Implementační dokumentace k 2. úloze do IPP 2018/2019
Jméno a příjmení: Michal Plšek
Login: xplsek03

Dokumentace k 2. úloze: interpret.py

Skript interpretu nejdříve zkontroluje argumenty programu a jejich kombinace. Vstupní XML je načteno pomocí xml.etree. Analýza souboru nejprve provede seřazení operací v XML podle jejich aributu *order* a taky případně seřadí argumenty operačního kódu, pokud jsou přeházené. Seřazení probíhá s využitím lambda funkce. Skript také určí zda jsou správně uvedeny atributy v kořenovém XML tagu *program*, pokud ne vrátí chybu.
Skript je rozdělen do více souborů: hlavní spouštěcí soubor intepret.py, soubor s globálním nastavením skriptu (některé globální proměnné, zkompilované regexy, definice tříd) a soubor s funkcemi. Dokumentačně zajímavé části ze souboru s nastavením a proměnnými jsou: slovník návěští, globální rámec proměnných *GF* a dočasný rámec *TF*, na začátku nastaven na hodnotu None.
Interpret provádí dva průchody. V prvním průchodu provede syntaktickou kontrolu XML souboru (kvůli tomu, že vstupní soubor nemusí být vygenerovaný pomocí skriptu parse.php, tudíž může být podvržený). Při kontrole vstupního XML by bylo rovněž správné použít knihovny jako python3 *defusedxml*, mimo jiné kvůli prevenci útoků typu Billion Laughs apod. - merlin ale tyto knihovny nemá nainstalované.
Součástí prvního průběhu je také vytvoření slovníku návěští kvůli pozdějšímu použití (s kontrolou případné redefinice) a uložení všech globálních proměnných do rámce *GF* (a kontrola případné redefinice). Syntaktickou kontrolu popisovat podrobně nebudu, jedná se opět o konečně-stavový automat s použitím regulárních výrazů.
Druhý průchod je zajímavější. Nejprve dojde k vytvoření zásobníku rámců, zásobníku volání a datového zásobníku. Vlastní průchod nestrukturovaným programem je zajištěn cyklem while který je na začátku nastaven tak aby proběhl pouze jednou. V případě že má dojít ke skoku na návěští (ne za předpokladu že návěští je na úplném konci programu) se cyklus uzamkne na další provedení, je nalezeno návěští ze slovníku návěští a provede se skok na instrukci která následuje za návěštím. Samotné zpracování instrukce je znovu řešeno konečně-statvovým automatem, v závislosti na operačním kódu proběhne načtení argumentů instrukce a jejich sémantická validace (existence proměnné v rámci, existence rámce, správnost typu proměnné, existence položky na datovém zásobníku atd.).
Objektovou strukturu programu tvoří jednoduchá třída *FrameStack* použitá pro datový zásobník, zásobník volání a zásobník rámců, dále pak třída *Frame* pro instance *GF/TF* (LF je řešen pouze jako odkaz na první prvek zásobníku rámců), třída *Variable* pro proměnné uložené do rámce a třída *StackItem* pro symboly (konstanty a proměnné) uložené do datového zásobníku. Třídy jsou dost triviální a jsou dokumentovány v souboru *variables.py*.
Detaily k několika jednotlivým zpracovávaný instrukcím: instrukce *READ* může brát data buďto ze souboru nebo ze sys.stdin, v případě souboru je soubor na začátku programu otevřen, pomocí funkce readline() jsou z něj odstraněny konce řádků a je uložen do seznamu: se seznamu se posléze načítají položky (přes inkrementovaný čítač, kdykoliv dojde k operaci READ); v případě sys.stdin je použita funkce input(). V případě instrukce *CREATEFRAME* dojde k reinicializaci objektu *TF*, objekt byl na začátku programu nastaven na None. U instrukce STRLEN se pomocí re.findall shromáždí všechny escape sekvence, jejich počet se vynásobí třemi a odečte se od délky původního řetězce s escape sekvencemi.
Všude kde to jde jsou použity výjimky. Soubor *variables.py* obsahuje definice výjimek, které podle zadání v případě chyby vypíšou na standartní chybový výstup hlášení a ukončí program s příslušným návratovým kódem.

