Implementační dokumentace k 2. úloze do IPP 2020/2021
Jméno a přijmení: Dominik Horký
Login: xhorky32

Dokumentace ke skriptům ‘interpret.py’ a ‘test.php’

Skript interpret.py zkontroluje syntaktickou a lexikální správnost vstupu, kterým je XML soubor (výstup skriptu ‚parse.php’, viz 1. úloha) s instrukcemi IPPcode21 a jejich operandy. Vstup zpracuje tak, aby bylo možné zdrojový kód v IPPcode21 interpretovat a následně zadaný kód skript provede.
Druhým skriptem je test.php, který slouží k otestování funkčnosti skriptu z první úlohy (tedy parse.php) a výše zmíněného skriptu. Po provedení uživatelem zvolených testů tento skript generuje na standardní výstup HTML5 kód.

Skript ‚interpret.py‘

Nejprve se provede parsování argumentů skriptu. Zde jsem se snažil dodržet maximálně zadání, navíc jsem však vypracoval tři další možné přepínače (`--verbose`, `--recursive`, `--deparse`). Přepínač `--recursive` slouží jako zarážka při příliš velkém počtu volaných funkcí (resp. naplnění zásobníku volání, stává se při rekurzivním volání funkcí), program násilně ukončí, pokud zásobník volání obsahuje více jak 5000 volaných funkcí. Přepínač `--deparse` po kontrolách vstupu kód neinterpretuje, ale na standardní výstup vypíše původní zdrojový kód IPPcode21 a poté skript skončí (jako využití tohoto přepínače byla zamýšlena situace, kdy je uživateli dostupný XML soubor, ale zdrojový kód chybí a je potřeba ho např. nějak změnit).
Následně probíhají kontroly XML vstupu, zda je ‚well-formatted’, zda obsahuje instrukce a popř. jejich argumenty, zda instrukce mají kladné neduplicitní pořadí nebo argumenty mají pořadí v rozmezí 0-3 (žádná instrukce IPPcode21 nemá více jak 3 operandy-argumenty). V průběhu těchto kontrol již probíhá příprava na interpretaci a to tak, že se ukládají informace o:

- instrukcích, jejich pořadích (order) a nalezených argumentech (pole `PROG_INSTR`)
- argumentech, jejich příslušnosti k dané instrukci (rozlišeno pomocí order-pořadí instrukce), ukládají se jejich hodnoty a typy (pole `PROG_ARGS`)
- proměnných, jejich hodnotách a typech (pole `PROG_VARS`)
- návěštích, jejich pořadích (pole `PROG_LABELS`) a o voláních, resp. kde se nachází jaké volání funkce (pole `PROG_CALLS`)

Po ukončení kontrol vstupu se před syntaktickými, částečně sémantickými a lexikálními kontrolami provede seřazení polí. Instrukce mohou být totiž v XML zapisovány v náhodném (avšak nenulovém a nezáporném) pořadí, což by při interpretaci později způsobovalo problémy. Proto se pole `PROG_INSTR` (a v návaznosti i `PROG_ARGS`) nejprve vzestupně seřadí, a poté se pořadí-order instrukcí přepíše tak, že pořadí instrukce bude rovno indexu v daném poli + 1 (resp. vznikne posloupnost 1,2,3, ..).

Pomocí těchto informací je možné provést kontroly počtů argumentů daných instrukcí (zda má instrukce správný počet operandů, zda jsou uvedeny všechny) včetně lexikálních kontrol. Pro kontrolu správných počtů a typů operandů jednotlivých instrukcí slouží seznam `INSTR_LIST`, který je seřazen od instrukcí s 0 operandy až po instrukce s 3 operandy. Následně se provádí kontroly návěští a volání. U návěští se kontrolují duplicitní definice, u volání zase, zda je volaná funkce (návěští) v programu přítomna.
Po všech kontrolách je možné inicializovat zásobník rámců a začít interpretaci zdrojového kódu. Interpretace probíhá ve while cyklu, obsahující počítadlo (resp. číslo-pořadí prováděné instrukce, začíná od 1), které je porovnáváno s pořadím poslední instrukce programu. V případě, že je počítadlo větší, interpretace již skončila a ukončí se skript s návratovým kódem 0 (v případě přepínače `--verbose` a neprázdným zásobníkem rámců, volání či programovým zásobníkem vypíše příslušné varování). Pokud však počítadlo větší není, provede se instrukce s pořadím, které je rovno počítadlu.
Samotné instrukce se provádí (maximálně možně) dle zadání a při provádění jednotlivých instrukcí se vykonávají případné sémantické a jiné kontroly, které přísluší každé jednotlivé instrukci.

Rámce, zásobník rámců

Zásobník rámců je implementován jako seznam-zásobník, který obsahuje další seznamy-rámce. Minimální velikost zásobníku je 2, jelikož vždy obsahuje globální rámec a dočasný rámec. Po inicializaci jsou oba prázdné. Při volání `CREATEFRAME` se pouze nastaví indikátor, že se má používat dočasný rámec (který je již vytvořen při inicializaci zásobníku). Teprve po instrukci `PUSHFRAME` se vytvoří nový rámec, kterým bude lokální rámec, do kterého se přesunou všechny hodnoty z dočasného rámce. Po ukončení práce s lokálním rámcem se použije `POPFRAME`, který přesune hodnoty z nejvrchnějšího rámce v zásobníku do dočasného rámce a následně ho odstraní. Zároveň se při této instrukci vždy vymaže obsah dočasného rámce. Po ukončení interpretace se provede `frame_destruct()`, což odebere ze zásobníku 2 hodnoty-rámce (při správné práci s rámci to budou globální a dočasný rámec).

