Implementační dokumentace k 1. úloze do IPP 2020/2021

Jméno a příjmení: Roman Popelka
Login: xpopel24

1.Implementace

1.1 Scanner (lexikální analyzátor)

Pro scanner jsem vytvořil jednu hlavní funkci `get_tokens($file,&$statistics)`, která jako parametr brala soubor `$file` (v tomto případě stdin) a proměnnou `$statistics` typu `Statistics`, což je mnou vytvořená třída pro uchovávání statistik o programu (viz. rozšíření STATP) a jejich vypsání do souboru.

Funkce čte řádek po řádku, přičemž kontroluje, zda čtení proběhlo správně. Pokud při čtení nastala jakákoliv chyba, ukončuje funkci se správným chybovým kódem (chybové kódy jsou definovány ve třídě `ErrNums` v adresáři scripts, v souboru constants.php). Tyto návratové kódy se vrací v poli, z důvodu jednotnosti výstupu. Program dále pokračuje kontrolou, zda na řádku není komentář, kvůli statistikám.

Následuje nejduležitější část implementace Scanneru. Pokud se zjistí, že je řádek prázdný, nebo že je přes celý řádek komentář, řádek se přeskakuje. Pokud řádek obsahuje něco jiného, pokračujeme ořezáním mezer a komentářů vedle kódu. Celý řádek se pak rozdělí na jednotlivá slova do pole a začíná lexikální kontrola. Ta probíhá tak, že se každé slovo porovnává s regulárními výrazy definujícími jazyk. Ty jsou definované jako konstantní pole `REGEX_ARR` v třídě `TokenConstants` společně s tokeny. Index regulárního výrazu určujícího token, je stejný jako "číslo" tokenu. To nám umožňuje poměrně rychlé nalezení správného regulárního výrazu (jednoduchý cyklus) a také jednoduché vrácení správného tokenu (vracíme pouze index, nemusíme hledat přiřazený token).

Tyto operace se pak provádějí pro všechna slova v řádku. Funkce vrací celý řádek tokenů, nikoliv jen jeden (tedy pole tokenů vyskytujících se na daném řádku). Tato úprava pak zlepší práci parseru s tokeny

1.2 Parser (syntaktický analyzátor)

První věc, která se po spuštění parseru provede, je zavolání funkce `parse_args($argv,$argc)`, která vezme parametry na vstupu a zkontroluje jejich správnost (v případě chyby volá `exit()` se správným návratovým kódem, zároveň tvoří pole `$permissions_array` typu `StatisticsPermissions`, což je vlastně jen třída, která napodobuje strukturu a uchovává data o parametrech pro statistiky. Funkce vrací toto pole. Pokud nebyl použitý argument `--stat`, vrací prázdné pole.

Po parsování argumentu už jen vytvoříme novou instanci třídy `Statistics` a několik pomocných polí pro statistiky.

Následuje otevření standardního vstupu. Pokud se soubor podaří otevřít, voláme poprvé funkci `get_tokens($file,$statistics)`, které předáváme standardní vstup a naši instanci třídy `Statistics`.

Gramatiku jsem poněkud nešikovným způsobem definoval ve zdrojovém souboru `constants.php` jako třídu jménem `GRAMMAR` obsahující soubor konstatních polí skládajících se z tokenů (např. `LABEL <label>` je uložen jako `array(TokenConstants::KW_LABEL, TokensConstants::LABEL)`. S tím, že jedna instrukce je na jednom řádku, mi bylo umožněno nadefinovat gramatiku, právě jako pole kombinací tokenů. Není sice moc paměťově efektivní, na druhou stranu zvyšuje čitelnost kódu tím, že porovnávám výstup scanneru s gramatikou jen pomocí jednoduché php funkce `in_array()`.

Program pak už jen jede v jednoduchém cyklu, dokud nenarazí na konec vstupu a kontroluje gramatiku způsobem popsaným výše. Zároveň se vytváří struktura xml souboru pomocí knihovny `DOMDocument`, který se při úspěšném parsování zdrojového souboru vypíše na standardní výstup.