Implementační dokumentace k 2. úloze do IPP 2020/2021

Jméno a příjmení: David Hudák
Login: xhudak03

1 Úvod

Tato dokumentace se zabývá řešením druhého projektu v rámci předmětu principy programovacích jazyků. Jeho cílem je v první řadě implementace intepretu jazyka `IPPcode21`, který byl v předchozí části projektu předzpracován do formy XML souboru. V druhé řadě je jeho cílem implementovat skript na zpracování testů k interpretu (z druhé části projektu) a parseru (z první části projektu).

2 Interpret

Tato část dokumentace se zabývá řešením interpretu jazyka `IPPcode21`. Implementace proběhla v jazyce Python verze 3.8. K práci s XML souborem byla použita knihovna `xml.etree.ElementTree`[1]. Na zpracování užitečných regulárních výrazů pak byla použita knihovna `re`[2].

2.1 Ošetření vstupů

Program podporuje (kromě `--help` na vyvolání textové nápovědy) dva parametry, a to `--input=file` a `--source=file`. V případě (ne)použití argumentu `--source` se běh programu liší pouze v tom, že se použije příkaz `ET.parse()` buď na zadaný soubor, nebo na standardní vstup.

U (ne)použití argumentu `--input` se už běh programu liší poněkud více. V případě, že tento argument použit nebyl, nastaví se přepínač vstupu na `False` a v instrukci `read` (jediná instrukce, která pracuje se vstupy) je následně použit příkaz pro práci se vstupem, zatímco v případě zadaného souboru se rovnou načte celý vstup do pole, ze kterého se při každém požádání o vstup odebere první řádek.

2.2 Lexikální a syntaktická kontrola

Celou lexikální a syntaktickou analýzu obstarává funkce `lexSynAnalys`. Na počátku zkontroluje potřebné hlavičky funkcí.

Jako druhý krok (poněkud neefektivně) projede celý kód pro kontrolu přítomnosti atributu `order` a pro kontrolu jeho hodnot (musí se jednat o kladná čísla). Tento krok je důležitý pro další vykonávání programu z toho důvodu, že instrukce mohou být ve zdrojovém souboru zpřeházené, ale při samotné interpretace musí být interpretovány právě podle čísla `order`. Následně se soubor seřadí s pomocí příkazu `sorted` a jednoduché lambda funkce.

V poslední části lexikální a syntaktické kontroly dojde opět k projití celého kódu, kdy dochází:

- Ke kontrole existence instrukce ve slovníku[3] instrukcí.
- Ke kontrole počtu argumentů na základě informace slovníku instrukcí.
- Ke kontrole jednotlivých argumentů dle informací ze slovníku instrukcí.
- Při nalezení instrukce `LABEL` vykonání její operace (definice návěští ve slovníku návěští, kontrola proti redefinicím, zápis řádku návěští).

2.3 Interpretace instrukcí

Pro interpretaci instrukcí je použit `while` cyklus pracující s aktuálním číslem řádků a celkovým počtem řádků. To je důležité především z důvodu přítomnosti instrukcí skoku, které by se například při `foreach` cyklu dělaly poněkud obtížněji. V cyklu se tedy dá v podstatě libovolně vracet v kódu.

Srdcem tohoto cyklu pak je v podstatě konečný automat (není to tak primárně implementováno), který nejdříve rozhodne, zdali se jedná o instrukci skoku (například `CALL`, `JUMPIFEQ`, `JUMP` atd.), nebo o jinou instrukci. Pokud se nejedná o instrukci skoku, konečný automat rozhoduje, do jaké kategorie instrukce patří na základě informace ze slovníku instrukcí o počtu a typech operandů – některé instrukce nemají žádný operand, některé mají na první pozici nutně proměnnou, některé mohou mít pouze dva argumenty atp. (více detailů v zadání projektu).

Jednotlivé instrukce jsou pak každá jedna obsažena ve vlastní funkci. Z určitého pohledu to není efektivní z hlediska množství kódu (například v případě aritmetických instrukcí se dá hovořit přímo o plýtvání), avšak výhodou jsou snadnější opravy a přidávání či odebírání jednotlivých instrukcí.

Jako poslední je vhodné zmínit, že při interpretaci existuje globální proměnná `frames`, která je slovníkem slovníků a obsahuje jednotlivé rámce[4], v nichž jsou slovníkem zapisovány jednotlivé proměnné, které uchovávají informaci o hodnotě a typu.

