Implementační dokumentace k 2. úloze do IPP 2020/2021
Jméno a příjmení: Václav Sysel
Login: xsysel09

interpret.py

Interpret využívá knihovny argparse pro parsování argumentů. Vstupní cesty k souborům jsou ověřeny. Při chybě ověřování či jiných chybách se volá třída Error, která obsahuje enumerace všech možných chybových kódů, společně s krátkými informačními textovými zprávy. Pro lepší informaci o místě chyby je zde globální proměnná last_executed. Tato proměnná se při pádu vypisuje. Uchovává pořadí od 0 aktuálně interpretované instrukce. last_executed -1 znamená, že program našel chybu již při kontrole XML. Pro interpretaci se tvoří objekt interpret. Tento objekt je závislý na velkém množství pomocných tříd a objektů, které uchovává jako svoje atributy. Tyto pomocné objekty jsou například argument, stack (interně reprezentovaný jako list s pomocnými funkcemi), frame či jump_table (interně reprezentované jako jako slovník s pomocnými funkcemi). Některé z těchto objektů při neočekávaných vstupech volají třídu Error a ukončují běh programu. Některá selhání mohou mít více chybových kódů, z tohoto důvodu některé metody neukončují běh programu, ale pouze propagují chybu výš v podobě hodnoty None. Při tvorbě interpretu se otevřou soubory a uloží jako atributy, pokud jeden ze souborů chybí, nahradí se standardním vstupem. Tyto soubory jsou otevřené během celého běhu programu a zavírají se při finalizaci objektu.

V prvním kroce interpret načte xml etree ze vstupního souboru a získá jeho kořen. Tato váze je zcela závislá na knihovně xml etree, a spočívá pouze ve volání funkcí z této knihovny.

V druhém kroce se kontroluje, zda xml obsahuje očekávané elementy a atributy a kontroluje se order instrukcí. Zde se již nevyužívá funkcí xml etree ale masivně se zde využívají objekty vytvořené pomocí této knihovny.

V třetím kroce probíhá převod xml elementů do objektů argument a instruction. Nejdříve je vytvořen objekt instrukce, do kterého je uložen order a operační kód. Do této instrukce jsou následně přidány objekty argumentů. V této části zatím nedochází ke kontrole argumentů, pouze k jejich tvorně. Takto vytvořená instrukce je následně uložena do seznamu instrukcí v interpretu. Na konec jsou instrukce seřazeny dle své hodnoty order.

Ve čtvrtém kroce proběhne kontrola samotných instrukcí. Kontroluje se, zda instrukce existují a jestli načtené argumenty odpovídají zápisu. K tomuto účelu napomáhá slovník testovacích funkcí. Interpret kontroluje, zda načtená funkce má svou ekvivalentní testovací funkci ve slovníku. Pokud ano, tato funkce se zavolá. Testovací funkce blíže kontrolují, zda načtené argumenty jsou správné. Některé kontroly není možné provést a budou provedeny až za běhu, například kontroly typů proměnných. Pro kontrolu slouží dříve definované regulární výrazy, zabalené do funkcí. Při tomto kroku se taktéž ukládají návěští do tabulky skoků. Při samotné interpretaci se již pouze kontroluje, zda je návěští v tabulce skoků. Tento krok se již velmi podobá samotné interpretaci a využívá funkce, které se využívají i při interpretaci, například manipulace s atributem current_instruction či pomocné funkce, které zabalují časté manipulaci s argumenty (např: funkce get_argument, která vrací argument na dané pozici právě zpracovávané instrukce na pozici current_instruction, či funkce check_arguments_sum, která kontroluje počet argumentů).

Po dokončení se přejde k pátému kroku, samotné interpretaci. Tento krok je velmi podobný předchozímu, avšak místo funkcí ze slovníku testovacích funkcí, se získávají a volají funkce z „ostrého“ slovníku. Každá instrukce má tady dvě funkce ve dvou slovnících. Testovací funkci, která kontroluje argumenty a ostrou funkci, která provádí detailnější kontroly a provádí samotnou akci. Funkce instrukcí jsou od pomocných instrukcí výrazně odlišeny. Funkce instrukcí jsou vždy zapsány hůlkovým písmem (s výjimkou dodatku _test u testovacích funkcí) pomocné funkce jsou zpravidla zapsány malým písmem. Pozice instrukcí nejsou vnitřně reprezentovány svojí hodnotou order, ale svojí seřazenou pozicí, číslovanou od nuly. Při skocích je jednoduše přepsána hodnota current_instruction seřazenou pozicí v tabulce skoků či je hodnota current_instruction uložena do zásobníku pozic, (position_stacl) či je přemazána hodnotou ze zásobníku pozic. Hodnoty se kterými pracuje uživatel mohou být ukládány několika způsoby, buď jsou ukládána do nebo vyjímána ze zásobníku dat (data_stack), či mohou být ukládána do proměnných. Proměnné jsou pak uloženy ve slovnících globálního, dočasného či lokálního rámce. Lokální rámec je pak ve skutečnosti celý zásobník rámců a umožňuje se přístup pouze do vrchního. Se zásobníkem rámců lze pomocí instrukcí manipulovat a jednotlivé rámce lze vyjmout a dát na pozici dočasného rámce, či dočasný rámec uložit do zásobníku lokálních rámců. Pro snazší manipulaci se všemi proměnnými a argumenty existují mnohé pomocné funkce, které zabalují nejčastější užití, například funkce get_symb se podívá na argument na dané pozici, pokud je argument literál, propaguje jej výše, pokud je argument proměnná, pokusí se získat v ní uloženou hodnotu a tu pak propaguje výše. Či funkce prepare_var, která propaguje výše proměnnou, jejíž jméno je uloženo v daném argumentu. Některé funkce jsou jednořádkové a nejsou nahrazeny z úspory řádků, ale z důvodu lepší čitelnosti a menší redundance kódu. Některé málo se opakující se případy jsou stále redundantní. Manipulace s objekty ve většině případů kopírují popis ze zadání. Pokud current_instruction převýší seřazenou pozici poslední funkce, považuje se kód za úspěšně interpretovaný a program se ukončí. Jednotlivé funkce pak taktéž mohou program předčasně ukončit v případě chyb, či interních selhání. Program neobsahuje žádnou ochranu proti zacyklení. Program neobsahuje kontrolu přetečení zásobníků a ani kontrolu přetečení slovníků. Tyto problémy není běžné v pythonu řešit a proto nejsou řešeny ani v této implementaci.