Narzędzia informatyczne w językoznawstwie

Perl - Podstawowe operacje wejścia/wyjścia

Marcin Junczys-Dowmunt
junczys@amu.edu.pl

Zakład Logiki Stosowanej
http://www.logic.amu.edu.pl

5. grudnia 2007

Dzisiejszy wykład

- Omówimy sposoby wczytywania danych z wejścia standardowego[1]
- Zajmiemy się sposobami zapisu danych do wyjścia standardowego i wyjścia błędów
- Omówimy podstawowe sposoby odczytu i zapisu do plików

[1] Proszę przypomnieć sobie informacje z drugiego wykładu o wierszu poleceń

Potoki a Perl

```
C:\> perl test.pl < in.txt > out.txt 2> log.txt
```

- Podobnie jak w przypadku komend wiersza poleceń możemy przekierować strumienie wejścia/wyjścia
- Złożona komenda `perl test.pl` działa jak zwykła komenda wiersza poleceń
- W przykładzie przekierujemy plik `in.txt` na `STDIN`, zapisujemy `STDOUT` do `out.txt`, a `STDERR` do `log.txt`
- Najpierw musimy poznać wewnętrzne mechanizmy Perla pozwalające na przetwarzanie strumieni standardowych

Wczytywanie pojedynczego wiersza z STDIN

```
1  print "Podaj imie: ";

   $name = <STDIN>;
   chomp $name;
5
   print "Witaj, \"$in\"!\n";
```

- z `STDIN` możemy jedynie wczytywać dane
- Służą do tego operator `<...>` oraz operator przypisania `=`
- Fragment `STDIN` to nazwa *uchwytu do pliku*, tutaj do wejścia standardowego
- Wywoływanie operatora `<...>` domyślnie powoduje wczytanie jednego wiersza wraz ze znakiem terminującym
- Funkcja `chomp` usuwa znaki terminujące

Kolejne wczytywanie wszystkich wierszy z STDIN

```
while(defined($line = <STDIN>)) {
  chomp $line;
  print "Wiersz $.. zawiera ".length($line).
  " znakow\n";
}
```

- Kolejne wywołania operatora <...> wczytują kolejne wiersze
- Gdy operator dotrze do końca pliku zwraca wartość `undef`
- Funkcja `defined` sprawdza, czy dana wartość jest różna od `undef` – Dlaczego taka postać warunku?
- Zmienna specjalna `$.` zawiera aktualny numer wiersza

Kolejne wczytywanie wszystkich wierszy z STDIN (krócej)

```
while(<STDIN>) {
  chomp;
  print "Wiersz $.. zawiera ".length($_).
  " znaków\n";
}
```

- Taki zapis jest *idiomem* Perla równoważny z poprzednim przykładem
- Wewnętrznie te dwa programy niczym się nie różnią
- Brak jawnego zapisu do zmiennej, korzystamy ze zmiennej domyślnej `$_`
- Ze zmienną domyślną spotkamy się jeszcze nieraz

Wczytywanie wszystkich wierszy z STDIN do tablicy

```
chomp(@wiersze = <STDIN>);
foreach (@wiersze) {
  print "Wiersz $.. zawiera ".length($_).
  " znaków\n";
}
```

- Użycie operatora <...> w kontekście listowym spowoduje wczytanie wszystkich wierszy do elementów tablicy
- Funkcja `chomp` wykonana na tablicy powoduje obcięcie znaków terminujących w każdym elemencie tablicy
- Znowu pojawia się zmienna domyślna `$_` – iteruje ona po wszystkich elementach tablicy
- Tutaj zmienna `$.` działa w sposób nieoczekiwany – dlaczego?