Implementační dokumentace k 2. úloze do IPP 2020/2021
Jméno a přijmení: Dominik Horký
Login: xhorky32

Volání funkcí, skoky v instrukcích

Účely volání funkcí slouží seznam-zásobník `PROG_CALLSTACK`, který si zaznamenává, na kterém pořadí instrukce bylo volání provedeno. Při použití instrukce `CALL` se pořadí této instrukce vloží do tohoto zásobníku a dle informací ze seznamu návěští v poli `PROG_LABELS` program skočí na danou instrukci (resp. počítadlo v cyklu při interpretaci se změní na pořadí daného návěští). Naopak po použití instrukce `RETURN` se ze zásobníku odebera hodnota na jeho vrcholu a pokračuje se vykonávání programu o jednu instrukci dál, než je daná hodnota.
Skoky v instrukcích se vykonávají podobně jako volání funkcí s tím rozdílem, že se nic nikam nezaznamenává, ale pouze se změní počítadlo na pořadí daného návěští.

Doplňující informace

Protože jsem se rozhodl psát dokumentaci, komentáře i výstup skriptu (hlášky, nápovědu, ..) v první úloze česky, tak stejný jazyk používám i pro druhou úlohu tohoto projektu. Jelikož pro zpracování argumentů skriptu ‚interpret.py' používám `argparse`, bylo nutné případné chybové hlášky, apod. manuálně přeložit do češtiny, viz funkce `convertArgparseMessages(s)`.
Dále v argparse bylo nutné speciálně upravit návratové kódy při chybách, jelikož v původním nastavení by skript nevracel návratové kódy dle zadání. Tohoto jsem docílil pomocí modifikace chování při vyjímce `SystemExit` (po volání metody `parse_args()` z argparse).
Návratové kódy tohoto skriptu jsem se snažil volit tak, aby co nejlépe odpovídaly zadání. Navíc však existuje možnost návratového kódu 101, který skript vrátí pouze tehdy, pokud byla interpretace násilně ukončena kvůli velkému počtu volaných funkcí (viz přepínač `--recursive`).

Skript ‚test.php'

Parsování argumentů skriptu probíhá pomocí cyklu a regexu. Skript podporuje všechny parametry zadání, navíc zpracovává přepínač `--verbose`, který vypisuje průběh konání skriptu na standardní chybový výstup (vzhledem k povaze výstupu tohoto skriptu nebylo možné použít standardní výstup).
Skript po úspěšném parsování argumentů zkontroluje existenci vstupů, resp. zda existují soubory uvedené v argumentech programu (či jejich implicitní hodnoty dle zadání).
Následně probíhá fáze testování, kde se nejprve provede příprava. Ta spočívá v indexaci (vytvoření seznamu všech testů). Pokud je přítomen přepínač `--recursive`, tato indexace probíhá rekurzivně. Poté se kontroluje validita všech testů a to tak, že se zkontroluje existence souborů s příponami `src, rc, out, in`. V případě, že některý ze souborů (kromě `*.src`) chybí, tak se vytvoří a případně (pokud jde o `*.rc`) se dopíšou implicitní hodnoty-data. Pokud chybí zdrojový soubor (`*.src`), test se označí za nevalidní a nebude proveden. Jakmile je tato příprava hotová, začíná testování, které nespočívá v ničem jiném, než porovnávání výstupů a návratových kódu s referenčními. Pokud je přítomen přepínač `--verbose`, program bude na standardní chybový výstup vypisovat, kolikátý test zrovna (z celkového počtu testů) probíhá.
Po ukončeném testování se na standardní výstup začíná generovat HTML5 kód – stránka, která zobrazí přehled o tom, kolik testů bylo provedeno, kolik uspělo či ne a tabulku obsahující jména jednotlivých testů, jejich výsledky a případně poznámky, pokud test neskončil úspěšně.

Design HTML stránky

Zvolil jsem jednoduchý, nenáročný, rychlý, se všemi (nebo alespoň s většinou) zařízeními kompatibilní design spočívajícím v prostém textu a jednoduché tabulce. Bylo použito pouze prosté HTML5. Bohužel, na (pouze) školních serverech Merlin i Eva stránka z neznamého důvodu nepodporuje diakritiku a tento problém se mi nepodařilo vyřešit.
Stránka obsahuje krátký úvod, zvolené nastavení, které obsahuje informace o zvoleném způsobu testování (jako je zvolený adresář testů, přítomnost přepínače `--recursive`, apod.) a souhrn, s počtem všech provedených testů, počtem prošlých či neprošlých testů (včetně procentního vyjádření z celkového počtu testů).

Neprovedené testy

Jelikož se může stát, že některému testu chybí zdrojový soubor a je označen za neplatný, takový test se neprovede. Přišlo mi vhodné do celkového počtu testů (v souhrnu) takové testy nezahrnovat, nicméně jsem je nechtěl úplně vynechat. Proto jsou tyto testy označené v tabulce jako neprovedené a započítané zvlášť.