Implementační dokumentace k 1. úloze do IPP 2022/2023
Jméno a příjmení: Aleksandr Shevchenko
Login: xshevc01

1 Struktura programu

Moje implementace skriptu `parse.php` sestává z následujících logických částí:

- **functions**, kde jsou pomocné funkce pro výpis nápovědy k použití skriptu, parsování argumentů, zadaných uživatelem. Taky se v této sekci nachází všechny funkce pro práci s XML a ověření správnosti (neboli „matching“) jednotlivých operandů v jazyce IPPcode23 (dál *match-funkce*).
- **cases**, obsahující funkce, které pro každou instrukci jazyka IPPcode23 ověřují správnost jejich použití, a to počet předaných operandů a kontrola, zda hodnoty všech operandů odpovídají specifikaci.
- **main body**, což je hlavní tělo programu.

2 Práce s XML a regulárními výrazy

Pro jednodušší zápis do formátu XML jsem použil knihovnu **XMLWriter**. Začátek a konec XML kódu ovládám pomocí funkcí `xml_start` a `xml_stop`, zápis instrukcí a operandů funkcemi `xml_print_instr` a `xml_print_operand`.

Hlavní myšlenkou správného fungování skriptu je použití regulárních výrazů. Pomocí funkcí `preg_replace`, `preg_match` a `preg_split` ověřuji, zda instrukce a operandy vstupního programu odpovídají specifikaci.

3 Průběh programu

Úplně na začátku provádím kontrolu vstupních argumentů funkcí `parse_args`, která v případě přepínače `--help` zavolá výpis nápovědy `print_help`, a taky může skončit program návratovou hodnotou 10, pokud počet argumentů je špatný. Jinak skript běží dál.

Pak inicializuji pomocné proměnné **$xml**, kam postupně vkládám nutné informace pro výpis, **$order** pro počítání řádků instrukcí a **$header**, který se nastaví na hodnotu `true` po přečtení hlavičky.

Kostrou programu je `while` cyklus, ve kterém postupně čtu řádky kódu IPPcode23 za použití `fgets(STDIN)`. Pomocí již zmíněných funkcí pro práci s regulárními výrazy odstraňuji komentáře, prázdné řádky, mezery či tabulátory. Pokud první neprázdný řádek není hlavička, program se skončí hodnotou 21, v opačném případě $header se nastaví na `true` a pro další řádky kontrola hlavičky už probíhat nebude.

Každý následující po hlavičce neprázdný řádek inkrementuje hodnotu $order a prochází přes `switch`. Každou instrukci jazyka IPPcode23 jsem zařadil do skupin podle typů operandů, které potřebuje. Pak se ve switchu volá obecná funkce, která umí zpracovávat konkrétní skupinu instrukcí, jinak v případě nevalidní instrukce v `default` se vyvolá návratová hodnota 22 a program se ukončí.

Všechny funkce pro zpracování instrukcí (dál *case-funkce*) začínají přeponou `case_` a končí seznamem nutných parametrů, například `case_var_symb_symb`, kromě `case_no_params` v případě, že žádný nepotřebuje. Každá case-funkce kontroluje počet přijatých parametrů, případně ukončí program chybou 23, jinak volá `xml_print_instr` pro zápis dané instrukce do $xml a match-funkce pro jednotlivé operandy.

Match-funkcí je celkem 4: `match_var`, `match_label`, `match_symb` a `match_type`. Každá z nich pomocí regulárních výrazů provádí kontrolu, jestli předaný operand splňuje požadavky, uvedené v zadání (například jestli na místě pro $\langle type \rangle$ je opravdu int, string anebo bool).

V případě, když žádná chyba nenastane, obsah vnitřní paměti $xml jde na standardní výstup a program se končí.

Největší potíže během implementace tohoto projektu nastaly u regulárních výrazů, zvlášť pro string symboly. Kvůli tomu, že znak `\` se může vyskytovat jenom v případě dekadického zápisu znaků, nedařilo se mi napsat jeden regulární výraz pro správný match. Vyřešil jsem to tak, že na začátku z každého stringu vynechal všechny dekadické znaky pomocí `preg_replace`, a jenom pak kontroloval na vyskyt `\`. Taky stojí za zmínění, že pro nalezení jednoho `\` potřebujeme regulární výraz ze čtyř `\` kvůli vnitřní reprezentaci tohoto znaku v samotném php, což mě taky zabralo mnoho času.