Implementační dokumentace k projektu do IPP 2017/2018

Jméno a příjmení: Zdeněk Brhel
Login: xbrhel04

Parser – parse.php

Spouštění programu

`php parse.php --help` – vypíše nápovědu
`cat %SOUBOR% | php parse.php` – přeloží soubor `%SOUBOR%` na výstupní XML

`getParams($argc, $argv)` – v případě, že s programem přišel parametr –help, vypsala se nápověda na STDIN, jinak se program ukončil a vypsala se nápověda ke spuštění programu.
`fgets(STDIN)` – v cyklu se nahrává ze STDIN řádek po řádku, který se následně přes funkci `removeComments($file)` zbaví všech komentářů.
Odeberou se všechny prázdné řádky a řádek se rozdělí po slovech oddělených tabulátorem, nebo mezerou. Tyto pole slov následně reprezentuje řádek s instrukcí a její argumenty.
Kontroluje se povinná hlavička `.IPPCode18` a provede se lexikální analýza pomocí funkce `lexical($parts)`.

Lexikální analýza

Pomocí přepínače se kontroluje správnost všech možných instrukcí tak, že se kontroluje jméno instrukce (`MOVE,CONCAT,..`) a počet argumentů a jejich správnost pomocí funkcí:
`expectSymb, expectVar, expectLabel, expectFrame, expectLabel, expectType`.
Všechny funkce, krom `expectSymb` vrací boolovskou hodnotu, tato funkce vrací i datový týp určený podle předpon (`string@, int@, ..`). Tento datový typ se následně ukládá pro účely generování konečného XML. V případě, že do lexikální analýzy přijde neznámý příkaz, vypíše se na STDERR chyba a číslo řádky.

Generování výstupního souboru

Modul `xmlwriter` zajišťuje pohodlné generování výstupu, jako první je třeba vytvořit paměť XML souboru, což je možné pomocí `xmlwriter_open_memory()`. Nastavení odsazení, kódování a kořenu výstupního XML zajišťovaly funkce `xmlwriter_set_indent($output, 1)` (odsazení), `xmlwriter_start_document($output,'1.0','UTF-8')` (kódování) a `xmlwriter_start_element($output,'program')` (kořen XML).
Funkce `xmlwriter_start_attribute($output,%ATTRIB%)` je určena k nastavování atributů elementů (`%ATTRIB%` je řetězec znaků obsahující název atributu aktuálního elementu), následně pak `xmlwriter_start_text($output,%TEXT%)` nastavovala obsah naposled vytvořeného atributu, nebo obsah naposled vytvořeného elementu (`%TEXT%` reprezentuje obsah tohoto atributu). Funkce `xmlwriter_end_attribute($output)` pak ukončovala naposled vytvořený atribut, stejně jako funkce `xmlwriter_end_element($output)` ukončovala naposled vytvořený element. Po vygenerování celého souboru v paměti se vypsal pomocí `xmlwriter_output_memory($output)`

Důležité proměnné v souboru parse.php

`$instructionnumber` – slouží k uchovávání aktuálního pořadí příkazů
`$parts` – pole obsahující jméno instrukce a její parametry
`$args` – pole sloužící k uchovávání argumentů důležitých pro výstup
`$output` – paměť výstupního souboru

Interpret – interpret.py

Spouštění programu

`python interpret.py --help` – vypíše nápovědu
`python interpret.py --source=%SOUBOR%` – interpretuje vstupní soubor `%SOUBOR%`

Funkce `arguments()` zpracuje parametry a vrátí cestu k vstupnímu souboru. Zkontroluje, že je vstupní soubor existuje a zda je to platný XML soubor, následně se s pomocí knihovny `xml.etree.ElementTree` a její funkcí `parse()` rozdělí na strom, kdy kořen je nejníž zanořený element `<program>`.
V cyklu, který se ukončí jen v případě, že neexistuje takový příkaz, který by měl být další v pořadí.
Další příkaz se vyhledává příkazem
`xml.findall('instruction[@order=\''+str(currentIndex)+'\']')`, který nalezne příkaz s pořadím `currentIndex`. Důvod tohoto řešení jsou skokové instrukce a zjednodušení řízení toku programu. Zároveň se ošetřuje, zda příkaz s tímto indexem byl nalezen pouze jeden.

Interpretace programu

Samotná intepretace probíha pomocí funkce `interpret(currentCommand[0])`, kde parametrem této funkce je element s aktuálním příkazem. U každého elementu se kontroluje jeho atribut `opcode`, který obsahuje jméno instrukce. Podle jména instrukce se vykoná daná funkce.

Řešení rámců, štítků, skoků

Globální slovníky `globalFrame, tempFrame` zajišťují vhodnou funkcionalitu rámců, výjimka je pak `localFrame`, který je pole těchto slovníků z důvodů instrukcí `PUSHFRAME` a `POPFRAME`, které pozměňují tento stack rámců. Další speciální proměnná je pak `canCreateFrame`, která zajišťuje, že musí být zavolána instrukce `CREATEFRAME`, pokud se program snaží přistupovat do `tempFrame`.

Globální slovník `labels` pak zajišťuje funkcionalitu štítků. Do tohoto slovníku se ukládá jméno proměnné a následně i řádek, na které se nachází.

Skoky jsou pak řešeny pomocí proměnné `currentIndex`, která řídí tok programu. Vyhledá se štítek ve slovníku `labels` a následně se na `currentIndex` přiřadí jeho řádek. Na vrácení se pomocí funkce `return` pak je pole čísel, kdy číslo reprezentuje řádek, na který se má program vrátit.

Důležité proměnné v souboru interpret.py

`instCount` – počet vykonaných instrukcí
`dataStack` – zásobník dat
`globalFrame, tempFrame, localFrame` – vysvětleno výše
`currentCommand` – obsahuje element s aktuálním příkazem, jeho parametry, ...
`currentIndex` – obsahuje číslo aktuálně vykonávané instrukce

Zajímavý kóp v interpret.py

Kód uloží aktuální řádek do zásobníku a následně se aktuální řádek přepíše na jiný, program pokračuje podle dalšího řádku.

```
returnIndex.append(currentIndex)
currentIndex.labels[tmp.text)
```