Dokumentace úlohy cls: C++ Classes v PHP 5 do IPP 2016/2017
Jméno a Příjmení: Štěpán Vích
Login: xvichs00

Struktura skriptu

Skript se skládá z celkem 5 souborů. Hlavním souborem je soubor cls.php, který všechny ostatní soubory povinně importuje. Ve všech souborech se nachází třídy pro tvorbu objektů s výjimkou souboru cls.php, který řídí vytváření objektů a hlavní komunikaci mezi nimi. Pro čtení vstupních parametrů je použita knihovna getopt.

Lexikální analýza

Probíhá v souboru LexAnalyzator.php. Spouští se jako první ze souboru cls.php, vytvořením instance třídy LexAnalyzator, které do konstruktoru předáme adresu vstupního souboru. Pokud vstupní soubor není určen nepředáváme konstruktoru žádnou hodnotu. Objekt sám určí, že má číst ze standardního vstupu. Jakmile ověří, že soubor existuje, pokusí se ho otevřít a číst jednotlivé lexémy v něm obsažené. Čtení jednotlivých tokenů, řídí syntaktický analyzátor, kterému je při konstrukci objektu předán jako parametr lexikální analyzátor. Tokeny se vyvolávají postupně pomocí veřejné funkce next_token, objektu lexikálního analyzátoru. Jakmile dosáhneme konce souboru vrací tato funkce hodnotu false. Tokeny, které vrací funkce nextToken, představují instance třídy Token. Tyto instance uchovávají informace, které budou později využity při syntaktické analýze, především zda se jedná o klíčové slovo, identifikátor, operátor, nebo datový typ. U složených datových typů (např. long long int) vrací tokeny postupně po jednotlivých slovech. O složení datového typu, se pak stará až syntaktický analyzátor. Princip lexikálního analyzátoru je založen na konečném stavovém automatu sestávajícího pouze ze tří stavů (bílé znaky, identifikátor, operátor). Při načtení neznámého znaku, nastane čtvrtý, skrytý stav, vyvolávající lexikální chybu a ukončení programu s návratovým kódem 4.

Syntaktická analýza

Probíhá v souboru SyntaxAnalyzator.php. Spouští se ze souboru cls.php, vytvořením instance třídy SyntaxAnalyzator, které do konstruktoru předáme objekt lexikálního analyzátoru. Ten s tímto objektem pracuje tak, že vyvolává jednotlivé tokeny, při syntaktickém sestupu shora dolů. Provádí vždy pouze jeden průchod souborem, jelikož nemohou nastat dopředné definice tříd a sémantika zadaného parsovaného souboru není příliš složitá. Syntaktický sestup spouštíme veřejnou funkcí sestup, objektu syntaktického analyzátoru, která vrací list tříd představující již zpracované třídy nad kterými proběhly některé sémantické kontroly. Při sestupu shora dolů komunikuje syntaktický analyzátor s instancemi tříd ze souboru Model.php, kterými postupně naplňuje seznam tříd. Tyto objekty, reprezentují parsované třídy a jejich metody a atributy, při jejichž vzniku probíhají počáteční sémantické kontroly, jako je kontrola názvu třídy, který je již zabraný, kontrola konfliktu jmen ve třídě, kontrola existující bázové třídy při dědičnosti, kontrola správnosti datových typů a další. Při průchodu pravidlem popisujícím dědičnost mezi třídami, dojde zjednodušeně k tomu, že všechny atributy a metody, uložené v bázové třídě se naklonují (pomocí klíčového slova clone) do podtřídy. Ta se tak již nemusí téměř nikdy dívat do nadřazené třídy a získávat z ní potřebné informace.

Sémantická analýza

Jak jsme popsali výše, pravidla popisující potřebnou sémantiku jsou popsány a implementovány v souboru Model.php. K drtivé většině kontrol dochází během syntaktického sestupu. Na konci sestupu je vrácen list tříd, který se poté podrobí dvěma posledním sémantickým kontrolám. První kontrola zjistí, zda list neobsahuje přetížené metody, se stejnými argumenty. Tyto metody by totiž přetíženy nebyly, nýbrž by se jednalo o sémantickou chybu. Poslední kontrola zkontroluje výskyt konfliktu číslo 21. Tento konflikt nenastává ani u klasického překladače C++, ale až za běhu programu, při přístupu k proměnné nebo metodě, která je zděděna z více tříd, a má stejný název. Za běhu programu by pak kompilátor nevěděl, kterou vlastnost si zvolit.

Výpis informací

Jakmile provedeme syntaktický sestup a potřebné sémantické kontroly můžeme již list tříd vytisknout do požadovaného tvaru a formátu. List tříd se předává v souboru cls.php do konstruktoru objektu implementující rozhraní View, uložené v souboru View.php. Podle toho o jaký výstup uživatel požádá, takovou vybereme třídu objektu. V základní implementaci se nachází třída DetailView a třída PrettyXMLView. Obě dvě implementují rozhraní View, proto je snadné vytvářet další pohledy na data. Rozhraní definuje dvě veřejné metody, metodu printView a metodu buildView. Obě implementované třídy využívají k vybudování XML výstupu, knihovnu XMLWritter, která nabízí jednoduché rozhraní a možnost formátovat výstup vlastím řetězcem pro odsazení. Za

zmínku stojí, že třída DetailView buduje XML objekt procházením tříd v listu tříd a výpisem všech jejich atributů a metod, s výjimkou zděděných členů, které byli v rodičovské třídě privátní. Tito členové mohou stale vyvolat konflikt 21, nebo způsobit že třída bude abstraktní.