Narzędzia informatyczne w językoznawstwie

Perl - Zastowanie modułów

Marcin Junczys-Dowmunt
junczys@amu.edu.pl

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

25. marca 2008

Przetwarzanie wielojęzyczne

- Ostatnio omówiliśmy podstawy dotyczące zestawów znaków oraz kodowań
- Dziś przyjrzymy się przetwarzaniu tekstów w różnych kodowaniach, ich wprowadzaniu oraz wyświetlaniu

Przetwarzanie wielojęzyczne

- Ostatnio omówiliśmy podstawy dotyczące zestawów znaków oraz kodowań
- Dziś przyjrzymy się przetwarzaniu tekstów w różnych kodowaniach, ich wprowadzaniu oraz wyświetlaniu
- Pytanie podstawowe: Czy do automatycznego przetwarzania tekstów w różnych kodowaniach musimy być w stanie je wyświetlić?

Przetwarzanie wielojęzyczne

- Ostatnio omówiliśmy podstawy dotyczące zestawów znaków oraz kodowań
- Dziś przyjrzymy się przetwarzaniu tekstów w różnych kodowaniach, ich wprowadzaniu oraz wyświetlaniu
- Pytanie podstawowe: Czy do automatycznego przetwarzania tekstów w różnych kodowaniach musimy być w stanie je wyświetlić?
- Odpowiedź: Nie! Postać cyfrowa oraz informacja o użytym kodowaniu są jednoznaczne.
- Dopiero naoczna kontrola narzuca nam konieczność posiadania czcionek, odpowiednich programów itp.

Wyświetlanie danych Unicode

- Warunek konieczny: Posiadamy czcionki Unicode, np.
  - Arial Unicode MS (38,917 znaków, freeware)
  - Bitstream Cyberbit (32,910 znaków, freeware)
  - Code 2000 (51,239 znaków, shareware)
  - GNU Unifont (33,580 znaków, GPL)
  - New Gulim (46,567 znaków, Microsoft 0ffice 2000)
  - ...

Wyświetlanie danych Unicode

- Warunek konieczny: Posiadamy czcionki Unicode, np.
  - Arial Unicode MS (38,917 znaków, freeware)
  - Bitstream Cyberbit (32,910 znaków, freeware)
  - Code 2000 (51,239 znaków, shareware)
  - GNU Unifont (33,580 znaków, GPL)
  - New Gulim (46,567 znaków, Microsoft 0ffice 2000)
  - ...
- Nie musimy posiadać czcionek, które pokrywają cały Unicode (ok. 100,000 znaków). Wystarczają potrzebne zakresy znaków.
- Np. teksty zapisane w językach europy zachodniej i środkowej prawdopodobnie wyświetlimy za pomocą czcionek standardowych.
- Nie ma na pewno problemów z umlautami äöüß, są czasami z ogonkami itp. ąęćł...

Edytory tekstów

- *Notepad*: okazuje się, że ten prosty edytor w pełni współpracuje z Unicodem.
  - Zapisuje i wczytuje ANSI (CP-1250 według systemu), UTF-8, Unicode (???)
  - Wygląda na to że ISO-8859-2 nie działa

Edytory tekstów

- *Notepad*: okazuje się, że ten prosty edytor w pełni współpracuje z Unicodem.
  - Zapisuje i wczytuje ANSI (CP-1250 według systemu), UTF-8, Unicode (???)
  - Wygląda na to że ISO-8859-2 nie działa
- *EmEditor*: Zalecam!
  - 42 kodowania, w tym rodziny ISO, DOS, Windows CP i różne kodowania Unicode, np. UTF-8, UTF-16, UTF-7 ...
  - Algorytmy rozpoznawania kodowania
  - Może służyć jako konwerter między kodowaniami (do obsługi ręcznej)

Wprowadzanie danych Unicode

- Jeśli mamy potrzebne znaki na klawiaturze nie ma problemów. Program świadomy Unicodu zadba o odpowiednią reprezentację przy zapisie w odpowiednim kodowaniu, najlepiej w UTF-8

Wprowadzanie danych Unicode

- Jeśli mamy potrzebne znaki na klawiaturze nie ma problemów. Program świadomy Unicodu zadba o odpowiednią reprezentację przy zapisie w odpowiednim kodowaniu, najlepiej w UTF-8
- Jeśli ich nie mamy:
  - Zmiana układu i języka klawiatury

