa tylko 30 dni - potem należy albo program usunąć, albo zarejestrować. http://www.editplus.com/

PERL Code Editor ma status freeware. Jego rola ogranicza się wyłącznie do edytora plików Perla - prawie zerowe możliwości konfiguracji. http://www.perlvision.com/pce/

Context to darmowy, obsługujący wiele języków edytor. Obsługuje makra, eksportuje do formatu rtf, oraz html, bardzo szybki. http://www.fixedsys.com/context



- **CodeWhiz**. (wersja trial) http://www.incatec.com/
- **Visual Perl Editor**. http://www.xarka.com/vpe.html
- **Ultra Edit**. http://www.idmcomp.com
- **DzSoft Perl Editor**. http://www.dzsoft.com/dzperl.htm
- **MED**. http://www.utopia-planitia.de/
- **Zabobon Edytor**. Podświetla składnię Perla, php, javy, pascala. http://www.zabobonedytor.prv.pl/

4. Budowa i uruchamianie skryptu.

Perl jest, podobnie jak C, językiem bez ścisłego formatu. Nie występuje w nim struktura linii używana w Fortranie. W zależności od tego gdzie znajduje się interpreter. **Każda komenda musi kończyć się średnikiem** (;). Tekst rozpoczynający się od znaku **hash** (#) jest traktowany jako **komentarz**. Bloki kodu, które obejmują ciało warunkowe lub pętle są ograniczane nawiasami podobnie jak w C ({...}). Po napisaniu odpowiedniego kodu, zapisujemy go pod wybraną nazwą, nadając **rozszerzenie ".pl"**. Najprostszy program:

```
#!/usr/bin/perl -w

print "Hello, World!\n";
```

Pierwszą linią każdego programu jest "**shebang**". Linijka ta informuje nas, pod jaką lokalizacją znaleźć można interpreter Perla. W środowisku UNIX może to być usr/bin/perl. W Windows 98 - c:/perl/bin/perl.exe. Opcja -w oznacza włączenie ostrzeżeń. Funkcja "**print**" powoduje wysłanie na ekran łańcucha "Hello, World!". Po niej występuje średnik, który musi pojawić się na końcu każdej instrukcji. Aby uruchomić skrypt należy w wierszu poleceń wpisać "perl" i nazwę skrytpu:

```
C:\perl\> perl hello.pl

# lub w systemach UNIX:

znak_zachety$ moj_program.pl
```

Efektem będzie wyświetlenie się na ekranie napisu "Hello, World!" Jeśli skrypt nie działa:
● źle ustawiono ścieżkę do interpretera Perla,
● podano złą ścieżkę do pliku zawierającego skrypt,
● niepoprawnie zainstalowano interpreter Perla,
● (w systemach UNIX) nie ustawiono prawa dostępu do pliku (aby nadać skryptowi prawo wykonywalności należy użyć polecenia chmod a+x mój_program.pl).

Opcje wywołania programu.

Wymienionych poniżej opcji można używać podczas wywołania programu z linii poleceń, lub (systemy UNIX) w linii, gdzie podajemy ścieżkę do interpretatora:

-**h** # wyświetla pomoc dotyczącą parametrów uruchamiania programu,

-**c** # kompilacja programu w celu wykrycia błędów, bez jego wykonania,

-**d[:debugger]** # uruchamia skrypt pod danym debuggerem,

-**T** # uniemożliwia wykonanie operacji związanych z systemem plików lub systemem operacyjnym, co powoduje # zwiększenie bezpieczeństwa programów CGI,

-**v** # wyświetla aktualnie używaną wersję interpteratora Perl,

-**w** # powoduje wyświetlenie dodatkowych ostrzeżeń przy debugowaniu programu. Może powodować problemy #w skryptach wykorzystywujących interfejs CGI (przez serwer WWW).

