Pętle w PERLU – postać ogólna

```
ETYK:  while(WYRAŻENIE) BLOK;
ETYK:  while(WYRAŻENIE) BLOK
       continue BLOK;

ETYK:  for(WYR1;WYR2;WYR3) BLOK;
ETYK:  for(WYR1;WYR2;WYR3) BLOK
       continue BLOK;

ETYK:  foreach ZM(LISTA) BLOK;
ETYK:  foreach ZM(LIST) BLOK
       continue BLOK;
```

Blok continue

Blok *continue* (jeśli istnieje) jest wykonywany **zawsze** po zakończeniu **każdego** obiegu pętli, nawet jeśli wykonanie pętli zostało przerwane w sposób gwałtowny.

```
for(wyr1; wyr2; wyr3) { … }

wyr1;
while(wyr2) { … } continue { wyr3; }
```

Instrukcja **next** z etykietą

```
PETLA:
   foreach $ten1(@lista1) {
        foreach $ten2(@lista2) {
             $f = ($ten1 == $ten2);
             next PETLA if $f;
        }
   }
```

Instrukcja **next** bez etykiety

```
foreach $ten1(@lista1) {
     $f = ($ten == 100);
     next if $f;
     $ten = 0;
} continue {
     $licz++ if $f;
}
```

Instrukcja **redo**

```
PETLA: while($linia = getline()) {
       if(kontynuacja($linia))
         {
         $linia .= <STDIN>;
         redo;
         }
       wykonaj($linia);
     }
```

Gołe bloki (ang. *bare blocks*)

```
BLOK: {
      :
      last if $f;
      :
      redo unless $f;
      :
}
```

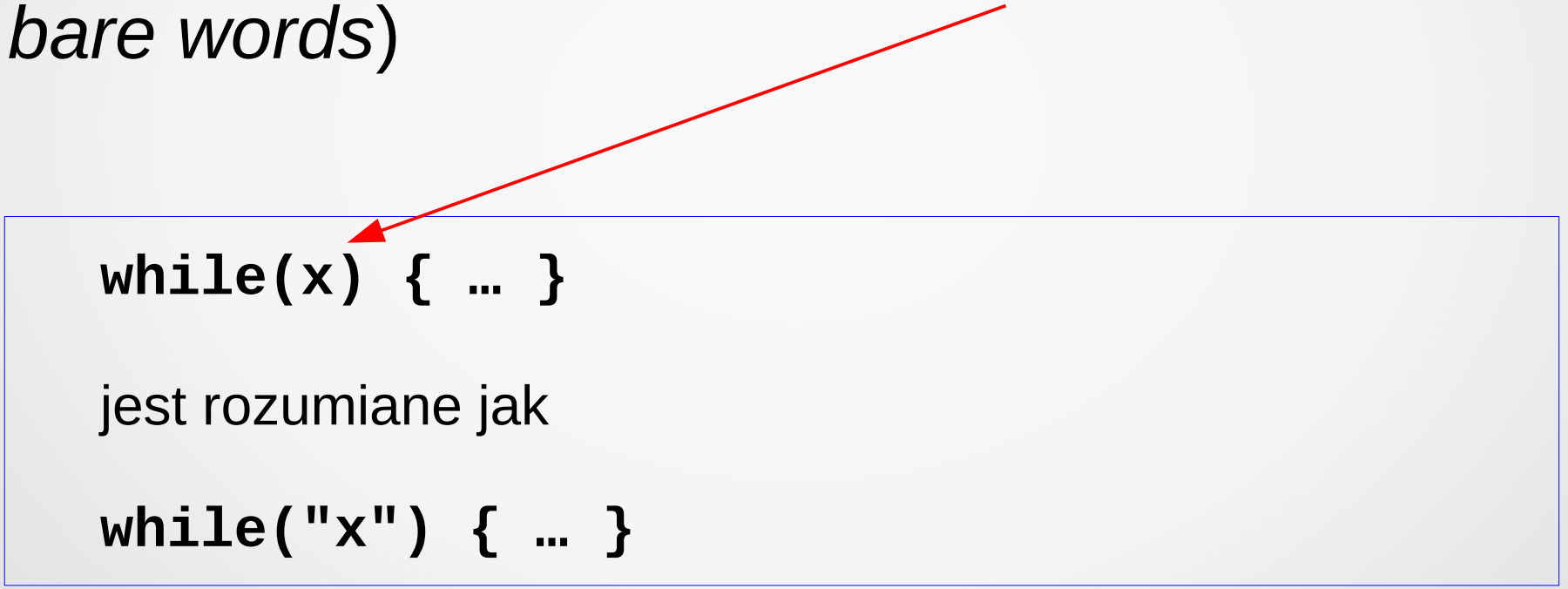
Instrukcja **switch**?

