Implementační dokumentace k 2. úloze do IPP 2019/2020
Jméno a příjmení: Jan Lorenc
Login: xloren15

interpret.py

Skript přijímá na vstupu xml reprezentaci IPPcode20 programu s jeho standardním vstupem a výsledek interpretace tiskne na standardní výstup. Je psán objektově s hlavní třídou `Interpret`, která spustí vlastní chod programu metodou `Run()`, sloužící tedy jako `main()` u jiných programovacích jazyků. Jednotlivé části interpretace obsluhují samostatné třídy uvedené dále.

`Interpret` je ústřední třídou řídící celý chod interpretace. Není však příliš obsáhlá, neboť využívá ostatní třídy. V již zmíněné metodě `Run()` dochází nejprve ke kontrole a načtení argumentů skriptu třídou `Args`, konverzi přijatého xml souboru za pomoci modulu `xml.etree.ElementTree` a následné kontrole kořenového elementu `<program>` třídní metodou `CheckRootElmt(cls, root)`. Poté proběhnou dva průchody synovskými elementy kořene. V prvním se vyhledají všechna návěští IPPcode20 programu a při té příležitosti rovnou kontroluji, zda se jedná pouze o elementy `<instruction>` a zda mají správné pořadí. V druhém průchodu se již jednotlivé instrukce vykonávají třídou `Decoder`. Na závěr se zavolá výpis statistických informací pro rozšíření STATI.

Toto rozšíření je implementováno třídou `Stats`. Ta uchovává informace o celkovém počtu vykonaných instrukcí, aktuálním počtu inicializovaných proměnných s pamatováním si maxima a údaje o pořadí vypsání do výstupního souboru. Metoda `Out()` pro výpis je volaná vždy a tedy ověření toho, jestli byly statistiky požadovány v argumentech, se kontroluje až zde a dle toho se do souboru zapíše či nikoliv.

Třída `Args` se stará a kontrolu formátu a kombinací vstupních argumentů skriptu. Jiné argumenty než zadané se nesmí vyskytovat a každý pouze jednou. Výjimku tvoří `--vars` a `--insts` pro rozšíření STATI. Dále načte vstupy ze zadaných souborů, případně standardního vstupu.

Užitečnou třídou se ukázala být `Error` implementující jedinou statickou metodu `Error(errCode, msg)`. Vzhledem k nutnosti ověřování chyb v téměř každé třídě je mít jednu ústřední implementaci chybového ukončení programu výhodné a dědí z ní proto většina tříd.

Nejvíce práce v celé aplikaci provádí třída `Decoder`. Obsahuje velké množství metod nejen na kontrolu správných typů a hodnot instrukčních operandů, ale i samotných instrukcí. V hlavní metodě `Decode(cls, instruction)` přijímá xml element pro instrukci přímo z dekódovacího průchodu zmíněného v popisu metody `Run()` třídy `Interpret`. Tento element zkontroluje, identifikuje, o kterou instrukci se jedná, dle toho ověří správné typy parametrů a na závěr zavolá konkrétní statickou metodu třídy `Executor` implementující provedení dané instrukce. Tato třída `Executor` je de facto jen obal pro statické metody, kde každá reprezentuje jednu instrukci.

Dále v programu využívám několik pomocných tříd. `Variable` a `Constant` pro reprezentaci proměnné a konstanty, které obsahují jen inicializátor svých instančních proměnných. Třída `Const` zahrnuje programové konstantní hodnoty, ku příkladu jména typů nebo rámců. Poslední je `State` seskupující globální proměnné symbolizující aktuální stav překladu, jako třeba celkový počet instrukcí, číslo aktuálně prováděné instrukce, definované proměnné a návěští, zásobník dat, rámců či volání funkcí.

test.php

Tento program slouží k testování skriptů parse.php a interpret.py. Pracuje s testy ve formátu 4 souborů a to test.src, který buď obsahuje zdrojový kód IPPcode20 v případě kompletního testu či testu samotného parse.php, nebo XML reprezentaci pro případ testování samotného interpret.py. Dále pak soubor file.rc obsahující očekávaný chybový kód testu, file.out s očekávaným výstupem testu a file.in sloužící pouze jako standardní vstup pro interpret.py. Pokud některý soubor chybí, je automaticky vytvořen dle zadání.