5. Typy danych

- zmienne
- wektory zmiennych
- skojarzone wektory zmiennych

Zmienne

Zmienna jest podstawowym typem w Perlu. Zmienna może mieć wartość całkowitą, zmiennoprzecinkową, lub znakową. Perl ustala typ zmiennej z kontekstu. Zmienne zawsze posiadają prefiks $. Np.:

```
$str = "Hello world!";
```

Zmienne w Perlu nie musza być deklarowane. Są alokowane dynamicznie. Domyślną wartością zmiennej jest, w zależności od kontekstu, 0 lub ciąg pusty. Zmienne znakowe użyte w kontekście liczbowym są interpolowane do ich wartości. Np.:

```
$x = 8;          # zmienna całkowita
$y = "15";       # ciąg znaków
$z = $x+$y;      # zmienna z jest typu całkowitego i równa 23
```

Konwersja może zachodzić również w drugą stronę np.:

```
$answer = 42;
print "the answer is $answer";     #na ekranie:"the answer is 42”
```

Wektory zmiennych

Perl zawiera wektory (lub listy) zmiennych. **Bieżącą wartość** udostępnia prefiks **at** (**@**). Można również przypisać elementy wektora przez nazwę. Oto kilka sposobów:

```
@numbers = (3,1,4,1,5,9);
@letters = ("this","is","a","test");
($word,$another_word) = ("one","two");
```

Można się odwoływać do poszczególnych elementów, przy czym **pierwszy element ma indeks 0**:

```
$cos[2] = 2.718281828;
$message[12] = "wiadomosc\n";
```

Ciąg **$#** używa się do znalezienia **ostatniego ważnego indeksu** wektora, a nie jego rozmiaru. Zaś zmienna **$[** oznacza numer pierwszego elementu w każdym wektorze. Domyślną jego wartością jest 0. Przykład programu informującego o ilości elementów w wektorze @a :

```
$n = $#a - $[ + 1;
print "ilosc elementow w wektorze: $n \n";
```

Wektory są rozwijane dynamicznie. Wystarczy przypisać wartość zmiennej $#, a dany wektor zostanie alokowany.

```
$#months = 11;          # elementy 0 – 11.
```

Skojarzone wektory zmiennych

Jest to najbardziej użyteczna cecha Perla. Można dzięki niej tworzyć tablice użytkowników według "login name" i tablice nazw plików. Prefiksem dla tego rodzaju zmiennych jest znak procenta (%). Kluczem dla tych wektorów są zmienne znakowe (numeryczne ulegają konwersji do znakowych). Np.

```
%quota = ("root",100000,
          "pat",256,
          "fiona",4000);
```

odwoływanie się do elementów ma następującą postać:

```
$quota{dave} = 3000; # dave - klucz, 3000 – wartość
```

Należy zwrócić uwagę na wyłączanie się nazw. W Perlu zmienne, wektory, wektory skojarzone, funkcje i pakiety mogą mieć ta sama nazwę i nie będzie to rodzić konfliktów.

6. Operatory i znaki porównania

Zestaw operatorów i znaków porównania w Perlu jest bardzo zbliżony do C. Wszystkie operacje arytmetyczne z C są przeniesione do Perla. Poniższe są ważne tylko dla Perla:

** Operator wykładniczy	**= Przypisanie wykładnicze	x Operator powtórzenia
() Zerowa lista wektora	. Połączenie dwóch ciągów znakowych	.. Operator zakresu
.= Przypisanie połączenia	eq Równość ciągów znakowych (odpowiednik ==)	

7. Zmienne predefiniowane

Perl ma pewien zbiór zmiennych predefiniowanych. O wszystkich można przeczytać w manualu. Oto niektóre z nich:

$_ Domyślny argument funkcji i struktur.

$liczba Kolejne dopasowane podciągi z wyrażenia regularnego.

$. Numer linii w ostatnio czytanym pliku.

@ARGV Lista argumentów skryptu. $ARGV[0] jest pierwszym argumentem, nie jak w C, nazwą programu. Nazwa kryje się pod $0

%ENV Wykaz zmiennych środowiska

8. Kontrola przepływu

Perl ma wszystkie struktury kontroli przepływu, które są w normalnym języku procedur, jak również kilka dodatkowych.