W PERLU nie istnieje instrukcja *switch*, ale można próbować ją „podrabiać” z użyciem gołego bloku.

```
SWITCH: {
   if($i>0) { $dodatnie++;last SWITCH; }
   if($i<0) { $ujemna++; last SWITCH; }
   $zera++;
}
```

Dygresja

Gołe bloki nie są jedynymi gołymi elementami PERLA – istnieją również „gołe słowa” (ang. *bare words*)



```
while(x) { … }
```

jest rozumiane jak

```
while("x") { … }
```

Funkcje w PERLU

wstrząsający fakt #1:

funkcje w PERLU nie mają nazwanych parametrów, co jest ich WIELKĄ zaletą

Funkcje w PERLU

wstrząsający fakt #2:

w PERLU wszystkie zmienne są globalne, no chyba, że zostały zadeklarowane jako nie będące globalnymi

Funkcje w PERLU

wstrząsający fakt #3:

w PERLU funkcja może się dowiedzieć, czy została wywołana w kontekście skalarnym, tablicowym czy pustym oraz z iloma argumentami została wywołana; funkcja może się też dowiedzieć, jak nazywa się funkcja, która ją wywołała oraz jak nazywa się funkcja, która wywołała funkcję, która ją wywołała oraz …

Funkcje w PERLU

wstrząsający fakt #4:

w PERLU przesłanie do funkcji więcej niż jednej tablicy (hasza) sprawia, że tracą one swoją tożsamość i stają się JEDNĄ tablicą

Pierwsza funkcja w PERLU

```
sub funkcja {
   print "kto tam?\n";
   return 1;
}
```

Instrukcja return nie jest niezbędna.
Jeśli jej nie będzie, funkcja zwróci jako wynik wartość tego wyrażenia, które zostało obliczone jako **ostatnie**

Prototyp

```
sub funkcja($$) {
   print "kto tam?\n";
   return 1;
}
```

Prototyp służy tylko i wyłącznie do sprawdzenia poprawności wywołania; nie jest w żaden sposób używany do odzyskiwania wartości parametrów

Prototyp

```
sub funkcja($;$) {
   print "kto tam?\n";
   return 1;
}
```

Prototyp może również specyfikować parametry obowiązkowe (przed średnikiem) i opcjonalne (po średniku)

Przekazywanie parametrów

Przedstawiamy tablicę

@_

Przekazywanie parametrów

```
sub delta($$$) {
   return $_[1]**2 - 4 * $_[0] * $_[2];
}

$d = delta (1,-2,3);
```

Przekazywanie parametrów

```
#uwaga! @_ jest tablicą synonimów parametrów

sub SQR {
   $_[0] **= 2;
   return $_[0];
}

$a=2;
print SQR($a),SQR(2);
```

Błąd wykonania!

