Dokumentace úlohy CLS: C++ Classes v PHP 5 do IPP 2015/2016
Jméno a příjmení: Peter Tisovčík
Login: xtisov00

1 Úvod

Tato dokumentace popisuje implementaci skriptu v jazyce PHP 5. Úkolem vytvořeného skriptu je analýza hlavičkových souborů jazyka C++11 a jejich následná reprezentace pomocí XML syntaxe. V dokumentaci se nachází popis implementace zpracování parametrů programu, parsování vstupního hlavičkového souboru a následného generování výstupního souboru v XML formátu.

2 Zpracování parametrů

Ke zpracování vstupních parametrů slouží v PHP funkce `getopt()`, která načte parametry na základě zadaných přepínačů. Samotná funkce `getopt()` však nestačí k ošetření všech možností, které ze zadání vyplývají. Tyto nedostatky řeší třída `InputArgv`.

Tato třída ošetřuje situace, kdy jsou zadány nesprávné argumenty, nesprávné parametry argumentů jako například neexistující vstupní soubor, zadání nesprávné hodnoty přepínače `pretty-xml`, zadání krátkých přepínačů a jejich kolize s dlouhými přepínači.

3 Načtení hlavičkového souboru C++11

Pro usnadnění zpracování vstupního hlavičkového souboru slouží třída `ParseInputFile`. Třída obsahuje metodu `ParseInput()`, která v sobě implementuje konečný automat, na základě kterého se analyzují jednotlivé části vstupního souboru.

Konečný automat využívá při své analýze metodu `ReadTo()`, jejímž parametrem je pole znaků, po které má načítat. Zanalyzované části vstupního souboru se následně ukládají do pole objektů typu `TemplateClass`, které reprezentují třídu ve vstupním hlavičkovém souboru. Třída v sobě obsahuje všechny potřebné informace, které budou později potřeba pro generování výstupního XML souboru.

Mezi nejdůležitější metody, které se používají v konečném automatu, patří metoda `CopyClass()`, která se využívá při dědičnosti a zkopíruje informace o metodách a atributech do aktuální třídy. Další velmi důležitou metodou je `CheckClass()`, která umožňuje redefinovat metody a atributy. Zároveň tato metoda volá metodu `CheckConflict()`, která kontroluje, zda nenastaly konflikty při dědění. Nachází se tu i metoda `addUsing()` pro zpřístupnění metody či atributu v daném jmenném prostoru, díky které je možno předejít konfliktům.

4 Generování výstupního XML

Pro generování XML výstupu byla použita třída `XMLWriter`. Objekt z této třídy reprezentoval strukturu vstupního souboru, která byla uložena v poli objektů pro analyzované třídy - `TemplateClass`. Na výstup mohly být vygenerovány tři typy XML výstupů. Struktura byla podobná, avšak každý typ v sobě ukrýval jiná úskalí.

Strom dědičnosti popisuje dědičnost mezi třídami, pro vygenerování tohoto výstupu byla použita metoda `generateB()`, která rekurzivně prohledává jednotlivé objekty zpracovaných tříd a na výstup vypisuje potřebné informace. Při zpracování vstupního souboru bylo třeba správně identifikovat, zda sa jedná o abstraktní nebo konkrétní metodu. Dalším typem byl výpis detailů o zadané třídě či třídách. Zde bylo třeba ošetřit, aby se zbytečně nevypisovaly některé elementy, které v sobě již neobsahovaly další detaily. Dále bylo potřeba zajistit, aby se nevypisovaly zděděné privátní členy. Posledním typem byl výpis `XPath` struktury, který však bylo nutné převést zpět do XML objektu a správně naformátovat odsazení.

5 Výpis konfliktů

Rozhodl jsem se implementovat i bonusový úkol CFL, jehož cílem bylo vypsat konfliktní členy. Pro implementaci tohoto rozšíření bylo nutné přidat nový přepínač `-conflicts`. Dále bylo nutné omezit chybová hlášení při nalezení konfliktu při zadání tohoto přepínače. Také bylo třeba konfliktní typy správně identifikovat, o což se stará metoda `CheckConflict()`. Největší pozornost byla věnována implementaci XML výstupu s konfliktními členy. Při tomto výpisu bylo potřební zjistit, ve kterých třídách se konfliktní typy nacházejí, a jejich správný výpis, který generuje metoda `generateB()`.