Dokumentace úlohy SMS: SMS Compress v Perl do IPP 2010/2011

Jméno a příjmení: Martin Knapovský
Login: xknapo02

Toto je dokumentace popisující návrh a implementaci projektu do předmětu „Principy programovacích jazyků a OOP“, který dle parametrů příkazového řádku provádí konverzi vstupního textu do normalizované formy bez diakritiky, dále používá expanzivní a redukční pravidla pro nahrazení podřetězců dle pravidel určených slovníkem a převádí text do tzv. Camel notace.

Parametry programu a jejich význam je dostupný přímo v programu zadáním parametru --help, nebo spuštěním programu bez parametrů.

Obecné informace o implementaci

Pro uložení informací, které je potřeba mít dostupné v celém programu jsou použity globální proměnné, které obsahují informace o názvech vstupních a výstupních souborů, o pravidlech pro expanzi a redukci a také o typech parametrů, které byly zadány na příkazové řádce.

Pro implementaci je použit mód `strict` jazyka Perl, dále kódování pomocí `use utf8` a `use locale`. Program používá jediný externí modul `IO::File` pro načtení a ukládání souborů s kódováním UTF-8.

Pro uložení načteného řetězce a jeho následné konverze je použito globální pole `$processed`.

Odstranění diakritiky

Jazyk Perl je velmi dobře vybaven pro práci s textem, a proto je kód pro nahrazení znaků s diakritikou jejich ekvivalentem bez diakritiky velmi jednoduchý.

```
$string =~ tr/ĚŠČŘŽĎŤŇÁÉÍÓÚŮÝěščřžďťňáéíóúůý/ESCRZDTNAEIOUUYescrzdtnaeiouuy/;
```

Informace o tom, zda byla provedena konverze alespoň jednoho znaku je navracena zpět z funkce, což je vhodné pro určení typu vstupního řetězce (s diakritikou, bez diakritiky).

Převod do Camel notace

Zadání projektu specifikovalo 3 typy převodu do Camel notace: klasický převod, převod slov psaných malým písmem a převod všech slov kromě slov psaných velkým písmem.

Algoritmus převodu a jeho implementace

Nejprve převedeme vstupní řetězec na malá písmena a poté ho rozdělíme na znaky, které následně procházíme.

```
my @chars = split "", $line;
```

Procházíme tedy řetězcem po znacích a rozpoznáváme jejich typ. Dále musíme vědět, zda se nacházíme uvnitř slova, nebo zda je načtený znak jeho počátkem – k tomu slouží proměnná `my $inside_word.`

Pokud je zpracovávaným znakem písmeno a zároveň se nacházíme na začátku slova, převedeme písmeno na písmeno velké, jelikož jsme se tak dostali dovnitř slova, nastavíme příznak `$inside_word` a převedený znak přidáme do globálního pole `$processed`. Pokud dále načítáme znaky, zůstává příznak `$inside_word` nastavený a znaky bez konverze přidáváme do pole `$processed`. Při načtení jakéhokoliv jiného znaku než je písmeno vypneme příznak `$inside_word` a pokud se nejedná o mezeru, přídáme ho opět do pole `$processed.` Takto zpracováváme řetězec až do načtení jeho posledního znaku.

Převod slov složených pouze z malých znaků a slov, která nejsou zapsána velkými písmeny do Camel notace

Pro tyto konverze je využito předešlého algoritmu pro převod řetězce do Camel notace, avšak pokud narazíme na slovo, pak ho po znacích ukládáme do proměnné `$buffer` a při přečtení jiného než písmenného znaku zkontrolujeme, zda slovo v proměnné `$buffer` je složeno pouze z malých, nebo velkých písmen a na základě výsledku kontroly převádíme slovo do Camel notace, nebo ho ponecháváme v současné podobě a přidáváme ho do globální proměnné `$processed`.

Expanze a redukce pomocí slovníkových pravidel

Pro použití pravidel ze slovníku bylo nutné nejprve slovník zpracovat do podoby, s níž je možno pracovat v rámci programu.

Jednotlivé řádky souboru XML jsou načítány a rozpoznávány pomocí regulárních výrazů. Korektní slovník pravidel musí obsahovat párový tag `<sms-abbreviation-dictionary>`, jehož přítomnost je testována a pokud není nalezen, je slovník nesprávně definovaný, a tudíž je vytištěna příslušná chybová hláška. Při načtení párového tagu `<rule>`, který může obsahovat další atributy `casesensitive` a `expansive` (to zda se jedná o expanzivní pravidlo, nebo redukční) vyhledáváme jeho elementy `<abbrev> a <text>`. Tyto elementy musí být přitomny oba, avšak nemusí být přítomny přesně v uvedeném pořadí. Při jejich nalezení uchováváme jejich položky do hashovacího pole `%rule_hash`, na nějž při načtení ukončovacího tagu pravidla `</rule>` uchováváme ukazatel do daného seznamu pravidel, který je určen podle toho, zda se jedná o expanzivní, či redukční pravidlo a dále zda se jedná o pravidlo typu case-sensitive. Pokud nejsou oba elementy pravidla, nebo ukončovací tag nalezeny, pak se jedná o nesprávně definovaný slovník, je vytištěna chybová hláška a program je ukončen s odpovídající návratovou hodnotou.

Celkem tedy máme 4 pole s ukazateli na hashovací pole obsahující pravidla, která jsou následně procházena a uplatňována dokud v řetězci nacházíme podřetězce, na něž se pravidla mají aplikovat.

Určení počtu SMS nutného pro uložení řetězce

Jak již bylo popsáno výše, při převodu vstupního řetězce na řetězec bez diakritiky je využito funkce, která navrací příznak, zda vstupní řetězec obsahoval diakritiku, či nikoliv. Tohoto je využito při výpočtu počtu SMS na které je potřeba výstupní řetězec rozdělit. Pokud zpráva obsahuje interpunkci a je delší než 70 znaků, pak délku zprávy vydělíme číslem 67 (počet znaků, který je schopna uložit jedna SMS při zprávě skládající se z více SMS zpráv) , přičteme 1 a odsekneme desetinnou část výsledku, jelikož určujeme minimální nutný počet SMS, který je celočíselný. Obdobně je prováděn výpočet pro řetězec bez diakritiky (samotná SMS bez diakritiky je schopna obsáhnout 160 znaků a delší zpráva z více SMS 153 znaků).

Pozn.: Pro další informace je zdrojový kód programu podrobně komentovaný.