Przekazywanie parametrów

```
#uwaga na tablice!

sub dump {
   foreach $elem (@_) {
     print "$elem ";
}

@a = (1,2);
@b = (3,4);

dump(@a,@b);
```

Przekazywanie parametrów

```
#kopiowanie @_ do zmiennych

sub fun {
   $a = shift @_; # albo krócej: shift
   $b = shift;
   return $a**2 + $b**2;
}
```

Lokalność w sensie PERLA

```
# lokalność statyczna

sub fun {
   my $a;
   my ($b,$c);
   my ($p1,$p2) = @_;
}
```

Lokalność w sensie PERLA

```
# lokalność dynamiczna

sub fun {
   local $a;
   local ($b,$c);
   local ($p1,$p2) = @_;
}
```

Lokalność w sensie PERLA

```
# lokalność pakietowa

sub fun1 {
   our $a;
   :
}

sub fun2 {
   our $a;
   :
}
```

Lokalność statyczna vs. dynamiczna

```
sub prn { print "$ZMIENNA\n"; }
sub f1 { my $ZMIENNA = 111; prn(); }
sub f2 { local $ZMIENNA = 222; prn();}
# zaczynamy
$ZMIENNA = 0; prn();
f1(); prn();
f2(); prn();
```

Symbole specjalne

__FILE__

__LINE__

Funkcja **caller**

```
($package,
 $filename,
 $line,
 $subr,
 $has_args,
 $wantarray)  = caller($i);
```

Liczba ramek stosu, o które chcemy się cofnąć.

Uwaga! Opisano tylko część tablicy zwracanej jako wynik!

Funkcja **caller**

```
#! /usr/bin/perl

# call.pl
sub f1 { f2(); }
sub f2 { f3(); }
sub f3 {
        print join(',',(caller(0))[0,1,2,3]) . "\n";
        print join(',',(caller(1))[0,1,2,3]) . "\n";
        print join(',',(caller(2))[0,1,2,3]) . "\n";
        print join(',',(caller(3))[0,1,2,3]) . "\n";
}

f1();
```

Funkcja **caller**

```
main,./call.pl,4,main::f3
main,./call.pl,3,main::f2
main,./call.pl,12,main::f1
```

Zwracanie wartości

```
return;
return (wantarray ? () : undef );
sub MAX_HANDLES { 20; }
```

Wyrażenia regularne w PERLU

Operatory dopasowania wzorca:

```
m/wzorzec/gimosx
/wzorzec/gimosx
s/wzorzec/substytut/egimosx
tr/wzorzec/koder/cds
```

Operator m

`m/wzorzec/gimosx`

jeśli użyto **m**, to zamiast / można użyć dowolnego znaku nie-alfanumerycznego

`m#wzorzec#gimosx`

wyjątek:

`m(wzorzec)gimosx`

Opcje operatora m

opcja	znaczenie
g	dopasuj globalnie
i	ignoruj różnice wielkości liter
m	traktuj napis jako wiele wierszy
o	kompiluj wzorzec tylko raz
s	traktuj napis jako pojedynczy wiersz
x	użyj rozszerzonej składni

Operatory dopasowania

operator dopasowania pozytywnego

```
$skalar =~ /wzorzec/
```

operator dopasowania negatywnego

```
$skalar !~ /wzorzec/
```

$_ - skalar domyślny

obie poniższe formy są równoważne:

```
$_ =~ /wzorzec/

/wzorzec/
```

Operator m - przykłady

```
#sprawdzanie odpowiedzi użytkownika
$answer =~ /^y/i and do_something();

#wyłuskiwanie podnapisu
if($s =~ /Version: *([0-9.]+)/){
   $ver = $1;
}

#unikanie konfliktu wykałaczek
next if $l =~ m#/var/spool/mail#;
```

Operator s

`s/wzorzec/substytut/egimosx`

zamiast / można użyć dowolnego
znaku nie-alfanumerycznego

`s#wzorzec#gimosx#`

wyjątek:

`s(wzorzec)(substytut)`

Opcje operatora s

opcja	znaczenie
e	traktuj substytut jako wyrażenie
g	dopasuj globalnie
i	ignoruj różnice wielkości liter
m	traktuj napis jako wiele wierszy
o	kompiluj wzorzec tylko raz
s	traktuj napis jako pojedynczy wiersz
x	użyj rozszerzonej składni

