Perl
Practical Extraction / Report Language – język programowania pierwotnie stworzony do wydobywania danych z raportów, przetwarzania ich, generowania innych. Współcześnie właściwie uniwersalny.

Skrypty piszemy jako pliki tekstowe. Tradycyjnie rozszerzenie .pl
Linie kończymy średnikiem.

Pierwsza linia skryptu:

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

lub

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

Helo world:

```
#!/usr/local/bin/perl
print "Hello, World!\n";
```

Zmienne:
Każdą zmienną zaczynamy od znaku $. Niezależnie od tego, czy zmienną używamy w zapisie czy odczycie. Jeżeli programujemy w trybie strict (co z początku jest zalecane) – dodajemy linię

```
use strict;
```

na początku programu, pierwsze wystąpienie zmiennej należy poprzedzić zapisem jej zakresu (najczęściej my – zmienna lokalna w bieżącym segmencie kodu). Większość przykładów będzie pisana tak, aby dało się je uruchomić w strict.

```
my $a = 4;
my $g = "zmienna to $a";
my $h = ‘zmienna to $a’;
```

wtedy wartość zmiennej $g to "zmienna to 4", a $h – "zmienna to $a".

Dla celów debugowania możemy wartości wprowadzać z klawiatury:

```
my $variable = <STDIN>;
```

A jeżeli chcemy pozbyć się ewentualnych białych znaków okalających wartość, używamy polecenia:

```
chomp $variable;
```

Jeżeli poddajemy tej operacji tablicę, to chomp zadziała dla każdego jej elementu.

Operatory arytmetyczne
w przeciwieństwie do sh, są wbudowane. +, -, /, *, ** (potęgowanie), % (modulo).
Operator łączenia łańcuchów tekstowych: . (kropka)

Jeżeli

```
my $a="kk";
my $b="kl";

if ($a eq $b)
{
        print "equal\n";
}
else
{
        print "Not equal\n";
}
```

Możliwe testowanie dla liczb: ==, !=, <, >, <=, >=
Dla łańcuchów tekstowych stosujemy **eq** i **ne**! W przeciwnym wypadku (==) warunek jest interpretowany jako ciągle spełniony.

Możemy również używać parametrów testu dla plików:
`if (-e ”plik.txt”) {` - jeżeli plik.txt istnieje
`-z` – jest pusty
`-d` – jest katalogiem
`-r` – jest możliwy do odczytu
`-w` – można do niego zapisać
`-x` – można go uruchomić
Trzy ostatnie testy odczytują atrybuty pliku. Uwaga na przenośność (W Windows nie ma atrybutu -X!)
Znaków ” ” należy użyć, by Perl nie wziął znaku . jako operatora!

Łączenie warunków – or i and (także && i ||):

```
if ($a == $b or $c == 50)
{
…
```

Ze względu na to, że operatory te podobnie do shell’owych pracują na kodzie ostatnio wykonanej operacji, można ich użyć do skracania IF-a, patrz:

```
open (my $MYFILE, "<", "~/.cfg.rc") or die "Configuration file open failed";
```

Jeżeli otwarcie pliku się nie powiedzie, to skrypt zakończy się z błędem. Polecenie **exit;** powoduje wyjście ze skryptu bez kodu błędu, die(„...”) wyprowadza na konsole błąd.

Tablice

W przeciwieństwie do zmiennych, nazwę tablicy zaczynamy od @. W tablicy może znaleźć się wszystko:

```
my @tablica=(2.5, $a, ‘testing’, 2+1);
```

Element tablicy: $tablica[2] == testing

Tablica @ARGV – parametry skryptu.

Numer ostatniego elementu - $#ARGV
Można również używać ujemnych indeksów: $ARGV[-1] gdy argumenty to 2 3 4 wynosi 4.

Iterujemy po tablicy

```
foreach my $var (@ARGV) {
        print $var;
        print "\n";
}
```

W praktyce, przy parsowaniu tekstu i podziale go na linie/rekordy/słowa najczęściej używaną jest właśnie pętla foreach.
Zamiast tablicy (@ARGV) możemy również stosować zakres, np. (1..19) czy (‘ala’, ‘ma’, ‘kota). Nie jest to dopasowanie znaków znane ze skryptów powłoki, lecz generowanie zakresu liczb. Można użyć podzakresu tablicy np. (@ARGV[2..4])

Pętla for - Tutaj podobnie do C:

```
for (my $i = 1; $i < 10; $i++) {
    print $i;
}
```

Pętla while

```
my $counter = 10;
while ($counter > 0)
{
       print $counter."\n";
      $counter -= 2;
}
```

Modyfikacja przebiegu pętli:

last – opuszcza pętlę
next – przerywa iterację i wykonuje kolejną
redo – powtarza bieżącą iterację bez modyfikacji zmiennych sterujących.

Zmienna domyślna $_

Używana jako skrócenie zapisu. Wiele funkcji wbudowanych po prostu działa na tej zmiennej gdy nie dostarczymy im zmiennej.

```
foreach(1..5)
{
  print "$_\n";
}
```

Zadanie:

Napisz program drukujący tabliczkę mnożenia o wymiarze zadanym w pierwszym argumencie programu. Uwzględnij przypadek gdy użytkownik nie podał argumentu.