If-Then-Else

W Perlu jest podobnie jak w C. Używa się tu tych samych operatorów: "&&" to "i", "||" - lub, "!" zaś to negacja. Jedyną różnicą jest brak jednolinijkowego wykonania warunku. To znaczy zamiast:

```
if ($error < 0)
fprintf(stderr,"Blad o kodzie %d\n",error);
```

należy w Perlu zapisać:

```
if ($error < 0)
{ print STDERR "Błąd o kodzie $error\n"; }
```

Natomiast odpowiednikiem dwóch słów w C else if jest w Perlu słowo elsif, poza tym istnieje słowo przeciwstawne unless. Na przykład:

```
unless ($#ARGV > 0) # czy są jakieś argumenty
{ print "Błąd, brak argumentow\n"; exit 1; }
```

Idea wartości logicznych jest identyczna do C. **Zero to fałsz, nie zero prawda**. Pusty ciąg znaków - fałsz, o długości 1 lub więcej - prawda. Wektory i ich skojarzone odpowiedniki o ilości elementów 0 - fałsz, więcej - prawda. Nieistniejące zmienne, mają z definicji wartość zero, czyli fałszu.

Konstrukcja while

W Perlu while ma różnorakie zastosowania. Na przykład warunkiem może być wywołanie funkcji (wyświetlanie tekstu wpisanego z klawiatury):

```
while (<STDIN>)
{
        print "wpisałeś ",$_;
}
```

Powyższy kod napisany przez początkującego, powinien wyglądać następująco:

```
while ($_ = <STDIN>)
{
        print "wpisałeś ",$_;
}
```

Poniżej przykład na wartość logiczną wektora. Pętla while będzie wykonywana dopóki @users będzie miał choć jeden element. Funkcja **shift zwraca pierwszy element** z @users i wyrzuca go z wektora @users.

```
@users = ("nigel","david","derek","viv");
while (@users)
{
        $user = shift @users;
        print "$user ma konto\n";
 }
```

Konstrukcja for i foreach

Te konstrukcje w Perlu są równoważne. Zasadniczo są dwa sposoby użycia for/foreach. Pierwszy, podobny do C:

```
@disks = ("/data1","/data2","/usr","/home");
for ($i=0; $i <= $#disks; ++$i)
{ print $disks[$i],"\n"; }
```

Dla przykładu odpowiednik powyższego kodu z jednoargumentowym wywołaniem foreach:

```
@disks = ("/data1","/data2","/usr","/home");
foreach(@disks)
{ print $_,"\n"; }
```

Należy zauważyć jest to bardziej zwięzłe i nie niszczy zawartości @disks, poza tym $_ jest wskaźnikiem, a nie kopią, zatem jeśli w kodzie zmienimy $_ to zmieni się także wektor.

GOTO

Konstrukcja goto etykieta - skieruje bieg programu do określonej etykiety. Podobnie jak w innych językach nie zaleca się stosować goto.

9. Funkcje wewnętrzne i systemowe

Perl ma bogaty zestaw funkcji wewnętrznych i C-podobnych. Dokładniejszy opis zawiera manual. Zostaną tu przedstawione najczęściej używane. Domyślnym argumentem tych funkcji jest $_. Należy zwrócić uwagę, że w większości nawiasy są opcjonalne.

Funkcje wewnętrzne

chop expr - zwraca ostatni znak w ciągu i wyrzuca go z tego ciągu.
Przydatne do odcięcia znaku nowej linii, po wczytaniu
zmiennej z klawiatury.
defined expr - sprawdza czy zmienna istnieje
die expr - wypisuje ciąg podany jako argument i kończy działanie skryptu
each array - zwraca parę klucz-wartość w skojarzonym wektorze
pop array - skraca wektor o ostatni element
shift - zwraca i wyrzuca pierwszy element wektora, skracając długość o 1.
Shift i unshift działają od lewej strony, zaś push i pop od prawej.

Funkcje typu UNIXowego

chmod - zmień bity dostępu do pliku
mkdir - twórz katalog
unlink - skasuj plik
rename - zmień nazwę pliku
rmdir - skasuj katalog

10. Operacje na plikach

W Perlu można korzystać z operacji nie tylko na plikach tekstowych.

