Implementační dokumentace k 2. úloze do IPP 2018/2019

Jméno a příjmení: Jakub Sadílek
Login: xsadil07

1 Úvod

Zadáním této úlohy bylo odevzdat pár skriptů. Interpret.py pro interpretaci kódu IPPcode19 ve formátu XML, který je výstupem skriptu z první úlohy a skript test.php, který slouží pro automatické testování obou předchozích skriptů. V případě chyby jsou skripty ukončeny odpovídající návratovou hodnotou podle zadání odprovázeny chybovou hláškou.

2 Interpret

Skript byl napsán v jazyce Python 3.6. Implementace nebyla příliš složitá, pokud byla správně provedena dekompozice problému. Při řešení jsem postupoval následovně.

2.1 Vstupní parametry

Skript podporuje celkem tři parametry, jejichž kombinace ovlivňuje průběh celého skriptu. Pro jejich zpracování byla použita třída `ArgumentParser()`. Vstupní soubory byly načteny do proměnné, pokud byly zadány, pokud nějaký chyběl, tak se načetl ze standardního vstupu. Rozdíl byl v tom, že zdrojový kód byl načten celý ihned, vstupní data se četla v případě potřeby funkcí `read()`.

2.2 Zdrojový kód IPPcode19

Poté co byl celý kód načten ve formátu XML, byl naparsován pomocí `xml.etree.ElementTree` a následně validována hlavička programu. Pak se celý program seřadil podle pořadí instrukcí a také jejich argumenty. Během řazení byly zároveň kontrolovány povinné a přebytečné atributy.

Pro usnadnění práce byla vytvořena třída reprezentující instrukci, která obsahuje operační kód a seznam argumentů. Argumenty jsou reprezentovány slovníkem, který obsahuje typ a hodnotu. Během parsování instrukcí do třídy je zároveň kontrolována jejich syntaxe tzn. ověření známého operačního kódu, formát konstant a proměnných. Zkrátka vše, co kontroluje skript z první úlohy, protože musíme očekávat, že vstup může zadat někdo jiný. Zároveň jsou v této fázi konvertovány escape sekvence v řetězci na znak, který reprezentují a načtení návěští, která jsou v kódu definována do pomocné struktury. U návěští si poznamenáme jeho jméno a pozici v kódu. Celý proces je implementován jako konečný automat. Výsledkem je seznam instrukcí, který reprezentuje celý kód.

2.3 Interpretace

Celá interpretace je opět realizována pomocí konečného automatu. Celý proces probíhá ve `while` cyklu, dokud se neprovede poslední instrukce. Třída pro interpretaci vrací hodnotu, která udává pozici instrukce v seznamu instrukcí, který jsme si vytvořili. Implicitně je tato hodnota inkrementována, ale může být i vynucena. Tohle řešení jsem zvolil kvůli případným skokům v kódu. Ve třídě je podle operačního kódu zvolen kód, který simuluje danou instrukci. Pro uchování dat při interpretaci třída obsahuje zásobník rámců realizovaný seznamem, seznam již definovaných návěští, zásobník volání, objekt reprezentující aktuálně prováděnou instrukci, počítadlo instrukcí, pozici v kódu a indikátory aktivních rámců. Každý rámec je reprezentován seznamem proměnných, které obsahuje.

2.4 Objektový přístup

Pro zachování určité abstrakce v kódu byl implementovány různé třídy.

- Třída reprezentující proměnnou: obsahuje jméno proměnné, hodnotu a typ. Je použita při samotné interpretaci v jednotlivých rámcích.
- Třída pro interpretaci
- Třída pro instrukci
- Třída pro práci s XML

3 Test

Skript byl napsán v jazyce PHP 7.3. Implementace byla poměrně snadná i zábavná, jelikož při větších projektech si sám pro sebe občas napíšu podobný skript, který mi pomůže najít chyby, které se postupně objeví a urychlí i samotné testování. Při řešení jsem postupoval následovně.

3.1 Vstupní parametry

Jelikož skript podporuje poměrně spoustu přepínačů, zvolil jsem funkci `getopt()`, která mi práci s nimi velmi usnadní. Po zpracování argumentů zkontroluji jejich počet a kolize.

3.2 Proces testování

Podle zadaného přepínače rozhodneme, který skript otestujeme. Celý proces se výrazně mezi nimi neliší a probíhá následovně. Jako první najdeme daný skript a testy, a ověříme přístupová práva. Hledání testů nám ulehčila vestavěná funkce `glob()`. Při rekurzivním hledání postupujeme stylem BFS tzn. uložíme zadaný adresář do seznamu adresářů a vyhledáme testy v něm a všechny podadresáře, které obsahuje si zase uložíme. Takto postupujeme s každým uloženým adresářem a ve výsledku získáme všechny testy ze všech podadresářů.

Když máme všechny „source“ soubory testů, tak ostatní přidružené soubory najdeme jednoduchou záměnou přípony. Pokud neexistují tak je vytvoříme s implicitní hodnotou podle zadání. Dále si vytvoříme dočasné soubory, do kterých si uložíme výstup skriptu, který porovnáme s referenčním. Opět u všech souborů ověříme přístupová práva včetně práv pro zápis u dočasného souboru. Porovnání je provedeno shell utilitou `diff` v případě interpretu, u skriptu parse.php je XML porovnání komplikovanější, protože umožňuje různé zápisy, které jsou funkčně ekvivalentní. K tomu slouží externí program jexamxml.jar, který je napsán v Javě. Po otestování dočasný soubor smažeme.

3.3 HTML report

Pro shrnutí výsledků je zadáno použít webové rozhraní, proto jsem si vytvořil třídu, která tento report bude reprezentovat. Celou stránku jsem napsal pomocí html a css ve stylu „dark mode“. Stránka obsahuje celkový počet prošlých a neprošlých testů a výpis každého testu včetně popisu chyby, pokud test neprošel. Třída se vytvoří na začátku běhu skriptu a obsahuje metody pro přidání výsledku testu, kde se výsledek přidá do tabulky testů a výpis celého html dokumentu, který se prování na konci. Počítadlo testů je realizováno pomocí globálních proměnných.