Wprowadzanie danych Unicode

- Jeśli mamy potrzebne znaki na klawiaturze nie ma problemów. Program świadomy Unicodu zadba o odpowiednią reprezentację przy zapisie w odpowiednim kodowaniu, najlepiej w UTF-8
- Jeśli ich nie mamy:
  - Zmiana układu i języka klawiatury
  - Edycja układu klawiatury: np. *Microsoft Keyboard Layout Editor*

Wprowadzanie danych Unicode

- Jeśli mamy potrzebne znaki na klawiaturze nie ma problemów. Program świadomy Unicodu zadba o odpowiednią reprezentację przy zapisie w odpowiednim kodowaniu, najlepiej w UTF-8
- Jeśli ich nie mamy:
  - Zmiana układu i języka klawiatury
  - Edycja układu klawiatury: np. *Microsoft Keyboard Layout Editor*
  - Klawiatura ekranowa z odpowiednim układem

Wprowadzanie danych Unicode

- Jeśli mamy potrzebne znaki na klawiaturze nie ma problemów. Program świadomy Unicodu zadba o odpowiednią reprezentację przy zapisie w odpowiednim kodowaniu, najlepiej w UTF-8
- Jeśli ich nie mamy:
  - Zmiana układu i języka klawiatury
  - Edycja układu klawiatury: np. *Microsoft Keyboard Layout Editor*
  - Klawiatura ekranowa z odpowiednim układem
  - Tablica Znaków

Wprowadzanie danych Unicode

- Jeśli mamy potrzebne znaki na klawiaturze nie ma problemów. Program świadomy Unicodu zadba o odpowiednią reprezentację przy zapisie w odpowiednim kodowaniu, najlepiej w UTF-8
- Jeśli ich nie mamy:
  - Zmiana układu i języka klawiatury
  - Edycja układu klawiatury: np. *Microsoft Keyboard Layout Editor*
  - Klawiatura ekranowa z odpowiednim układem
  - Tablica Znaków
  - Kody Alt + Numer znaku (nie działa u mnie, a u Państwa?)

Tablica Znaków

- Uruchamianie tablicy znaków (character map) w Windows XP
  - Uruchom > `charmap`
  - Wszystkie Programy > Akcesoria > Narzędzia systemowe > Tablica znaków

Tablica Znaków

- Uruchamianie tablicy znaków (character map) w Windows XP
  - Uruchom > `charmap`
  - Wszystkie Programy > Akcesoria > Narzędzia systemowe > Tablica znaków
- Wyświetla znaki dostępne dla wybranej czcionki według danego kodowania

Tablica Znaków

- Uruchamianie tablicy znaków (character map) w Windows XP
  - Uruchom > `charmap`
  - Wszystkie Programy > Akcesoria > Narzędzia systemowe > Tablica znaków
- Wyświetla znaki dostępne dla wybranej czcionki według danego kodowania
- Opcja zaawansowana: można grupować według zakresów językowych lub tematycznych

Tablica Znaków

- Uruchamianie tablicy znaków (character map) w Windows XP
  - Uruchom > `charmap`
  - Wszystkie Programy > Akcesoria > Narzędzia systemowe > Tablica znaków
- Wyświetla znaki dostępne dla wybranej czcionki według danego kodowania
- Opcja zaawansowana: można grupować według zakresów językowych lub tematycznych
- Ważne: W stopce wyświetla kod Unicode danego znaku i jego pełną nazwę, np.
  ą : U+0105 - LATIN SMALL LETTER A WITH OGONEK
  Ä : U+00C4 - LATIN CAPITAL LETTER A WITH DIAERESIS

Kodowania i Unicode a Konsola

- Z powodów znanych pewnie tylko Micorsoftowi konsola korzysta z innego kodowania niż Windows CP-1250 (na polskich systemach), a mianowicie z CP-852
- Czyli obok ISO-8859-2, Windows CP-1250 pojawia nam się Windows CP-852
- Kodowanie konsoli dotyczy danych wyświetlanych i wprowadzanych w konsoli

Kodowania i Unicode a Konsola

- Z powodów znanych pewnie tylko Micorsoftowi konsola korzysta z innego kodowania niż Windows CP-1250 (na polskich systemach), a mianowicie z CP-852
- Czyli obok ISO-8859-2, Windows CP-1250 pojawia nam się Windows CP-852
- Kodowanie konsoli dotyczy danych wyświetlanych i wprowadzanych w konsoli
- Do wyświetlenia aktualnego kodowania służy komenda `chcp` (Change Code Page) bez argumentu