Operacje tekstowe

Perl może czytać i zapisywać pliki tekstowe przez deskryptory. Są one zwyczajowo pisane dużymi literami. Pliki otwiera się komendą open. Ma ona dwa argumenty: deskryptor pliku i jego nazwę. Kolejne linie są czytane przez użycie deskryptora w ostrych nawiasach (<...>):

```
open(F,"data.txt");
while($line = <F>)
{
        # zrob cos z linia
}
close F;
```

Argument reprezentujący nazwę pliku może posiadać jeden z kilku prefiksów. Jeśli jest to <, plik jest otwarty do czytania (domyślny prefiks), > do pisania - jeśli już istnieje, jest nadpisywany, zaś >> do dopisywania. Przykłady:

```
open(PASSWD,"</etc/passwd");
while ($p = <PASSWD>)
{
        chop $p;
        @fields= split(/:/,$p);
        print "Uzytkownik $fields[0] ma katalog $fields[6]\n";
}
close PASSWD;

open(LOG,">>user.log");
print LOG "$user zalogowal sie\n";  # dopisz do logu

$response = <STDIN>;   # wczytaj linie
```

Są 3 predefiniowane deskryptory, o oczywistym znaczeniu: STDIN, STDOUT, STDERR. Próba otwarcia ww. może spowodować dziwne efekty. Inaczej zachowuje się Perl w przypadku wczytywania pliku do wektora. Wczytywany jest cały plik, przy czym każda linijka to jeden element.

```
$file = "some.file";
open(F,$file);
@lines = <F>;
close F;
```

Jest to użyteczna cecha, powinna jednak być używana z wielką uwagą. Wczytanie całego pliku o nieznanej długości do pamięci może być ryzykowne.

Pipe' y

Perl może używać funkcji open do uruchomienia komend shella, czytania i pisania do nich. Jeśli nazwa pliku zaczyna się od znaku pipe (|), nazwa ta traktowana jest jako komenda. Jest ona wykonywana, zaś dane wejściowe można dostarczać przez print. Jeśli zaś nazwa pliku kończy się |, efekt wyjściowy można wykorzystywać przy pomocy składni <...> :

```
open(MAIL,"| mail root");       # wyslij poczte do root'a
print MAIL "uzytkownik \"Jas\" jest glupi\n";
close MAIL;                     # poczta jest wysylana

open(WHO,"who |");              # kto jest w systemie
while ($who = <WHO>)
{
        chop $who;
        ($user,$tty,$junk) = split(/\s+/,$who,3);
        print "$user jest na terminalu $tty\n";
}
close(WHO);
```

Użycie komendy print

Użycie funkcji print jest bardzo różnorodne. Ogólnie print pobiera serie ciągów znakowych, przedzielonych przecinkami, interpoluje odpowiednio wartości, po czym wypisuje efekt na ekranie. Często wraz z printem używa się operatora połączenia ciągów (.). Poniższe przykłady dają ten sam efekt na ekranie:

```
print "Idz do 11.\n";
$level = 11;
print "Idz do ",$level,".\n";
print "Idz do $level.\n";
printf "Idz do %d.\n",$level;
print "Idz " . "do ". $level.".\n";
print join(' ',("Idz","do","$level.\n"));
```

Jak widać również funkcja printf z C jest dostępna.

11. Wektory w Perlu

W C można wywoływać funkcje zagnieżdżone np.: chdir(getenv("HOME")). Podobna własność posiada Perl. Nie tworzy on przy tym wektorów tymczasowych. Oto kilka przykładów. W pierwszym widać użycie funkcji sort:

```
@names = ("bill","hillary","chelsea","socks");
@sorted = sort @names;
foreach $name (@sorted)
{ print $name,"\n"; }
```

W rzeczywistości można to napisać w następujący sposób:

```
foreach $name (sort @names)
{ print $name,"\n"; }
```

a oto jeszcze lepszy przykład:

```
$name = (getpwuid($<))[6];
print "nazywam sie ",$name,"\n";
```

Funkcja getpwuid zwraca wektor. My zaś chcemy "prawdziwe imię i nazwisko" z linijki z passwd, wiec bierzemy tylko pole numer 6 ([6]) i podstawiamy je pod $name.

