Laboratorium 5

Skrypty w PERL-u

23/24.III.2022

Język PERL

W odróżnieniu od narzędzi w rodzaju `grep` czy `sed`, PERL jest językiem wysokiego poziomu umożliwiającym przetwarzanie tekstu w oparciu m.in. o wyrażenia regularne z wykorzystaniem zmiennych, pętli, itd. Składnia jest wzorowana na C, `bash`-u, `awk` i innych językach skryptowych ale jest znacznie mniej wrażliwa na spacje, wcięcia i znaki nowej linii niż w skryptach powłoki.

Programy w PERL-u zapisujemy w plikach o standardowym rozszerzeniu `.pl` oraz uruchamiamy poleceniem `perl program.pl`
Istnieje też możliwość uruchamiania programów analogicznie jak skrypty interpretera - należy w pierwszej linii pliku napisać:

```
#!/usr/bin/perl
```

oraz nadać plikowi programu prawo do wykonywania.

W każdym przypadku warto do polecenia `perl` dodać przełącznik `-w` który powoduje wyświetlanie ostrzeżeń dotyczących kodu programu.

Zmienne w PERL-u

Zmienne są trzech podstawowych typów: skalarne (liczby, napisy), tablice oraz tablice asocjacyjne (haszujące). Typ zmiennej określamy przez pierwszy znak jej nazwy - odpowiednio `$`, `@` lub `%`:

```
$liczba = 12;
$napis = "Hello world";
@tablica = (1, 2, 3, "kot");
%imiona = ("kot" => "Mruczek", "pies" => "Azor");
```

Jeśli zmienna o danej nazwie nie istnieje - zostanie utworzona; odwołanie do zmiennej, której nie przypisano wcześniej żadnej wartości powoduje wyświetlenie ostrzeżenia ale nie błąd (taka zmienna ma wartość `undef` - najczęściej będzie traktowana jak 0 lub pusty napis).

Zmienne `@ARGV` oraz `%ENV` to odpowiednio tablica argumentów przekazanych do programu (skryptu) oraz tablica haszująca zawierająca zmienne interpretera (klucz to nazwa zmiennej) wraz z ich wartością.

Operacje na skalarach i tablicach

Na zmiennych liczbowych operujemy standardowo jak w C: `+ - * / == != > <=`, itd., można też korzystać ze skrótowej notacji `+= --`, operatorów logicznych i bitowych.

Podobnie jak w `bash`-u znaki wewnątrz `'...'` nie są interpretowane, natomiast wewnątrz `"..."` już tak (np. nazwa zmiennej będzie zastąpiona przez wartość; aby tego uniknąć znaki specjalne jak `$` należy poprzedzić `\`); operatory porównania: `eq ne lt ge`, itd., złączanie napisów: `.`

Tablice są indeksowane od 0 oraz automatycznie "powiększane" w przypadku przypisania wartości do indeksu "spoza" tablicy; dostęp do elementu: `$tablica[0]` - uwaga: to jest skalar więc `$` zamiast `@`. Przypisanie `$x = @tablica;` jest poprawne, zmienna przyjmie wartość liczbową równą długości tablicy (można też napisać `$x=$#tablica;`).

Analogicznie działają tablice haszujące - dostęp do elementu o danym kluczu: `$imieKota=$imiona{"kot"};` Przypisania `$len=%imiona;` oraz `@tab=%imiona;` dają długość tablicy haszującej oraz tablicę (zwykłą) zawierającą kolejno klucze i wartości.

Struktura języka i pliki

Wszystkie instrukcje sterujące i pętle wyglądają tak samo jak w C, instrukcje należy kończyć średnikiem a bloki instrukcji otaczać nawiasami { ... }. Dodatkowo istnieje pętla `foreach` umożliwiająca przejście po kolejnych elementach tablicy. Argumenty do funkcji można przekazywać w nawiasach (jak w C) lub bez nawiasów.