Kodowania i Unicode a Konsola

- Z powodów znanych pewnie tylko Micorsoftowi konsola korzysta z innego kodowania niż Windows CP-1250 (na polskich systemach), a mianowicie z CP-852
- Czyli obok ISO-8859-2, Windows CP-1250 pojawia nam się Windows CP-852
- Kodowanie konsoli dotyczy danych wyświetlanych i wprowadzanych w konsoli
- Do wyświetlenia aktualnego kodowania służy komenda `chcp` (Change Code Page) bez argumentu
- Do zmiany kodowania wykorzystujemy tą samą komendę, np.
  - chcp 852 - Windows CP-852 (Latin 2) $\neq$ ISO LATIN 2 (ISO-8859-2)
  - chcp 1250 - Windows CP-1250 (europa środkowa)
  - chcp 1252 - Windows CP-1252 (europa zachodnia)
  - chcp 65001 - UTF-8 (!!!) i nawet działa czasem

Unicode a Perl

- W Perlu teoretycznie nie ma problemu z Unicodem, ponieważ wewnętrznie wszystko jest reprezentowane w Unicodzie

Unicode a Perl

- W Perlu teoretycznie nie ma problemu z Unicodem, ponieważ wewnętrznie wszystko jest reprezentowane w Unicodzie
- Ale jakie to jest kodowanie? UTF-7, UTF-8, UTF-16 ...?

Unicode a Perl

- W Perlu teoretycznie nie ma problemu z Unicodem, ponieważ wewnętrznie wszystko jest reprezentowane w Unicodzie
- Ale jakie to jest kodowanie? UTF-7, UTF-8, UTF-16 ...?
- Teoretycznie utf8 (nie UTF-8!), a praktycznie coś binarnego

Unicode a Perl

- W Perlu teoretycznie nie ma problemu z Unicodem, ponieważ wewnętrznie wszystko jest reprezentowane w Unicodzie
- Ale jakie to jest kodowanie? UTF-7, UTF-8, UTF-16 ...?
- Teoretycznie utf8 (nie UTF-8!), a praktycznie coś binarnego
- Załóżmy, że to nieznane nam kodowanie, o którym wiemy, że ładnie działa. Trzeba tylko wszystko do niego sprowadzić.

Unicode a Perl

- W Perlu teoretycznie nie ma problemu z Unicodem, ponieważ wewnętrznie wszystko jest reprezentowane w Unicodzie
- Ale jakie to jest kodowanie? UTF-7, UTF-8, UTF-16 ...?
- Teoretycznie utf8 (nie UTF-8!), a praktycznie coś binarnego
- Załóżmy, że to nieznane nam kodowanie, o którym wiemy, że ładnie działa. Trzeba tylko wszystko do niego sprowadzić.
- Korzystamy z tego kodowanie **tylko i wyłącznie wewnątrz** programu

Złota zasada numer 1

- **Wszystkie** dane wejściowe konwertujemy **przed rozpoczęciem przetwarzania** z kodowania źródłowego do kodowania wewnętrznego Perla.

Złota zasada numer 1

- **Wszystkie** dane wejściowe konwertujemy **przed rozpoczęciem przetwarzania** z kodowania źródłowego do kodowania wewnętrznego Perla.
- **Wszystkie** dane wyjściowe konwertujemy **po zakończeniu przetwarzania** z kodowania wewnętrznego Perla do kodowania docelowego.

Złota zasada numer 1

- **Wszystkie** dane wejściowe konwertujemy **przed rozpoczęciem przetwarzania** z kodowania źródłowego do kodowania wewnętrznego Perla.
- **Wszystkie** dane wyjściowe konwertujemy **po zakończeniu przetwarzania** z kodowania wewnętrznego Perla do kodowania docelowego.

Dlaczego?

- Nie ma wtedy problemów z wyrażeniami regularnymi

Złota zasada numer 1

- **Wszystkie** dane wejściowe konwertujemy **przed rozpoczęciem przetwarzania** z kodowania źródłowego do kodowania wewnętrznego Perla.
- **Wszystkie** dane wyjściowe konwertujemy **po zakończeniu przetwarzania** z kodowania wewnętrznego Perla do kodowania docelowego.

