Implementační dokumentace k 2. úloze do IPP 2018/2019

Jméno a příjmení: Martin Macháček
Login: xmacha73

1 Úvod & Úloha skriptů

První částí úlohy bylo vytvořit skript napsaný v jazyce Python (3.6) – interpret.py, který přijímá zdrojový soubor vygenerovaný skriptem předešlé úlohy, nebo napsaný již v XML podobě v jazyce IPPcode19 a podle mé implementace zkontroluje správnost kódu (ať už syntaktickou či sémantickou), vykoná zadané instrukce a popřípadě vypíše uživateli informace na obrazovku.

Druhou částí této úlohy byl skript napsaný v jazyce PHP (7.3) – `test.php`, který testuje samotné skripty `parse.php` z první části a interpret.py z části druhé (tyto názvy nejsou definitivní, můžou se měnit). Tato kontrola probíhá na základě spouštění skriptu `interpret.py` / `parse.php` pro každý dostupný test a poté vygenerováním konečného sumarizačního souboru napsaném ve značkovacím jazyce HTML5 doplněného o soubor stylů pro větší přehlednost pro uživatele.

V případě nalezení chyby se skripty ukončí s návratovou hodnotou specifickou pro danou chybu.

2 Skript interpret.py

Při implementace tohoto skriptu je využíváno několik knihoven: `argparse` je použito na zpracovávání argumentů od uživatele, `ElementTree` na lexikální a částečně i syntaktické zpracování XML vstupu, a nakonec knihovnu `re`, využitou při kontrolách syntaktické správnosti pomocí regulárních výrazů (převážně funkce `re.match`).

Skript si nejdříve celý vstupní soubor zpracuje do slovníku, zkontroluje jeho správnost, poté se tento slovník převede na přehledné pole, předá se do samotné intepretace, která si každou instrukci řadí podle názvu a uživatelem definovaným pořadníkem `order`, a pomocí cyklu prochází každou instrukci a interpretuje ji na základě vytvořených funkcí.

Pro interpretace zdrojového souboru bylo nutné pracovat s datovým zásobníkem a lokálním, dočasným, globálním rámcem. V mém případě je datový zásobník, lokální a dočasný rámec používán jako prosté pole, kde lokální rámec ukazuje na svůj první rámec, ovšem globální rámec je využíván jako slovník.

Velice zajímavým problémem bylo vytvořit interpret tak, aby se co nejméně kódu opakovalo. Tento problém v mém případě vyřešila kombinace funkcí `findVariable`, `getTypeAndValue` a funkce `updateVar`, které jsou využity napříč téměř všemi funkcemi pro interpretaci. Dalším zajímavým problémem byla funkce `read` a její vstupní data, které mohly být zadány i prázdné. Tato situace vracela ve funkci input výjimku, tudíž ji bylo potřeba ošetřit bloky `try`, `except`. V této funkci se také vyskytl problém načítání dat ze souborů, kde si skript měl pamatovat, které řádky již zpracoval. Funkce `readline` tento problém zcela vyřešila, s tím, že byla potřeba najít funkci, která bude ořezávat znaky nového řádku ze vstupních dat (`rstrip`).

Nemalým problémem byla implementace vracení chybových hodnot, které i přesto, že jsou uvedené v zadání, se pro mě přibližně v polovině projektu staly velice nepříjemnou překážkou, kvůli které jsem byl nucen změnit celý návrh skriptu. Jsem za tuhle situaci ale nakonec rád, protože je nyní skript mnohem přehlednější a je více využíván princip DRY.

Na otestování správnosti tohoto skriptu sloužil již zmíněný testovací skript `test.php`

3 Testovací skript test.php

Skript sloužící k testování skriptů `interpret.py` a `parse.php`, nejdříve načte adresu složky, ve které jsou testovací soubory, vyhledá testovaný skript, definuje si pole povolených parametrů rozšířené o `--better-font` a zkontroluje správnost a formát parametrů zadaných od uživatele. Parametry, které mají uvedené rovnítko, jsou poté rozděleny podle toho, co je před rovnítkem, a co za ním, a zkontrolovány. Pokud byl zadán parametr `--recursive`, je využit objekt třídy `RecursiveIteratorIterator`, který nám zprostředkuje rekurzivní prohledávání adresářů, ve kterých můžou být testovací soubory, jinak se testovací soubory hledají pouze v zadaném, či výchozím adresáři.

Skript si vyhledá pouze soubory s příponou `src`, a pro každý tento soubor kontroluje, jestli existuje soubor s příponou `in`, `out`, a `rc`, v případě neexistence takovýchto souborů si je skript dokáže vygenerovat prázdné.

Testování samotných skriptů je provedeno funkcí `test_interpret_or_parser`, která nejdříve zjistí, který skript se má testovat, pomocí funkce `exec` spustí testovaný skript a podle návratových hodnot této funkce je porovnává s referenčními (správnými) návratovými hodnotami. Pokud jsou návratové hodnoty shodné, a jsou 0, skript provede také porovnání výstupních souborů kombinací funkce `shell_exec` a příkazu `diff`. Při testování skriptů v kombinaci si skript nejprve provede test pro skript `parse.php`, kterým se vygeneruje výstup potřebný pro otestování interpretace a poté spustí již zmiňovanou funkci `test_interpret_or_parser` s parametrem určující test skriptu `interpret.py`.

Skript si poté uloží výsledek a předá ho samotnému generování HTML5 výstupu.

Výstup skriptu, již zmíněný soubor HTML5 doplněný o soubor stylů CSS je generován příkazem `echo`. Zadání úlohy nedovoluje používat externí odkazy na fonty, či ikony, ale po dotazu na fóru v předmětu IPP mi bylo doporučeno, že v případě, kdy by uživatel chtěl získat výstup s lepším fontem, může být skript rozšířený o argument `--better-font`. Skript tedy obsahuje takovýto argument navíc. Ikony tohoto souboru nejsou nijak importované z externího zdroje, ale jsou vytvořené čistě v CSS, pomocí `:before` a `:after` selektorů.

Responzivita tohoto výstupu bohužel není provedena z důvodu nedostatku času na ostatní projekty, ovšem může být implementována pomocí `@media` dotazů.