12. Podprogramy i pakiety

Perl oferuje możliwość programowania modułowego i bibliotek.

Podprogramy

Perl może zawierać funkcje, posiadające parametry i zwracające wartości. Poniżej przedstawiono szkielet:

```
sub sub1
{
        local($param1,$param2) = @_;
        # zrob costam
        $value;
}
```

Zaś wywołanie może mieć postać:

```
$return_val = do sub1("to jest","test");
```

Słowo kluczowe do można zastąpić znakiem &. I jest to metoda polecana:

```
$return_val = &sub1("to jest","test");
```

Pisząc funkcje należy pamiętać, że parametry umieszczane są jako wektor **@_**. Ponieważ wszystkie zmienne domyślnie są globalne, użyto tu funkcji local() w celu skopiowania wartości pod zmienne lokalne. Perl posiada konstrukcje return, która może być użyta eksplicite do zwrócenia wartości. Jest to zwykle niepotrzebne, gdyż **zwracaną wartością jest wynik wykonania ostatniej linii procedury**. Zatem jeśli chcemy, żeby funkcja zwracała 0, jako ostatnia linie piszemy po prostu **0;**

Pakiety

Perl ma bibliotekę funkcji, które mogą być używane w skryptach. Odpowiednikiem #include z C jest tu require. Na przykład dołączenie funkcji getopt:

```
require 'getopts.pl';
&Getopts('vhi:');
if ($opt_v) { print "Wlaczony jest tryb sledzenia\n"; }
```

Zamiast require można użyć use. Wywołania funkcji nie muszą być wtedy poprzedzone nazwą pakietu:

```
use Cwd;
   $here = getcwd();
```

zamiast:

```
require Cwd;
$here = Cwd::getcwd();
```

Do standardowej dystrybucji Perla dołączony jest pakiet CGI, dzięki któremu można obsługiwać formularze.

13. Programowanie obiektowe

W Perlu programowanie obiektowe jest w znacznym stopniu nieformalne — prawie wszystko trzeba wykonać samodzielnie.

Klasa to pakiet (tworzy odrębną od reszty kodu przestrzeń nazw), który może udostępniać metody.

Metoda to procedura wbudowana w klasę lub obiekt. Metoda jako pierwszy parametr otrzymuje wskaźnik na obiekt lub nazwę klasy.
Obiekt to element wskazywany wskaźnikiem. Obiekty tworzy się na podstawie klas.

Dziedziczenie to proces wywodzenia jednej klasy, nazywanej klasą pochodną, z innej klasy, bazowej. W klasie pochodnej są dostępne metody klasy bazowej.

bless – funkcja do ustanowienia połączenia między wskaźnikiem i klasą.

my - ogranicza leksykalnie (uwaga a nie lokalnie!) zasięg zmiennej do procedury. Zmienna jest związana z otaczającym ją blokiem, instrukcją warunkową itp. Zmienne ograniczone leksykalnie — nie są widoczne także w procedurach wywoływanych wewnątrz zasięgu zmiennej.

sub – przed definicją funkcji

Oto **przykład klasy** Classl, obsługującej konstruktor o nazwie new. W tym konstruktorze tworzymy wskaźnik na anonimową asocjację zawierającą dane obiektu (danych nie trzeba przechowywać akurat w asocjacjach, można

użyć tablicy lub nawet skalara). Następnie asocjacja jest wiązana z bieżącą klasą, po czym w końcu zwraca wskaźnik:

```
package Classl;
sub new
{
my $self = {};
bless ($self);
return $self;
}
return 1;
```

Obiekty

W Perlu obiekt nazywamy instancją klasy. W celu utworzenia obiektu wywołuje się konstruktor klasy — zwykle o nazwie new. Oto przykład utworzenia obiektu przygotowanej wcześniej klasy Classl:

```
use Classl;
my $objectl = Classl->new();
```