V základním nastavení provede test celé interpretace, tedy vezme soubor s příponou .src obsahující zdrojový kód v programu IPPcode20 a jeho obsah pošle skriptu parse.php na standardní vstup. Pokud dojde k chybě, zkontroluje se obsah souboru .rc a souhlasí-li chybové kódy, test končí úspěšně. V opačném případě je XML výstup uložen v dočasném souboru, poslán v parametru skriptu interpret.py, který dokončí překlad, a jeho výstup opět v dočasném souboru porovnán s obsahem .out programem diff.

V případě přepínačů --parse-only a --int-only provede jen první či druhou část výše zmíněného postupu s tím rozdílem, že u --parse-only výstup neporovnává programem diff, nýbrž jexamxml.jar, který slouží právě k porovnávání XML souborů.

V implementaci jsem se držel objektově orientovaného přístupu opět s jednou třídou jako řídící a dalšími pomocnými. Mimo to jsem implementoval abstraktní třídu `AbstractError` implementující chybovou metodu, ze které většina zdejších tříd dědí, neboť chybové ukončení programu zde bylo velmi často používáno.

Hlavní a největší třídou je `Tester`. Obsahuje jak instance pomocných tříd, tak i atributy pro statistické výpočty úspěšnosti testů. Řízení celého skriptu se odehrává v metodě `Run()`, kde dojde k identifikaci typu testu, iteraci v zadaných složkách s testy, které se spouští, a nakonec k vypsání výsledné HTML stránky na standardní výstup. Načtení složek probíhá v metodě `LoadDirs()`, která z argumentů zjistí, ve kterých složkách se testy nachází. Jedná se tedy o jednu složku v základním chování a vícero při parametru –testfile. Samostatné testy z –testfile se uloží bokem a spustí se mimo hlavní smyčku. Samotné spouštění testů pak probíhá v metodě `RunTests($tests, $run, $testGroup)`, která dostane pole s cestami k testům, informaci, jaký druh testu spustit, a skupinu, do které testy patří (většinou se jedná o jejich složku). Dále se testy dle parametrů předávají programům parse.php, interpret.py, jexamxml.jar a diff, jejichž výstupy a návratové hodnoty se dále kontrolují a udávají výsledky testů.

Další třídou je opět `Args` starající se o načtení programových argumentů, kontrolu jejich formátu a kombinací. Správné kombinace zde nebudu ze zadání opakovat, nicméně zmíním implementaci rozšíření FILES, která přidává dva možné argumenty navíc a sice filtraci testů regulárním výrazem a možnost přidat sadu složek / individuálních testů v samostatném souboru. Tato možnost --testfile=file nesmí být použita s --directory=path.

Třída `TestDir` reprezentuje zadanou testovací složku. Jako atributy má svůj název, pole s názvy testů, které obsahuje, informaci o tom, jestli ony testy má v sobě hledat rekurzivně, a filtr typu regulární výraz oněch hledaných testů. Obsahuje pouze jednu významnější metodu, kterou je `FindTests()`, která právě na základě filtru a boolean hodnoty o rekurzivitě vyhledá soubory s příponou .src, vytvoří chybějící .in, .out .rc dle zadání a vyhledaný test uloží do pole svých testů.

Poslední třídou je `HtmlWriter`, která je velmi triviální. Jejím úkolem je vytvořit výslednou HTML stránku, nicméně šablonu pro ni má již obsaženou v metodě `GenerateHtml($type, $result, $recursive)`, která do této šablony jen doplní informace o testech a jejich výsledky. Na standardní výstup stránku vypíše metodou `Out()`.