Implementačná dokumentácia k 2. úlohe do IPP 2020/2021
Jméno a příjmení: Kristián Královič
Login: xkralo05

Interpret.py :

Cieľom bolo vytvoriť skript, ktorý vykoná interpretáciu xml reprezentácie jazyka IPPcode21. Skript je rozdelený na množstvo pomocných funkcií pričom každá má svoju špecifickú funkciu.

1. Parsovanie argumentov

Ako prvá funkcia je volaná funkcia *argumentParse()*. Táto funkcia spracuváva zadané argumenty skriptu pomocou knihovne *argparse*. Funkcie taktiež kontroluje správnosť zadaných argumentov a vracia názov vstupného súboru a vstupného súboru . V prípade, že jeden zo súborov nebol zadaný argumentom interpreter načíta daný súbor zo štandardného vstupu.

2. Kontrola xml

Kontrola xml prebieha vo funkcii *xmlcheck(file)*, ktorá načíta obsah zdrojového súboru a vykoná základné lexikálne a syntaktické kontroly, ako napríklad kontrolu či argument je jeden zo 4 zadaných typov. Funkcia taktiež kontroluje správnosť samotnej xml reprezentácie. Funkcia vracia skontrolovaný xml strom.

3. Kontrola opcode a argumentov instrukcie

Ako ďalšou je volaná funkcia *checkOpcodeAndArguments(xmlTree)*. Táto funkcia skontroluje lexikálnu a syntaktickú správnosť opcode a argumentov ako aj samotnú existenciu opcode.

4. Vytvorenie listu instrukcii

Pre vytvorenie listu inštrukcií je volaná funkcia *makeInsList(xml)*. Táto funkcia vytvorí objekt *Instruction* a objekt *argumentu*, ktorý naplní potrebnými dátami a uloží do listu inštrukcií.

5. Vytvorenie listu náveští

Ako ďalšie sa vytvorí list náveští pomocou funkcie *checkSemOfLavelandVariable(instructionList),* ktorý je potrebný vo fáze interpretácie kódu. V tejto funkcii prebieha aj sémantická kontrola náveštia a premennej.

6. Interpretácia kódu

Samotná implementácia kódu prebieha vo funkcii *interpretCode(instructioList, ListofLabels)*. Na začiatku implementácie sú vytvorené potrebné rámce. Globálny rámec je typu dict. Lokálny rámec je implementovaný, ako pole a na začiatku implementácie je prázdny. Dočasný rámec je iplmntovaný ako dict. Celá implementácia sa odohráva vo vnútry while cyklu ktorý cyklí postupne cez všetky inštrukcie. Implementácia prebieha tak, že pre každý oppcode je vytvorená podmienka *if,* ktorá porovnáva opcode vykonávanej inštrukcie s dopredu definovanými opcode. Keď sa podarí programu vojsť do tela ifu začne vykonávanie potrebných úkonov pri implementácii. Ako prvé za vždy získa hodnota argumentov príslušnej inštrukcie pomocou funkcie *getValueFromVarAndArg(GF, LF, TF, argument)* a následne prebieha sémantická kontrola a správnosť argumentov.

Ďalej sa kontroluje kam sa má výsledná hodnota zapísať, a ak daná premenná v danom rámci neexistuje hlási program chybu. Každá kontrola má svoju vlastnú funkciu, ako napríklad *checkVar(argument)*, *checkLabel(argument)* a pod.

Test.php:

Cieľom bolo vytvoriť skript ktorý slúži na automatické testovanie skriptu *parse.php* a *interpret.py.*

1. Spracovanie argumentov

Ako prvé skript spracuváva argumenty, ktoré mu boli zadané. Spracovanie prebieha cez funkciu *getopt()*. Skript kontroluje správnosť zadaných argumentov ako aj ich nepovolené kombinácie. V prípade, že bol zadaný parameter *--direcotry* skript kontroluje existenciu zvoleného adresára. Ak parameter nebol zadaný berie skript testy z aktuálnej zložky v ktorej sa nachádza. Tak isto kontroluje aj existenciu jednotlivých skriptov v prípade, že boli zadané parametre *--parse-script* a *--int-script*.

2. Získanie súborov

Skript prechádza zadaný adresár s testovacími súbormi a ukladá mená súborov do pola mien súborov . V prípade že bol zadaný parameter *--recursive* skript prechádza zložky pomocou *RecursiveDirectoryIterator() a RecursiveIeratorIterator()* . V prípade že parameter *--recursive* nebol zadaný používa skript *scandir().* Toto všetko vykonáva funkcia *getFiles(testPath,recursion).*

3. Vykonávanie testov

Testy sú rozdelené do 3 skupín pomocou podmienky if, elseif, else. V prípade že bol zadaný parameter *--parse-only* skript prechádza všetky súbory s uložené v poli názvov súborov, ktoré boli načítané pomocou funkcie *getFiles(testPath, recursion)*. Ďalej test spúšťa parse.php a kontroluje výsedný exit code parseru s načítaným očakávaným návratovým kódom zo súboru s koncoukou *.rc*. Ak je návratový kód rovnaký ako očakávaný návratový kód porovná skript výslednú xml reprezentáciu so vzorovou xml reprezentáciou pomocou programu *jexamxml.* Rovnakým spôsobom skript kontroluje aj *interpreter.py.* Jediný rozdiel je že pre porovnanie výsledkov používa skript nástroj *diff*. V prípade že bol zadaný parameter *–int-only.* V prípade že nebol zadaný parameter *--parse-only* ani *–int-only* skript najprv spustí *parse.php* kde kontroluje len návratovú hodnotu a vysledok následne predáva do *interpret.py*. V prípade že je návratový kód interpreteru správny skript porovná výstup s očakávaným výstupom pomocou nástroja *diff.*

4. Tvorba html výstupu

Tvorba začína funkciou *generateHTMLstart()*, ktorá vygeneruje hlavičku html súboru. Ako ďalšia je volaná funkcia *generateRestHTML()*, ktorá vygenerúje údaje o celkovom počte vykonaný testov, počte testov ktoré prešli a počte testov ktoré neprešli. Ďalej je volaná funkcia *GenerateTableStart()*, ktorá vygeneruje prvý riadok tabuľky s výsledkami testov. Počas testovania je pri rozhodnutí či test prešiel alebo nie volaná funkcia *generateTable(table, filePath, parseRC, interRC, expectedEC, result),* ktorá generuje jednotlivé riadky tabuľky podľa výsledku testu a ukladá ich do premennej *$htmlTable*, ktorú následne vypíše. Ako posledná je volaná funkcia generateTableEnd, ktorá vypíše posledný riadok tabuľky.