Zapisywanie do STDOUT

```
print STDOUT "Wypisujemy dane do STDOUT";
```

- Wyjście **standardowe** jest takie jak jego nazwa wskazuje
- Korzystając z `print` domyślnie (standardowo!) zapisujemy do wyjścia standardowego `STDOUT`
- Możemy więc opuścić nazwę uchwytu:

```
print "Wypisujemy dane do STDOUT";
```

Zapisywanie do STDERR

```
1  print STDERR "Wypisujemy dane do STDERR";
```

- Gdy zapisujemy dane do STDERR, musimy jawnie podać nazwę uchwytu

Uchwyty do plików

- Poznaliśmy juz trzy standardowe uchwyty do plików (wirtualnych): STDIN, STDOUT i STDERR
- STDIN to uchwyt otwarty tylko do odczytu
- STDOUT oraz STDERR są otwarte tylko do zapisu
- Możemy tworzyć własne uchwyty do konkretnych plików
- Własne uchwyty obsługujemy tak samo jak uchwyty standardowe

Wczytywanie danych z plików

```
1  open(IN, "<uchwyt.pl")
     or die "plik nie istnieje";

   while(<IN>) {
5    chomp;
     print "Wiersz $.. zawiera ".length($_).
     " znaków\n";
   }
   close(IN);
```

- Funkcja open służy do tworzenie własnych uchwytów
- Podajemy dwa argumenty: nazwę uchwytu, sposób korzystania z pliku połączony nazwę pliku
- Sposób korzystania dla pliku tylko do odczytu oznaczamy przez <

Zapisywanie danych do plików

```
1  open(IN, "<uchwyt.pl")
     or die "plik nie istnieje";
   open(OUT, ">log.txt")
     or die "Nie moglem zapisac danych";
5
   while(<IN>) {
     chomp;
     print OUT "Wiersz $.. zawiera ".length($_).
     " znaków\n";
10 }
   close(IN);
   close(OUT);
```

- Sposób korzystania z uchwytu oznaczamy przez >
- Jak będzie działał znak > a jak znak >> ?
- Zapis do pliku odbywa się jak poprzednio do STDERR

Operator diamentowy <>

```
while(<>) {
  chomp;
  print "Wiersz $.. zawiera ".length($_).
  " znaków\n";
}
```

- Operator diamentowy to kolejny idiom perlowy (perlizm)
- Operator diamentowy wczytuje wszystkie dane ze wszystkich plików podanych w następujący sposób (jako argumenty do programu w wierszu poleceń):

  `perl diament.pl plik1.txt plik2.txt ... plikn.txt`
- Gdy nie podamy żadnego pliku, wczytuje dane z `STDIN`

Tablica specjalna @ARGV

- Wewnętrznie operator diamentowy korzysta ze specjalnej wbudowanej tablicy `@ARGV`
- Ta tablica zawiera wszystkie argumenty podane w wierszu poleceń za nazwą programu

```
for($i = 0; $i < @ARGV; $i++) {
  print "Element o indeksie $i to $ARGV[$i]\n";
}
```

Możemy wykonać powyższy program np. w taki sposób:

```
perl argv.pl zupa tygrys 45 tango 5.7 -h test
```


Specjalny uchwyt plikowy DATA

```
while(<DATA>) {
  chomp;
  print "Wiersz $.. zawiera ".length($_).
  " znaków\n";
}

__END__
Taki sobie tekst
ktory sluzy
jedynie przykladem
```

- Uchwyt `DATA` służy tylko do odczytu danych zapisanych po `__END__`, kod Perla tutaj nie działa
- Przydatne przy testowaniu programów, nie trzeba tworzyć zewnętrznych plików

Podsumowanie

Wiemy teraz jak:

- Wczytywać dane z wejścia standardowego (też z klawiatury)
- Wczytywać dane z dowolnego pliku
- Wczytywać dane ze środowiska `DATA`
- Korzystać ze zmiennej wbudowanej `@ARGV`
- Zapisywać dane do wyjścia standardowego i wyjścia błędów
- Zapisywać dane do dowolnego pliku

Wniosek: Nasze programy od tej chwili potrafią się komunikować ze światem zewnętrznym