Metody

Kiedy mamy już obiekt z metodami, można tych metod używać tak, jak to pokazano poniżej — metoda calculate wykonuje na wartościach $operand1 i $operand2 pewne obliczenia i wynik zapisuje w $result:

```
$result = $objectl->calculate($operand1, $operand2);
```

Dziedziczenie
Pozwala wyprowadzić z istniejącej klasy nową klasę, która odziedziczy wszystkie metody i dane klasy pierwotnej. Do klasy pochodnej można dodawać nowe metody, które rozszerzą jej możliwości. W poniższym przykładzie używamy klasy Classl jako klasy bazowej dla Class2. Warto zwrócić uwagę na metodę get-text, dostępną później w klasie Class2:

```
package Classl;
sub new
{
my $self – {};
bless($self) ;
return $self;
}
sub gettext {return "Hello!\n";}
return 1;
```

```
package Class2;
use Classl;
@ISA = qw(Classl);
sub new
{
my $self = Classl->new;
bless($self);
return $self;
}
return 1;
```

Oto klasa Class2 dziedzicząca po klasie Classl. Dziedziczenie polega na użyciu instrukcji **use Classl** i umieszczenie Classl w tablicy @ISA (tablicę tę interpretuje się tak, że Class2 znajduje się w relacji, „jest przykładem" z Classl):

Destruktory zawierają kod uruchamiany przy usuwaniu obiektów (na przykład przy wyjściu poza ich zakres, czy przy kończeniu pracy interpretera). W przeciwieństwie do konstruktorów, destruktory mają w Perlu jednoznacznie określoną nazwę: destroy. Tak jak inne funkcje wywoływane niejawnie, tak i nazwę destroy zapisuje się wielkimi literami. destroy jest wywoływana przez Perla, a więc nie można jej wywołać samemu jawnie.

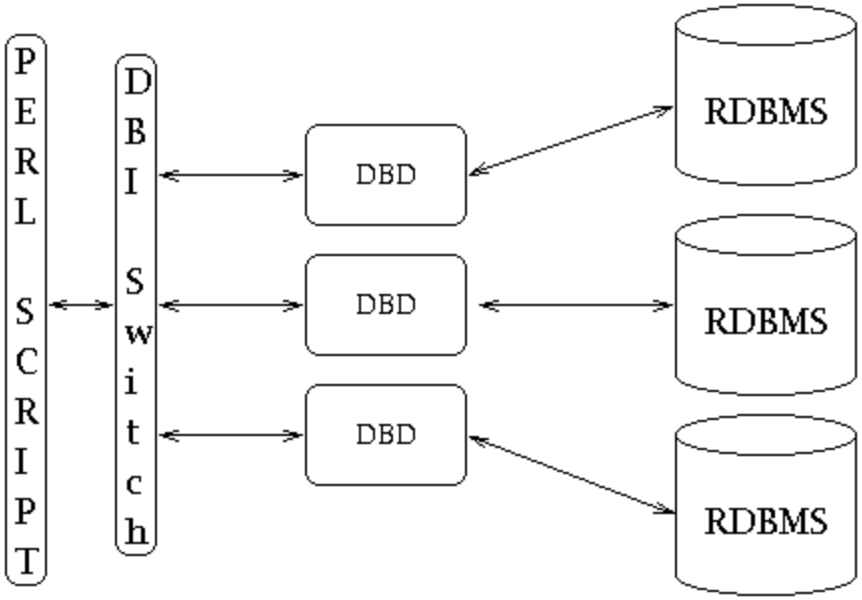
```
package Classl;
sub new
{
my $self = {};
bless($self);
return $self;
}
sub DESTROY{print "Obiekt jest usuwany!"}
return 1;
```

Kiedy usuwany będzie obiekt klasy Classl, zostanie wyświetlony komunikat — w tym przypadku usuwanie obiektu jest spowodowane zamykaniem programu:

```
use Classl;
my $objectl = Classl->new();
exit;
```

