Implementačná dokumentácia k 2. úlohe do IPP 2020/2021

Meno a priezvisko: Jakub Bartko
Login: xbartk07

1 Interprét

Interprét popísaný v tejto dokumentácií napísaný v jazyku Python 3.8 slúži na interpretáciu XML reprezentácie kódu `IPPcode21`.

1.1 Činnosť

Činnosť skriptu `interpret.py` je možné rozdeliť na spracovanie argumentov príkazového riadka, načítanie XML reprezentácie zdrojového kódu, lexikálnu a syntaktickú analýzu inštrukcií a argumentov, ich vykonávanie, generovanie výstupov a zbieranie štatistík o týchto inštrukciách.

Na základe argumentov príkazového riadka sa získa XML reprezentácia, ktorá je prevedená na stromovú štruktúru s využitím modulu `xml.etree.ElementTree`. Po kontrole koreňového prvku sú inštrukcie zoradené podľa atribútu `order` a skontroľujú sa ich tagy, atribúty a počet a formát argumentov.

V rámci tejto kontroly sa predom definujú cieľové návestia a inštrukcie `LABEL` sú ďalej ignorované.

Ďalej sú iterované objekty jednotlivých inštrukcií. Podľa ich operačného kódu sa zavolá príslušná obslužná funkcia, získajú a skontrolujú sa argumenty (v stromovej štruktúre potomkovia objektu inštrukcie) a vykoná sa vnútorná funkcionalita danej inštrukcie.

1.2 Štruktúra kódu a implementácia

Funkcie na získavanie a kontrolu argumentov, obsluhu inštrukcií a riadenie toku programu a pomocné štruktúry využívané na sémantickú analýzu a zbieranie štatistík sú súčasťou triedy `Interpret`. Činnosť metód tejto triedy je možné rozdeliť na niekoľko kategórií.

Argumenty inštrukcií sú na základe daného objektu inštrukcie a poradia argumentu získavané funkciou `get_arg`. Podľa definovaného typu argumentu, t. j. `var`, `symb`, `label` alebo `type`, sa skontrolujú jeho atribúty a text a vráti sa spracovaný argument nasledovne:

- `var` - názov rámca a premennej,
- `symb` - typ a hodnota premennej alebo konštanty,
- `label` - názov návestia,
- `type` - objekt enumerátora `Type`.

Funkcie na **prácu s premennými** využívajú asociatívny zoznam obsahujúci globálny a dočasný rámec a zásobník lokálnych rámcov. Každá premenná je reprezentovaná objektom triedy `Var` uchovávajúcim dvojicu: typ a hodnota. Na zistenie unikátnosti/výskytu určitej premennej sa využíva funkcia `is_unique` a na uloženie jej typu a hodnoty do príslušného rámca pod špecifikované meno funkcia `store`. Jednotlivé rámce sú teda implementované ako asociatívne zoznamy vo formáte *názov_premennej: Var(typ, hodnota)*.

Aritmetické, **relačné** a **logické operácie** využívajú spoločnú funkciu `operations`. V rámci tejto funkcie sa cieľové úložisko a hodnoty operandov získajú buď z argumentov inštrukcie, alebo v prípade koncovky „*-S*“ z dátového zásobníka (*Stack*). Následne sa podľa typu operácie (aritmetická, relačná, ...) skontrolujú typy operandov, získa sa jej výsledok využitím asociatívneho zoznamu lambda výrazov a uloží sa buď do cieľovej premennej alebo na dátový zásobník.

Operácie s **dátovým zásobníkom** pracujú nad zoznamom dvojíc (`tuple`) typov a hodnôt. V prípade chýbajúceho argumentu využijú dvojicu na vrchu zásobníka.

Do kategórie **riadenia toku programu** patrí funkcionalita na spravovanie návestí, vyhodnocovanie podmienených skokov a výber nasledujúcej inštrukcie v hlavnej iterácií. Návestia sú definované v rámci kontroly inštrukcií spomenutej v kapitole 1.1, t. j. sú uložené do asociatívneho zoznamu návestí ako *názov:* `order`. V prípade skoku na návestie sa z tohoto zoznamu získa príslušný `order` a jeho hodnota sa vráti do tela iterovania inštrukcií. Ak návratová hodnota zavolanej obslužnej funkcie chýba, iterácia pokračuje inštrukciou na nasledujúcom indexe; v opačnom prípade sa v stromovej štruktúre XML reprezentácie nájde inštrukcia s príslušnou hodnotou argumentu `order` a iterácia pokračuje inštrukciou na indexe, ktorý nájdenej inštrukcií nasleduje.

Popisy kategórií inštrukcií neuvedené v tejto kapitole boli vynechané z dôvodu triviálnosti ich riešenia a priamočiarosti implementácie. Ich obsluhu je možné zhrnúť na: získanie cieľovej premennej, získanie hodnôt argumentov a kontrolu ich typov a uloženie výsledku, ktorého získanie sa zvyčajne zmestilo do samotnej operácie ukladania.

1.3 Rozšírenia

Rozšírenie `FLOAT` je implementované ako prídavná funkcionalita obslužných funkcií pre inštrukcie aritmetických, relačných a logických operácií, inštrukcie načítavania, výpisu atď. Okrem toho sú v rámci toho rozšírenia implementované funkcie na prevod medzi typmi `float` a `int` (a ich zásobníkové verzie).

Ako už bolo uvedené v predchádzajúcej kapitole 1.2, operandy inštrukcií rozšírenia `STACK` sú získavané z dátového zásobníku, na ktorý sa taktiež ukladajú ich výsledky. V ostatných ohľadoch sú interpretované analogicky so svo-

jimi nezásobníkovými verziami.