Dlaczego?

- Nie ma wtedy problemów z wyrażeniami regularnymi
- Działa np. zamiana wielkości liter za pomocą `uc` i `lc`

Złota zasada numer 1

- **Wszystkie** dane wejściowe konwertujemy **przed rozpoczęciem przetwarzania** z kodowania źródłowego do kodowania wewnętrznego Perla.
- **Wszystkie** dane wyjściowe konwertujemy **po zakończeniu przetwarzania** z kodowania wewnętrznego Perla do kodowania docelowego.

Dlaczego?

- Nie ma wtedy problemów z wyrażeniami regularnymi
- Działa np. zamiana wielkości liter za pomocą `uc` i `lc`
- Konwersja do i z kodowanie wewnętrznego jest prosta

Złota zasada numer 1

- **Wszystkie** dane wejściowe konwertujemy **przed rozpoczęciem przetwarzania** z kodowania źródłowego do kodowania wewnętrznego Perla.
- **Wszystkie** dane wyjściowe konwertujemy **po zakończeniu przetwarzania** z kodowania wewnętrznego Perla do kodowania docelowego.

Dlaczego?

- Nie ma wtedy problemów z wyrażeniami regularnymi
- Działa np. zamiana wielkości liter za pomocą `uc` i `lc`
- Konwersja do i z kodowanie wewnętrznego jest prosta
- Inne programy nie rozumieją tego kodowania

Złota zasada numer 2

- **Poza** Perlem **wszystkie** pliki tekstowe sprowadzamy do kodowania UTF-8 (o ile to możliwe)

Złota zasada numer 2

- **Poza** Perlem **wszystkie** pliki tekstowe sprowadzamy do kodowania UTF-8 (o ile to możliwe)

Dlaczego?

- UTF-8 jest najbardziej popularnym kodowaniem Unicode (można stosować prawie zamiennie, o ile się pamięta różnicę. Jaka jest różnica?)

Złota zasada numer 2

- **Poza** Perlem **wszystkie** pliki tekstowe sprowadzamy do kodowania UTF-8 (o ile to możliwe)

Dlaczego?

- UTF-8 jest najbardziej popularnym kodowaniem Unicode (można stosować prawie zamiennie, o ile się pamięta różnicę. Jaka jest różnica?)
- Nie musimy pamiętać w jakim kodowaniu jest dany plik tekstowy (bo to zawsze UTF-8)

Złota zasada numer 2

- **Poza** Perlem **wszystkie** pliki tekstowe sprowadzamy do kodowania UTF-8 (o ile to możliwe)

Dlaczego?

- UTF-8 jest najbardziej popularnym kodowaniem Unicode (można stosować prawie zamiennie, o ile się pamięta różnicę. Jaka jest różnica?)
- Nie musimy pamiętać w jakim kodowaniu jest dany plik tekstowy (bo to zawsze UTF-8)
- Wszystkie systemy znakowe możemy kodować tym samym kodowaniem, w końcu to kodowanie Unicode.

Złota zasada numer 2

- **Poza** Perlem **wszystkie** pliki tekstowe sprowadzamy do kodowania UTF-8 (o ile to możliwe)

Dlaczego?

- UTF-8 jest najbardziej popularnym kodowaniem Unicode (można stosować prawie zamiennie, o ile się pamięta różnicę. Jaka jest różnica?)
- Nie musimy pamiętać w jakim kodowaniu jest dany plik tekstowy (bo to zawsze UTF-8)
- Wszystkie systemy znakowe możemy kodować tym samym kodowaniem, w końcu to kodowanie Unicode.
- Możemy używać dokładnie te same programy do przetwarzania tekstów w różnych językach (ale w tych samym kodowaniu)

Konwersja przy korzystaniu z uchwytów - Warstwy

```
use strict;

binmode(STDIN, ":encoding(utf8)");
binmode(STDOUT, ":encoding(utf8)");
binmode(STDERR, ":encoding(cp852)");

open(LOG, ">:encoding(utf8)", "log.txt");

while(<STDIN>) {
    my $c1 = s/\b(\p{Ll})/uc($1)/eg;
    print STDERR "W $. powiekszono $c1 liter\n";
    my $c2 = s/\b(\p{Lu})/lc($1)/eg;
    print STDERR "W $. zmniejszono $c2 liter\n";
    print $_;
    print LOG "$.\t$c1\t$c2\n"
}
```

Gdy nie ma uchwytów...