14. Współpraca z bazami danych.

Jeśli chodzi o współpracę z bazami danych w Perl 5 zastosowano rozwiązanie zwane DBI (Database Interface) autorstwa Tim'a Bunce'a. DBI to najważniejszy interfejs baz danych dla języka Perl. Jest niezależny od rodzaju bazy, a więc udostępnia te same metody programowania bez względu na to, czy pracujemy z bazą Oracle, Sybase, Ingres, Informix czy dowolną inną. DBI wprowadza abstrakcyjną warstwę miedzy kodem Perla a BD. Pozwala to w łatwy sposób przełączać się miedzy różnymi implementacjami BD. Poniższy diagram przedstawia działanie DBI:



Program napisany w Perlu komunikuje się z DBI a ten z kolei ze sterownikiem bazy danych (Database Driver - DBD) odpowiadającym wybranej bazie danych.

Tworzymy string identyfikujący BD, z która chcemy się połączyć. Zwany jest **on data source name**, albo **DSN**. Dla przykładu chcemy się połączyć z BD phonebill zaimplementowaną w MySQL. DSN składa się z **dbi : rodzaj sterownika : nazwa naszej bazy danych.** Łączymy BD z DBI:

```
use DBI;
my $dbh = DBI->connect("dbi:mysql:phonebill", $user, $password);
```

DBI zwrócił nam uchwyt do naszej bazy danych (przechowywany w zmiennej $dbh) przydatny do wykonania zapytań.

Wykonanie zapytania. Przygotowujemy kod SQL, następnie wykonujemy zapytanie, wreszcie pobieramy wyniki.

```
my $sth = $dbh->prepare(<<SQL);
   select recipient, calldate, calltime, $duration
   from call
   where duration > 60
   order by duration desc
SQL

$sth->execute;

my %calls;
while (my @row = $sth->fetchrow_array()) {
   my ($recipient, $calldate, $calltime, $duration) = @row;
   $calls{$recipient} += $duration;
   print "Called $recipient on $calldate\n";
}
```

Przyklad insertu do BD:

```
   my $sth = $dbh->prepare(<<SQL);
INSERT INTO call (recipient, calldate, calltime, duration)
VALUES (?, ?, ?, ?)
SQL
```

```
while (my $data = <FILE>) {
    my ($recipient, $date, $time, $duration) = split /:/, $data;
    $dbh->execute($recipient, $date, $time, $duration);
}
```

DBI wprowadza pewne ułatwienia, np. gdy nie chcemy otrzymać wyników jak np. w poleceniu DELETE.

```
# Ignore short calls.
$dbh->do("delete from calls where duration < 5");
```

Oprócz DBI są tez inne abstrakcyjne warstwy Perla nad SQLem np. **Class::DBI** (Tony'ego Bowden'a), **the DBIx::RecordSet, DBIx::SearchBuilder** i wiele innych.

15. Nowości w PERL 6.

Niektóre zasadnicze zmiany wprowadzone w Perlu 6:
- gruntowna przebudowa systemu wyrażeń regularnych

- Operator wywołania metody '.'

```
$obj->metoda();        # Perl 5
$obj.metoda();         # Perl 6
```

- Operator łączenia łańcuchów '~'

```
$b = $a . '_b';        # Perl 5
$b = $a ~ '_b';        # Perl 6
```

- Zniesione nawiasy

```
if ($zmienna){...}      #Perl 5
if $zmienna {...}       #Perl 6
```

- Podobnie w warunkach pętli itp.

```
while $zmienna {...}
```

16. Praca dla programistów PERLA.

1) Miejsce pracy: Gdańsk

Oczekiwane kwalifikacje:
- biegła znajomość projektowania i tworzenia skryptów parsujących dane
- minimum 2 letnie doświadczenie w programowaniu w PERL
- bardzo dobra znajomość zagadnień relacyjnych baz danych
- doświadczenie w pracy na platformie Linux
- znajomość PHP i CVS będzie dodatkowym atutem

Do zadań zatrudnionej osoby będzie należeć:
- tworzenie skryptów do przetwarzania, importu i eksportu danych do bazy
- rozwój istniejącej aplikacji do przetwarzania danych

http://www.gratka.pl/praca/oferta.phtml?id=86143

2) Miejsce pracy: Warszawa

Wymagania:
- doskonała znajomość perla,
- znajomość baz danych MySQL,
- umiejętność czytania dokumentacji w jez. angielskim,
- systematyczność i dobra organizacja pracy.