Argumenty príkazovej riadky rozšírenia `STATI` sú spracované spolu so základnými funkciou `get_args` a predané inštancií triedy `Interpret`. Na sledovanie počtu vykonaných inštrukcií sa využíva vnútorné počítadlo tejto triedy, aktualizované pri zavolaní príslušnej obsluhy prostredníctvom funkcie `Interpret.run`. Na tomto mieste sa taktiež zväčší hodnota počtu zavolaní aktuálnej inštrukcie v asociatívnom zozname s formátom `order:` *počet_zavolaní*, prípadne so doň táto dvojica vloží ako prvý výskyt s hodnotou 1. Na sledovanie počtu platných inicializovaných premenných sa využívajú počítadlá aktualizované pri zápise hodnoty do premennej funkciou `Var.set` v prípade, že jej pôvodná hodnota nie je definovaná, a pri manipulácií so zásobníkom lokálnych rámcov tak, že v rámci obsluhy funkcie ... sa inicializované premenné:

- `CREATEFRAME`: zahodeného dočasného rámca **odčítajú**,
- `PUSHFRAME`: prekrytého lokálneho rámca **odčítajú**,
- `POPFRAME`: zahodeného dočasného rámca odčítajú a inicializované premenné odkrytého lokálneho rámca sa pripočítajú.

Tieto štatistiky sú na po spracovaní všetkých inštrukcií alebo po zavolaní inštrukcie `EXIT` zapísané do špecifikovaného súboru.

2 Testovací skript

Skript `test.php` slúži na testovanie činnosti skriptov `parse.php` a `interpret.py` buď samostatne, alebo ako jednotného funkčného celku.

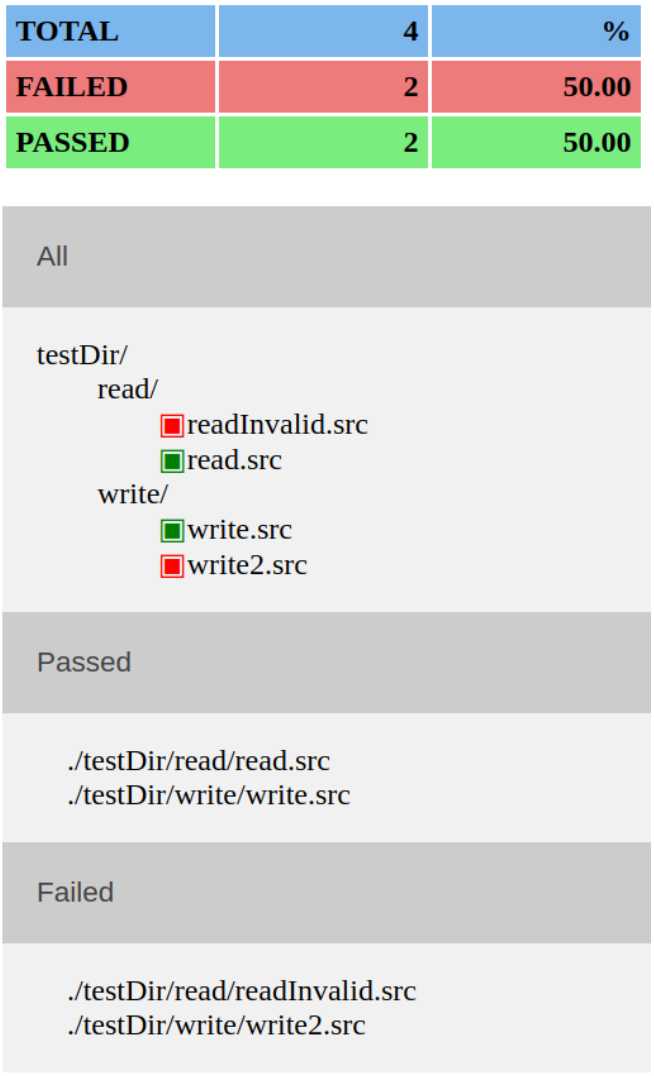
2.1 Činnosť a implementácia

Tento skript využíva dve globálne štruktúry: `Handles` – na uloženie spracovaných argumentov príkazového riadka a kontrolu špecifikovaných vstupných súborov; a štruktúru `Outputs`, ktorá sa využíva na sledovanie úspešnosti testov a zápis generovaného HTML výstupu.

Špecifikovaný súbor s testami môže byť iterovaný rekurzívne, no z hľadiska implementácia sa od nerekurzívneho prechádzania líši len v konštrukcií iterátora a formáte výstupu. Pre každý zdrojový súbor (.src) sa skontrolujú a v prípade potreby dogenerujú príslušné referenčné súbory. Ďalej sa zostaví a zavolá príkaz na základe špecifikovaného testovaného skriptu/, ktorý spustí testované skripty (alebo ich kombináciu) s príslušným vstupným súborom na vstupe, uloží výstup a získa návratovú hodnotu.

Daný test potom validuje tak, že porovná získanú a referenčnú návratovú hodnotu a v prípade úspešného ukončenia testovaného skriptu porovná aj získaný výstup s výstupom referenčným unixovým nástrojom `diff`, prípadne výstup vo formáte XML nástrojom `A7Soft JExamXML`. Na záver validácie sa upracú vytvorené dočasné súbory.

Výsledok validácie sa vráti do tela iterácie, kde sa ďalej aktualizujú počítadlá na sledovanie úspešnosti testov a pripíše sa HTML výstup vygenerovaný pre daný test.



Obr. 1: Ukážka výstupného HTML súboru

Na záver sa generovaný HTML výstup finalizuje a vypíše sa na `STDOUT`. Výstupný výpis (obr. 1) pozostáva z absolútneho a relatívneho počtu a interaktívnych zoznamov testov zoskupených podľa úspešnosti.