Implementačná dokumentácia k 2. úlohe do IPP 2021/2022

Meno a priezvisko: Matej Koreň
Login: xkoren10

Časť 1. – interpret.py

Úlohou bolo vytvoriť sémantický analyzátor a interpret XML kódu v jazyku Python 3.8 , ktorý načíta vstup zo štandardného vstupu pomocou zadaných parametrov. Úspešným výstupom je správne vygenerovaný a interpretovaný kód vypísaný na štandardný výstup.

Skript po spustení najprv kontroluje správnosť zadaných parametrov pomocou knižnice `argparse`. Pri zadanom parametre `--help` vypíše na štandardný výstup nápovedu, inak vracia chybu.

Následne pomocou knižnice `xml.etree.ElementTree` načíta vstup zadaný v parametri `--source` a skontroluje základné prvky XML súboru. Pri úspechu sa pokračuje v ďalšej analýze, kde sa v súbore kontroluje a vyhľadáva názov inštrukcie, poradie a jednotlivé argumenty. V prípade správneho zápisu sa vytvoria objekty pre jednotlivé argumenty a všetky sa priradia do objektu novej inštrukcie. Tá sa na konci pripojí podľa jej poradia ako kľúča do slovníku `instruction_array`. Pri dorazení na koniec súboru sa inštrukcie zoradia podľa kľúčov a neskôr prepíšu tak, aby boli v poradí začínajúcim od 1.

V ďalšom kroku sa rýchlo skontroluje výskyt inštrukcie `LABEL` (kvôli možným skokom vpred) a prípadné náveste sa ukladajú do slovíku `label_array`. Pomocou cyklu while sa prechádzajú jednotlivé inštrukcie v slovníku a podľa operačného kódu sa postupne vykonávajú.

Pri kontrole argumentov jednotlivých inštrukcii sa používajú dve vopred definované funkcie:

`find_variable` : vracia *boolean* hodnotu podľa toho, či bola premenná už zadefinovaná

`check_type` : vracia *boolean* hodnotu podľa toho, či zadaný typ argumentu sedí s hodnotou argumentu

Interpretácia kódu potom prebieha podľa druhu argumentov inštrukcie a pracuje sa s vopred zadefinovanými premennými a návesťami. Pri generovaní kódu sa taktiež používajú vstavané funkcie python-u, ako napríklad `print()`, `chr()`, `int()`, `exit()` a pod. Pri chybách sa vracajú patričné návratové hodnoty. Inštrukcie, ktoré nie sú implementované neovplyvňujú priebeh cyklu.

Pri vytváraní interpreta som narazil na prekážky, ktoré boli hlavne kvôli časovej zložitosti ťažko prekonateľné. Vo výsledku som preto upustil od spracovania určitých inštrukcii a zameral sa hlavne na správnosť tých, ktoré implementované boli. Čiastočne som využil aj prvky z *test driven develompentu*-u, ktoré mi pomohli odhaliť ľahko prehliadnuteľné chyby. Celkovo ma ale zaujalo to, ako v skutočnosti funguje interpretácia kódu a koľko vecí počas nej treba pre správnosť kontrolovať.

Implementačná dokumentácia k 2. úlohe do IPP 2021/2022
Meno a priezvisko: Matej Koreň
Login: xkoren10

Časť 2 – test.php

Úlohou bolo vytvoriť skript v jazyku PHP 8.1, ktorý bude slúžiť pre automatické testovanie aplikácie **parse.php** a **interpret.py.** Výstupom je prehľadný súhrn výsledkov testov vo formáte HTML5 na štandardný výstup.

Skript po spustení najprv kontroluje správnosť zadaných parametrov. Pri zadanom parametre `--help` vypíše na štandardný výstup nápovedu, inak vracia chybu. Každý zadaný parameter mení hodnoty vo vopred vytvorenom objekte.

Testovanie sa začína prehľadaním zložky zadanej v parametri `--directory=path`. Ak bol zadaný aj parameter `--recursive`, zložka sa prehľadáva rekurzívne a preskakujú sa cesty "." a ".." . Pri úplnom zanorení sa spúšťa každý test jednotlivo pomocou funkcie `start_test`.

Pred spustením testu sa najprv skontroluje výskyt súborov s príponami `.src`, `.rc`, `.in` a `.out`y prípadne sa vytvoria. Medzitým sa do vopred skonštruovaného html objektu sa dopíše názov testovanej zložky. Načíta sa očakávaný návratový kód, vstupný a výstupný súbor.

Pri testovaní parse.php sa pomocou príkazu `exec()` spustí test s použitím zadaného scriptu (alebo implicitne nastaveného). Na základe návratového kódu funkcie `exec()` sa test vyhodnotí a prípadne sa dodatočne porovnajú výstupy cez JExamXML.

Rovnaký priebeh má aj testovanie interpreta, avšak na testovanie výstupu sa používa funkcia `diff()`.

Pri testovaní oboch skriptov naraz sa nepremazávajú pomocné súbory z parse.php a taktiež sa nekontroluje výstup cez JExamXML.

Vo všetkých prípadoch sa vyhodnocuje test pomocou návratových hodnôt alebo rozdieloch vo výstupe na `test_passed()` alebo `test_failed()` , ktoré zapíšu výsledky do html objektu. Po prejdení všetkých testov v adresári sa taktiež vypíšu štatistiky spúšťaných testov (počet úspešných a neúspešných testov). Funkcia `delete_files()` vymaže dočasne vytvorené súbory (pri nezadaní parametru `--noclean`)

Na záver sa dokončí konštrukcia html súboru, vypočíta sa úspešnosť testovania a funkcia RenderHTML vypíše celý vytvorený objekt na štandardný výstup.

Vytvorenie testovacieho rámcu bolo veľmi nápomocné pri konštrukcií interpreta. Prehľadný výpis v HTML je skôr estetickejší než funkčný, ale dáva zmysel aj nezainteresovaným osobám. Pri veľkom množstve testov je vhodný pre svoju prehľadnosť, ale celkovo práca s prepisovaním súborov pri jeho vytváraní bola zdĺhavá. Vedel by som si predstaviť jeho využitie aj pri vytváraní parse.php, kde by pomáhal odhaľovať chyby vo výstupe pomocou JExamXML, avšak časová tieseň by asi neumožnila jeho vytvorenie už pri prvej časti projektu.