Dodatkowe atuty:
- znajomość systemów Linux,
- znajomość HTML i JavaScript.

Oferujemy:
- atrakcyjne wynagrodzenie, adekwatne do posiadanego doświadczenia oraz umiejętności,
- możliwość rozwoju oraz podnoszenia kwalifikacji zawodowych,
- prywatna opiekę zdrowotna,
- zatrudnienie w pełnym wymiarze godzin, umowa o prace (w okresie próbnym - umowa zlecenie). Osobom szukającym pracy zdalnej dziękujemy.

http://www.grupy.egospodarka.pl/Warszawa-programista-perl-um-o-prace,t,177315,8.html

3) Miejsce pracy: Centrala LUKAS Banku we Wroclawiu

WYMAGANIA:
- biegła znajomość Perla
- doświadczenie w zakresie tworzenia aplikacji w środowiskach Webowych
- znajomość technologii XML, XML Schema, XSLT

- doświadczenie w analizie i projektowaniu obiektowym
- umiejętność pracy w zespole
- znajomość języka angielskiego
- mile widziana znajomość zagadnień integracji aplikacji w oparciu o najnowsze standardy
- mile widziana umiejętność programowania w języku Java
- mile widziana znajomość technologii .NET (C#, ASPX)
- znajomość zagadnień bankowych będzie dodatkowym atutem

ZAKRES OBOWIAZKOW:
- projektowanie i implementacja nowych modułów aplikacji bankowych
- integracja aplikacji pracujących w banku
- rozwój i utrzymanie narzędzi programistycznych do tworzenia aplikacji
- rozwój i utrzymanie istniejących aplikacji
- przygotowanie dokumentacji technicznej i użytkowej
- raportowanie stanu i przebiegu zleconej pracy

http://www.damprace.net/ogloszenia,praca,projektant,programista,perl,w,departamencie,informatyki,w,centrali,lukas,banku,we,wroclawiu,14206.aspx

17. Bibliografia

- najlepsze źródło wiedzy to manual (polecenie systemowe man w Unixie)
- książki:
  - Larry Wall & Randal Schwartz "Programming Perl"
  - Randal Schwartz "Learning Perl"
  - Holzner_Steven_-_Perl._Czarna_ksiega

- grupy dyskusyjne
  - comp.lang.perl

- pl.comp.lang.perl
- comp.lang.perl.announce
- com.lang.perl.mixc
- com.lang.perl.modules
- comp.lang.perl.tk - informacje o łączeniu Perla z pakietem Tk języka Tcl. Pakiet Tk obsługuje szereg elementów graficznych, jak choćby przyciski czy menu, dzięki czemu elementy te można automatycznie używać także w Perlu.
- comp.infosystemx.www.authoring.cgi - zastosowania Perla w CGI.

- strony internetowe
  - www.perl.com
  - www.perl.com/perl/faq
  - www.cpan.org - moduły i rozszerzenia Perla (np. obsługa obrazków lub moduły internetowe do obsługi interfejsów do baz danych)
  - www.media.mit.edu/thej3erljournal - kwartalnik poświęcony Perlowi

18. Źródła opracowania.

Holzner_Steven_-_Perl._Czarna_ksiega_
http://www.kt.agh.edu.pl/other/perl/
http://www.cgi.csd.pl/index-v.php?n=edytory
http://www.webhelp.pl/artykuly/
http://history.perl.org/
http://www.perl.com/pub/a/2003/10/23/databases.html
Bartłomiej Jakubski- „Parrot i Perl 6 –Wprowadzenie” Jesień Linuksowa 2004