[1] Viz dokumentace `xml.etree.ElementTree`.
[2] Viz dokumentace `re`.
[3] Slovníkem je v tomto mínění přímo myšlena klasická struktura slovníku v jazyce Python.
[4] `GF`, `LF` a `TF`.

3 Testovací skript

Tato část dokumentace se zabývá řešením testovacího skriptu pro interpret jazyka `IPPcode21`. Program je implementován v jazyce PHP ve verzi 7.4 a pro testování shody výstupních XML souborů byl použit Java balíček JExamXML[5].

3.1 Práce s HTML

Pro tvoření HTML souboru bylo použito pouze funkcí pro tisknutí a zpracování řetězců jazyka PHP. Krom funkcí na přidání základní hlavičky, konce souboru a titulků také byla využita práce s HTML tabulkami.

3.2 Načítání souborů

Pro načtení souborů je použita funkce obsahující funkci `scandir`, která načte celý adresář dle názvu. Načtená data se projdou `foreach` cyklem, který oddělí adresáře od souborů (do dynamického pole přidá pouze soubory), a pokud uživatel zvolil možnost `--recursive`, pak také při každém procházeném adresáři do původního zdroje nahraje všechen nově načtený obsah.

3.3 Jednotlivé testy

Nový test vzniká nalezením souboru s příponou `src` a z něj se s pomocí regulárního výrazu odstraní přípona. Výsledek bez přípony je pak názvem testu. Jednotlivé testy jsou pak nahrávány do pole struktur `MyResult`, která obsahuje jméno testu, výsledek testu (binárně `True` a `False`) a informaci o testu (pokud byl test úspěšný, informace je prázdný řetězec, pokud neúspěšný, pak obsahuje krátkou informaci o důvodu selhání – rozdílná výstupní data, nebo rozdílné výstupní kódy).

3.4 Způsob testování

Při testování vznikají tři různé možnosti, co může nastat:

- Testovat se bude jenom parser, zdrojem bude kód v jazyce `IPPcode21` a výstupem XML soubor.
- Testovat se bude jenom interpret, zdrojem bude kód ve formátu XML a výstupem výstup interpretace kódu.
- Testovat se bude kombinace parseru a interpretu, kde zdrojem bude kód v jazyce `IPPcode21` a výstupem výstup interpretace kódu.

V případě testování pouze parseru dochází k postupnému procházení všech testů, kde se nejprve s pomocí nástroje příkazu `diff` porovnají výstupní hodnoty programu, a pokud obě byly 0 (program proběhl správně a program měl proběhnout správně), nástroj JExamXML porovná výstupy očekávané a výstupy dodaného parseru.

V případě interpretu se oproti pouze parseru změní jenom typ vstupních dat (a způsob jejich zápisu – přes argumenty programu) a zahrne se možnost vstupních dat (soubory s příponou `in`). Místo JExamXML je pak i na výstupy očekávané a dodané použit nástroj `diff`. V případě testování parseru i interpretu se pak postupuje podobně, kdy vstup je v jazyce `IPPcode21`, pokud dojde k úspěchu parseru (je vrácena hodnota 0), pokračuje se interpretem. Následně se kontroluje výstup, jak bylo zmíněno pár vět dříve.

3.5 Tvořené soubory

Pro samotné ladění vzniká při testování několik souborů u každého testu. Jsou jimi:

- Soubory s příponou `log` a `log2`, které obsahují standardní chybové výstupy programů (při kombinaci testů parseru i interpretu se používá `log` pro parser a `log2` pro interpret, v případě jenom jednoho typu testů se používá jen `log`.
- Soubory s příponou `resv`, které obsahují výslednou výstupní hodnotu.
- Soubory s příponou `my` a `myr`, které obsahují výstup a výsledek porovnání výstupu.
- Soubory s příponou `xml` jsou používány při testech obojího (nikoliv při testech samotného parseru) a obsahují mezivýsledek překladu (XML kód získaný z parseru).

  Všechny tyto soubory jsou obsaženy ve stejné složce, ve které se nachází test.

4 Závěr

Výsledkem řešení projektu jsou dva skripty. Jeden řeší interpretaci XML souborů obsahující zpracovaný kód jazyka `IPPcode21` a druhý (nejen) jeho testování na testovacích souborech. Program byl testován na Ubuntu v systému WSL a na školním serveru Merlin.

[5] Viz GitHub.