Dostęp do plików jest bardzo prosty: pisząc `$linia=<nazwa_pliku>` odczytujemy pierwszą (kolejną) linię pliku a `@calosc=<nazwa_pliku>` daje listę zawierającą wszystkie linie. Można użyć specjalnej nazwy pliku `STDIN` do odczytu standardowego wejścia (istnieją także `STDOUT` i `STDERR`). Aby zapisać dane używamy np. funkcji `print` podając jako pierwszy argument nazwę pliku do zapisu (domyślnie `STDOUT`).

Przykładowo, poniższy program wypisze (na standardowe wyjście) drugą linię każdego z plików podanych jako argumenty wejściowe:

```
foreach $plik (@ARGV) {
  @linia = <$plik>;
  print $linia[1];
}
```

Wyrażenia regularne

Wyrażenie regularne to sposób zapisu "struktury" ciągów znaków, których będziemy np. szukać w tekście. Ma postać napisu zawierającego "zwykłe" znaki, ich grupy i zakresy (np. `[abc]` - dowolna z liter `a,b,c`, `[0-9]` - dowolna cyfra) oraz symbole specjalne - w `PERL`-u, to m.in.:

`\t, \n, .`	tabulator, nowa linia, dowolny znak poza `\n`
`\s, \S`	dowolny znak odstępu, dowolny znak nie będący odstępem
`\w, \W`	dowolna cyfra lub litera, dowolny znak nie będący cyfrą ani literą
`^, $`	początek tekstu, koniec tekstu

Dodatkowo w wyrażeniach można używać znaków odnoszących się do liczby wystąpień wzorca bezpośrednio poprzedzającego: `*` - występuje dowolną liczbę razy (także 0), `+` - występuje przynajmniej raz, `?` - występuje co najwyżej raz, `{n,m}` - występuje od `n` do `m` razy (włącznie). Dopuszczalny jest także znak `|` oznaczający "lub", tzn. wystąpienie wzorca podanego bezpośrednio po lewej lub po prawej stronie.

Przykładowo wyrażenie regularne `"-?[1-9][0-9]*"` opisuje ciągi znaków zawierające liczbę całkowitą: występuje w nich (lub nie) znak `-`, później niezerowa cyfra, później dowolna liczba cyfr 0-9; natomiast wyrażenie `"^[\w._-]+@[\w._-]+.[a-zA-Z]{2,}$"` opisuje (w uproszczeniu) ciągi znaków, które jako całość są poprawnymi adresami email.

Przetwarzanie tekstu w PERL-u

Wyrażenia regularne stosujemy wraz z operatorem `=~` lub `!~` do wyszukiwania i przetwarzania tekstu, najczęściej w postaci:

```
if (tekst =~ /wzorzec/) ...
```

W przypadku dopasowania tekstu do wzorca operator zwraca wartość logiczną `true`, aktualizowane są również wartości zmiennych: `$&` - dopasowany tekst, `$`` - fragment tekstu przed dopasowaniem do wzorca, `$'` - fragment tekstu za dopasowaniem do wzorca, `$1, $2`, itd. - kolejne fragmenty tekstu dopasowane do elementów wzorca otoczonych nawiasami `( )` Uwaga: znajdowany jest pierwszy (z lewej) i najdłuższy fragment tekstu pasujący do wzorca.

Dopasowanie tekstu do wzorca: `m/wzorzec/`

zapisywane najczęściej bez `m` (jak wyżej)

Zamiana tekstu pasującego do wzorca na inny: `s/wzorzec/zamiana/`

uwaga: w określeniu czym zastąpić pasujący fragment można wykorzystać zmienne `$&`, `$1`, itp.

Powyższe operacje mają różne modyfikatory, zapisywane po ostatnim znaku `/` - przykładowo `/i` dopasowuje tekst ignorując wielkość liter, natomiast `/g` znajduje wszystkie dopasowania.

Inne przydatne przy przetwarzaniu funkcje: `split`, `pos`, `splice`