- Nasze dane nie zawsze będą pochodzić z plików zewnętrznych czy strumieni standardowych

Gdy nie ma uchwytów...

- Nasze dane nie zawsze będą pochodzić z plików zewnętrznych czy strumieni standardowych
- Tak samo nie będziemy zawsze zapisywali do plików czy do wyjścia standardowego

Gdy nie ma uchwytów...

- Nasze dane nie zawsze będą pochodzić z plików zewnętrznych czy strumieni standardowych
- Tak samo nie będziemy zawsze zapisywali do plików czy do wyjścia standardowego
- Sytuacje, w których nie możemy korzystać z kodowania za pomocą warstw:
  - Komunikujemy się z bazą danych. Odbywa się to za pomocą modułów perlowych (np. DBI). Informacje są zwracane przez funkcje tych modułów jako łańcuchy znakowe.

Gdy nie ma uchwytów...

- Nasze dane nie zawsze będą pochodzić z plików zewnętrznych czy strumieni standardowych
- Tak samo nie będziemy zawsze zapisywali do plików czy do wyjścia standardowego
- Sytuacje, w których nie możemy korzystać z kodowania za pomocą warstw:
  - Komunikujemy się z bazą danych. Odbywa się to za pomocą modułów perlowych (np. DBI). Informacje są zwracane przez funkcje tych modułów jako łańcuchy znakowe.
  - Ściągamy strony internetowe za pomocą modułu LWP::Simple. Strony te dostajemy w postaci pojedynczego łańcucha znakowego.

Gdy nie ma uchwytów...

- Nasze dane nie zawsze będą pochodzić z plików zewnętrznych czy strumieni standardowych
- Tak samo nie będziemy zawsze zapisywali do plików czy do wyjścia standardowego
- Sytuacje, w których nie możemy korzystać z kodowania za pomocą warstw:
  - Komunikujemy się z bazą danych. Odbywa się to za pomocą modułów perlowych (np. DBI). Informacje są zwracane przez funkcje tych modułów jako łańcuchy znakowe.
  - Ściągamy strony internetowe za pomocą modułu LWP::Simple. Strony te dostajemy w postaci pojedynczego łańcucha znakowego.
  - Komunikujemy się z innym programem nieperlowym przez specjalny interfejs.

Gdy nie ma uchwytów...

- Nasze dane nie zawsze będą pochodzić z plików zewnętrznych czy strumieni standardowych
- Tak samo nie będziemy zawsze zapisywali do plików czy do wyjścia standardowego
- Sytuacje, w których nie możemy korzystać z kodowania za pomocą warstw:
  - Komunikujemy się z bazą danych. Odbywa się to za pomocą modułów perlowych (np. DBI). Informacje są zwracane przez funkcje tych modułów jako łańcuchy znakowe.
  - Ściągamy strony internetowe za pomocą modułu LWP::Simple. Strony te dostajemy w postaci pojedynczego łańcucha znakowego.
  - Komunikujemy się z innym programem nieperlowym przez specjalny interfejs.
  - Korzystamy z Parsera XML napisanego w C++ (np. XML::Expat).
  - ...

Moduł Encode - czyli konwersja łańcuchów znakowych

```
use strict;
use Encode qw(encode decode_utf8 encode_utf8);

open(LOG, ">", "log.txt");

while(<>) {
 $_ = decode_utf8($_);

 my $c1 = s/\b(\p{Ll})/uc($1)/eg;
 print STDERR encode("cp852","$. pow. $c1 lit.\n");
 my $c2 = s/\b(\p{Lu})/lc($1)/eg;
 print STDERR encode("cp852","$. zmn. $c2 lit.\n");
 print $_;
 print LOG encode_utf8("$.\t$c1\t$c2\n");
}
```

Wewnętrzne kodowanie znaków a kod programu

```
use strict;
binmode(STDIN, ":utf8"); binmode(STDOUT, ":utf8");

my $count = 0;
while(<>) {
  if(/\x{0119}\x{0107}/) {
    print "$. - Znalaz\x{0142}em ";
    print "\"\x{0119}\x{0107}\"!\n";
    $count++;
  }
}
print "Znalaz\x{0142}em \"\x{0119}\x{0107}\" ";
print "a\x{017c} $count razy\n";
```