Dokumentace k 2. úloze: test.php

Skript nejprve nastaví výchozí hodnoty pro zpracování: výchozí soubory parseru a interpretu a výchozí cestu k testům *$path*. Poté zkontroluje argumenty programu: pomocí funkce *getopt* uloží argumenty do pole a poté testuje nastavení každého z nich a případné kolize mezi nimi (kolizní argumenty jako *int-only* a *parse-script* apod.). Cesta může být zadána se znakem / na konci nebo bez něj, je to vyřešeno pomocí funkce *rtrim* a následném připojení znaku /. Pokud zadaná cesta není adresářem, skript se s chybovým hlášením ukončí.
Poté dojde k nastavení módu testovacího skriptu (proměnná *$mode*, 1 = testuj pouze parser, 2 = testuj pouze interpret, 3 = testuj oba). Otestuje se také jestli existují soubory těchto skriptů.
Poté je vygenerován začátek HTML kódu, hlavička s inline css a otevírací html tag tabulky s přehledem výsledků testů. Jsou vytvořeny dva dočasné soubory, *temp_output* pro dočasný výsledek jednotlivých testů a *temp_both* pro výsledek testování obou skriptů zaráz (mód 3). Pokud soubory nejdou vytvořit, program vrátí chybový kód. Nyní je vygenerováno pole polí *$folders*, jehož vnitřní pole obsahují jednotlivé položky v konkrétním adresáři a tento konkrétní adresář je uložen jako klíč tohoto vnitřního pole. Položky vnitřního pole jsou uloženy jako názvy (ať už složek nebo souborů) a budou dále zpracovány. Pokud byla argumentem programu nastavena rekurze, pole polí je naplněno názvy pomocí objektu *$RecursiveIteratorIterator*, který iteruje přes kontejner objektů *$RecursiveDirectoryIterator* a přitom přeskakuje soubory .. a . . V případě že rekurze nastavena není, pole je jednoduše naplněno soubory za použití *DirectoryIterator*, iterátoru, který postupně prochází pouze aktuální adresář *$path*.
Proměnná *$foundsome* je na začátku nastavena na false a má oddělovat v tabulce výsledku testů jednotlivé adresáře pro zvýšení přehlednosti.
Testování pole polí *$folders* probíhá v cyklu po jednotlivých adresářích: pokud je v adresáři (klíč vnitřního pole) nalezen testovací soubor s příponou .src, je zvýšen čítač počtu testů.
Po zahájení testování konkrétním testem ze souboru .src jsou nejprve případně dogenerovány chybějící části testu (.rc, .in, .out). Ze souboru .rc je funkcí *file_get_contents* načten návratový kód, úspěšnost testu je nastavena na výchozí 1 (neprošel). V závislosti na módu testování se spustí příkazem *exec* externí program. Při každém spouštění externího programu je standartní chybový výstup zahazován a návratové kódy jsou vraceny pomocí do proměnné pomocí argumentu funkce *exec*.
Pokud obsahuje soubor .rc konkrétního testu jinou hodnotu než 0 (tj. kód má vygenerovat chybu, proměnná *$output_override* zajistí že se u tohoto testu nebude zkoumat výstup interpretu / parseru na sys.stdin (na fóru k IPP je napsáno, že u výsledku testů se v případě chyby bude hledět pouze na návratový kód).
V případě módu 1 dojde k testování parseru na souboru se vstupním kódem v IPPcode19, porovnání návratového kódu a potom případnému porovnání výstupu pomocí skriptu jexamxml. Soubor options s nastavením tohoto skriptu byl uložen a je používán ze stejného adresáře jako testovací skript
V případě módu 2 se spustí intepret s požadovanými vstupními argumenty –source a –input, pokud návratový kód souhlasí tak dojde k porovnání vygenerovaného obsahu a souboru .out pomocí *diff*. Pokud se testují oba skripty (mód 3), tak se nejprve do dočasného souboru uloží výstup parseru (jeho vstup tvoří .src soubor), tento dočasný výstup je poslán na vstup interpretu a tento druhý dočasný výstup se porovná pomocí *diff* za předpokladu, že souhlasí návratový kód interpretu s obsah souboru .rc .
Zda test prošel se rozhoduje podle návratového kódu posledního spouštěného programu. Pokud je 0, je zvýšen čítač úspěšných testů, pokud je jiný je zvýšen čítač neúspěšných testů a do záznamu v tabulce přidán popis důvodu proč test neprošel (očekávaný/obdržený návratový kód nebo chyba porovnání výstupu). V obou případech je do tabulky přidán záznam o provedení testu a obsah dočasných souborů je smazán.
Po otestování každého souboru s příponou .src je dogenerován zbytek HTMl kódu s přehledem počtu testů. Skript následně zavře dočasné soubory a spustí externí program, který je odstraní.