Operator s - przykłady

```
# zamień lotki na skrzydła, ale nie podlotki
$samolot =~ s/\blotki/skrzydła/g;

# wystrzegaj się wykałaczek
$path =~ s(/usr/bin)(usr/local/bin)
```

Operator s - przykłady

```
# wzorzec i substytut interpolowane
$user = 'guest'; $nuser = 'root';
$login =~ s/Login: $user/Login: $nuser/;

# modyfikowanie w locie
$cola = 'to jest to';
($polococta = $cola) =~ s/to/tamto/;
```

Operator s - przykłady

```
# policz zmiany
$ileSz = ($tekst =~ s/Szanowny/Szanowna/g);

#substytut z wyrażeniem
$org = 'abc123xyz';
$org =~ s/(\d+)/$1 * 2/e;
$org =~ s/(\d+)/$1 x 2/e;
```

Operator s - przykłady

```
#usuń komentarze z programu w języku C
$program =~ s(/\*.*?\*/)()g;

#odwróć dwa pierwsze pola
$linia =~ s/([^ ]*) *([^ ]*)/$2 $1/;
```

Operator **tr**

`tr/wzorzec/koder/cdx`

np.

`tr/A-Z/a-z/;`

można użyć innych ograniczników jak w operatorze s

Opcje operatora tr

opcja	znaczenie
c	szukaj znaków, których nie ma we wzorcu
d	usuń znaki, które nie zostały zamienione
s	eliminuj znaki zamienione i powtórzone

Operator tr - przykłady

```
# zamień na małe litery
$ARGV[1] =~ tr/A-Z/a-z/;

#policz gwiazdki na niebie
$stars = ($sky =~ tr/*/*/);

#policz cyfry
$digits = ($line =~ tr/0-9//);

#koder krótszy od wzorca
$abb =~ tr/abc/ab/;
```

Operator tr - przykłady

```
#eliminuj powtórzenia
$name = 'violletta';
$name =~ tr/a-z//s;

#zamiana w locie
($HOST = $host) =~ tr/a-z/A-Z/
```

Podstawowe operacje we/wy

```
#funkcja open

open UCHWYT, WYRAŻENIE

# UCHWYT – reprezentuje otwarty plik
# WYRAŻENIE – łańcuch zawierający nazwę pliku
# zwraca undef w razie niepowodzenia
# UCHWYT może być skalarem
```

Podstawowe operacje we/wy

```
# otwarcie pliku do odczytu (READ)

open $f,”<file.txt”

# lub

open $f,”file.txt”
```

Podstawowe operacje we/wy

```
# otwarcie pliku do zapisu (WRITE)

open $f,”>file.txt”

# otwarcie pliku do dopisywania (APPEND)

open $f,”>>file.txt”
```

Podstawowe operacje we/wy

```
#funkcja close

close UCHWYT

# UCHWYT – reprezentuje otwarty plik
# zwraca undef w razie niepowodzenia
```

Podstawowe operacje we/wy

```
# uchwyty wstępnie otwarte

STDIN
STDOUT
STDERR
```

Podstawowe operacje we/wy

```
# operator odczytu tekstowego - <>
# zwraca undef w razie niepowodzenia

<UCHWYT>

$scalar = <F>;

@array = <F>;
```

Podstawowe operacje we/wy

```
# sprawdzanie końca pliku

# źle!

while($line=<F>){...}


# dobrze

while(defined($line=<F>)){...}
```

Podstawowe operacje we/wy

```
# dwie przydatne funkcje:

chop $string
# chop usuwa z łańcucha ostatni znak i
# zwraca ten znak jako wynik

chomp $string
# chomp usuwa z łańcucha kończące go znaki
# końca linii i zwraca liczbę usuniętych
# znaków jako wynik
```

Podstawowe operacje we/wy

```
# pisanie do uchwytu

print UCHWYT LISTA;

# uwaga!
# pomiędzy UCHWYT a LISTA nie ma przecinka!
```