Wewnętrzne kodowanie znaków a kod programu

```
use strict;
binmode(STDIN, ":utf8"); binmode(STDOUT, ":utf8");

my $count = 0;
while(<>) {
  if(/\x{0119}\x{0107}/) {
    print "$. - Znalaz\x{0142}em ";
    print "\"\x{0119}\x{0107}\"!\n";
    $count++;
  }
}
print "Znalaz\x{0142}em \"\x{0119}\x{0107}\" ";
print "a\x{017c} $count razy\n";
```

- \x{nnnn} jest odpowiednikiem numeru unicodowego U+nnnn

Wewnętrzne kodowanie znaków a kod programu

```
use strict;
binmode(STDIN, ":utf8"); binmode(STDOUT, ":utf8");

my $count = 0;
while(<>) {
  if(/\x{0119}\x{0107}/) {
    print "$. - Znalaz\x{0142}em ";
    print "\"\x{0119}\x{0107}\"!\n";
    $count++;
  }
}
print "Znalaz\x{0142}em \"\x{0119}\x{0107}\" ";
print "a\x{017c} $count razy\n";
```

- \x{nnnn} jest odpowiednikiem numeru unicodowego U+nnnn
- Istnieje zapis będący odpowiednikiem nazwy znaku?

Wewnętrzne kodowanie znaków a kod programu

```
use strict;
use charnames qw(latin);
binmode(STDIN, ":utf8"); binmode(STDOUT, ":utf8");

my $count = 0;
while(<>) {
  if(/\N{e with ogonek}\N{c with acute}/) {
    print "$. - Znalaz\x{0142}em ";
    print "\"\x{0119}\x{0107}\"!\n";
    $count++;
  }
}
print "Znalaz\N{l with stroke}em \"\x{0119}";
print "\x{0107}\" a\x{017c} $count razy\n";
```

Wewnętrzne kodowanie znaków a kod programu

```
use strict;
use charnames qw(latin);
binmode(STDIN, ":utf8"); binmode(STDOUT, ":utf8");

my $count = 0;
while(<>) {
  if(/\N{e with ogonek}\N{c with acute}/) {
    print "$. - Znalaz\x{0142}em ";
    print "\"\x{0119}\x{0107}\"!\n";
    $count++;
  }
}
print "Znalaz\N{l with stroke}em \"\x{0119}";
print "\x{0107}\" a\x{017c} $count razy\n";
```

- W zależności od opcji pragmy charnames możemy korzystać z nazw skróconych lub pełnych (l with stroke, LATIN SMALL LETTER L WITH STROKE)

Pragma utf8 i jego niebezpieczeństwa

```
use strict;
use utf8;

binmode(STDIN, ":utf8");
binmode(STDOUT, ":utf8");
binmode(STDERR, ":encoding(cp852)");

while(<STDIN>) {
  my $c = tr/ąćęłńóśżźĄĆĘŁŃÓŚŻŹ/acelnoszzACELNOSZZ/;
  print STDERR "W $. znormalizowałem $c znaków\n";
  print;
}
```

Pragma utf8 i jego niebezpieczeństwa

```
use strict;
use utf8;

binmode(STDIN, ":utf8");
binmode(STDOUT, ":utf8");
binmode(STDERR, ":encoding(cp852)");

while(<STDIN>) {
  my $c = tr/ąćęłńóśżźĄĆĘŁŃÓŚŻŹ/acelnoszzACELNOSZZ/;
  print STDERR "W $. znormalizowałem $c znaków\n";
  print;
}
```

- Za pomocą pragmy utf8 możemy stosować bezpiecznie polskie (i wszystkie inne) znakie w kodzie. Nawet w nazwach zmiennych i funkcji.

Pragma utf8 i jego niebezpieczeństwa

```
use strict;
use utf8;

binmode(STDIN, ":utf8");
binmode(STDOUT, ":utf8");
binmode(STDERR, ":encoding(cp852)");

while(<STDIN>) {
  my $c = tr/ąćęłńóśżźĄĆĘŁŃÓŚŻŹ/acelnoszzzACELNOSZZ/;
  print STDERR "W $. znormalizowałem $c znaków\n";
  print;
}
```

- Za pomocą pragmy utf8 możemy stosować bezpiecznie polskie (i wszystkie inne) znakie w kodzie. Nawet w nazwach zmiennych i funkcji.
- Ale program **musi** być zapisany w kodowaniu UTF-8! Inaczej nie będzie działał.