Dokumentace úlohy CSV: CSV2XML v perlu do IPP 2011/2012
Jméno a příjmení: Karel Hala
Login: xhalak00

Úvod

Za úkol jsem měl převést CSV dokument do XML dokumentu. CSV je formát, popisující dokument, ve kterém jsou jednotlivé záznamy odděleny oddělovačem.

Práce s argumenty

Cely program pracuje s hashovací tabulkou, jejíž jednotlivé položky jsou okomentovány a vysvětleny v kódu.
Zpracování parametrů jsem prováděl pomocí rozparsování pomocí grep. Měl jsem na výběr z getopts::long. Nicméně jsem zpracování parametrů provedl již na začátku semestru, tak jsem se do getopts::long nepouštěl. Nicméně po zjištění, co by obnášelo použití knihovny getopts::long ničeho nelituji.
Jak jsem již nastínil, parametry rozděluji pomocí grep a několika jednoduchých regulárních výrazů. Mám jistotu, jak se bude script v danou chvíli chovat. Zpracování parametrů je v subrutině s názvem get_params(@ARGV) subrutina požaduje pole argumentů, která bere jako jediný vstupní parametr pole argumentů. V případě, že je zavolán parametr --help tisknu nápovědu. Dále hlídám různé kombinace parametrů a zda jsou zadání správně.
Na konci subrutiny get_params kontroluji určité speciální stavy. Jako například prázdný název souboru, prázdný obalovací element a podobné. V momentě kdy načtu --missing-val=““ budu nahrazovat prázdné pozice prázdným řetězcem. Přijde mi to jako nejlogičtější a nejelegantnější řešení. Při chybějících hodnotách u parametrů -r a -l vypisuji chybu 30. Subrutina vrací naplněnou hashovací tabulku.

Načítání a tisknutí dat

Druhá subrutina vykonává většinu práce. Jmenuje se work(%hash), subrutina požaduje hash tabulku, díky které se řídí. Nejdříve načte soubor, nebo načte data ze standardního vstupu. Ověří, zda je soubor v pořádku a data nejsou nějak porušená. Pracuji s knihovnou Txt::CSV[1], jelikož je nejrychlejší a nejrobustnější pro zpracování CSV souboru. Za pomoci CSV->getline() načítám postupně každý řádek souboru.
Knihovnu CSV jsem použil v nastavení binary = 1 pro možnost zpracovat řádky v dokumentu. Konec řádku jsem nastavil na **CRLF** a oddělovač na aktuální oddělovač. V případě, že uživatel požaduje jako oddělovač tabulátor (TAB), nastavím oddělovač na \t. Jak je uvedeno v dokumentaci k dané knihovně.
Tisknutí samostatného XML kódu provádím ručně. Mám radši, když vím přesně co program, kdy dělá a já mám nad ním plnou kontrolu. Proto jsem si zvolil tuto možnost. Nejdříve načíst data pomocí knihovny getline a poté tisknout data podle sebe. Je spousta knihoven na tisk XML kódu, nicméně, já jsem jsem si zvolil tisknutí postupně jednoho elementu za druhým.
Jak jsem již napsal, data načtené CSV knihovnou jsou v dvourozměrném poli, které postupně procházím po řádcích a poté po jednotlivých datech. Kontroluji postupně, zda se v dokumentu nachází správné množství sloupců. Pokud ne, končím s chybou. Data netisknu hned, jak je načtu z pole, ale postupně je ukládám do proměnné. Pokud bych je tiskl, hned jak je načtu z pole, musel bych řešit mazání souboru a to není moc dobrá praktika. Navíc by program obsahoval spoustu podmínek otevření a zavření souboru, během, kterých by vznikalo spoustu chyb.
Celková generace dat probíhá ve 2 cyklech. Nejdříve načtu řádek, ten poté uložím do proměnné, kterou postupně procházím po jednotlivých datech. V průběhu první cyklu kontroluji dostatečný počet sloupců. V těle druhého cyklu převádím nevalidní znaky pomocí subrutiny validate($string), subrutina požaduje řetezec, který zvaliduje.
Jednotlivé elementy kontroluji na nevalidní znaky[2] pomocí subrutiny invalid_tag($string), subrutina požaduje řetězec, který bude kontrolovat. Nejdříve nahradím nevalidní znaky pomlčkou („-“) a poté zkontroluji, zda je element stále platný a validní. Pokud ne rovnou končím s chybou.
Dále data ukládám do proměnné, podle toho, zda jsem načetl hlavičku. Nebo nemám ošetřovat různé délky sloupců a podobně. Ošetření různé délky sloupců mi nabobtnalo na 4 podmínky. Pravděpodobně tohle a dlouhý zápis šel obejít použitím nějaké knihovny na generování XML souboru, ale já mám radši 100% kontrolu nad tím co program dělá.
Pokud zpracování načtených dat proběhlo v pořádku a vygenerování výsledného XML dokumentu vede k validnímu XML dokumentu. Data vytisknu podle volby uživatele na standardní výstup, nebo do souboru.

Ošetření některých chyb a nejasností

Pokud knihovna CSV načte špatný soubor, výsledek se uloží do error_input () a vrátím chybu číslo 4. Pokud načtu prázdný soubor, script končí s chybou číslo 30. Nevalidní XML.
Pokud uživatel zadá stejný vstupní a výstupní soubor. Načtu data ze vstupního souboru, ty upravím a poté přepíši soubor. Takže se ztratí data z CSV souboru a ty se nahradí za XML data. Tímto způsoben se chová většina aplikací, tak jsem tohle považoval za správný přístup.
V případě že poslední řádek je ukončen **CRLF**, tak se script chová, jakoby v dokumentu byl řádek navíc a pokud není dostatečný počet sloupců a není zapnut error recovery, script končí s chybou 32.

1Text::CSV. *Text::CSV* [online]. [cit. 2012-04-06]. Dostupné z: http://search.cpan.org/~makamaka/Text-CSV-1.21/lib/Text/CSV.pm

2Names and Tokens podle specifikace XML. In: [online]. [cit. 2012-04-06]. Dostupné z: http://www.w3.org/TR/REC-xml/#sec-common-syn