Implementační dokumentace k projektu do IPP 2017/2018
Jméno a příjmení: Vojtěch Čurda
Login: xcurda02

1 Parser

Parsovací skript jako celek se skládá ze dvou hlavních částí – Tříd `Scanner` a `Parser`. Pro zapisování XML reprezentace kódu se používá knihovna `SimpleXML`. V případě syntakticky/lexikálně chybného kódu se vyhazuje výjimka s odpovídající návratovou hodnotou. Výjimky jsou zachycovány ve funkci `main()` v souboru `parse.php`, kde se v případě zachycení ukončí program s příslušnou návratovou hodnotou a XML reprezentace kódu se v tomto případě nevypisuje.

1.1 Třída `Parser`

Třída `Parser` se stará o celé parsování. Prochází vstupní kód ve formě tokenů převzatých od třídy `Scanner` a rekurzivním sestupem kontroluje správnost jejich posloupnosti. Výstup do XML souboru provádí třída `Parser` skrze třídu `Instruction`, kdy zavoláním jejího konstruktoru s parametry kořenového prvku XML souboru, operačního kódu a pořadí instrukce se vytvoří XML záznam této instrukce. Voláním metody `addArg` se přidá operand instrukce do dané XML reprezentace.

1.2 Třída `Scanner`

Načítání vstupu je prováděno ve třídě `Scanner`. Ta načítá a převádí načítané lexémy do formy tokenů a postřednictvím třídy `Token` vrací třídě `Parser` aktuálně načtený token. Parser voláním metody `next()` na objekt typu `Token` provede načtení dalšího tokenu. Ve třídě tokenu se vede záznam o textové reprezentaci lexému a typu tokenu. Typy tokenu jsou následující: `AT_SIGN` - znaménko '@', `EOL` - Konec řádku, `EOF` - Konec vstupu, `HEADER` - Hlavička (.IPPcode18) a všechno ostatní se klasifikuje jako `STR` kvůli kontextové závislosti jazyka IPPcode18. Z tohoto důvodu tomu se většina kontrol tokenů provádí až ve třídě `Parser`.

1.3 LL-gramatika

Parsování se řídí následujcí LL-gramatikou:

```
P    -> .IPPcode18 eol INSL eof
INSL -> INS eol INSL
INS  -> move VAR SYMB
INS  -> createframe
INS  -> pushframe
INS  -> popframe
INS  -> defvar VAR
INS  -> call LABEL
INS  -> return
INS  -> pushs SYMB
INS  -> pops VAR
INS  -> add VAR SYMB SYMB
INS  -> sub VAR SYMB SYMB
INS  -> mul VAR SYMB SYMB
INS  -> idiv VAR SYMB SYMB
INS  -> lt VAR SYMB SYMB
INS  -> gt VAR SYMB SYMB
INS  -> eq VAR SYMB SYMB
INS  -> and VAR SYMB SYMB
INS  -> or VAR SYMB SYMB
INS  -> not VAR SYMB
INS  -> int2char VAR SYMB
INS  -> stri2int VAR SYMB SYMB
INS  -> read VAR <int|string|bool>
INS  -> write SYMB
INS  -> concat VAR SYMB SYMB
INS  -> strlen VAR SYMB
INS  -> getchar VAR SYMB SYMB
INS  -> setchar VAR SYMB SYMB
INS  -> type VAR SYMB
INS  -> label LABEL
INS  -> jump LABEL
INS  -> jumpifeq LABEL SYMB SYMB
INS  -> jumpifneq LABEL SYMB SYMB
INS  -> dprint SYMB
INS  -> break
INS  -> eps
SYMB -> VAR
SYMB -> <int|string|bool>@<cnst_val>
VAR  -> <GF|LF|TF>@LABEL
LABEL -> <id>
```

2 Interpret

Interpretace vstupního kódu je řízena třídou `Interpreter`. Načítání vstupu ze XML souboru je řešeno knihovnou `ElementTree`. Kontrola syntaxe vstupu se provádí během interpretace jednotlivých instrukcí.

2.1 Třída `Interpreter`

Začátek interpretace kódu se volá v souboru `interpret.py` metodou `runcode()` na objekt typu `Interpreter`. Na tuto metodu je prováděno zachycování výjimek, které značí chybu interpretace.

Ještě před samotným prováděním jednotlivých instrukcí se projde celý vstup za cílem získání deklarací návěští. Návěští jsou poté uložena v proměnné `label_dict` ve formě slovníku, kde název návěští je klíč, podle kterého se vyhledá jeho odpovídající pozice v kódu. Poté je nutno získat informace o aktuální instrukci a jejích operandech. O toto se stará třída `FileParser`, která bude popsána později. Než se začnou interpretovat jednotlivé instrukce, zkontoluje se syntaktická správnost operandů.

2.2 Třída `FileParser`

Třída `FileParser` slouží jako prostředník mezi XML vstupem a interpretem. Její metody umožňují kromě získávání instrukcí a jejich operandů ještě kontrolu nad pohybem ve vstupním souboru. To se využívá zejména při přechodu na další zpracovávanou instrukci a také na skokové instrukce, kdy při nutnosti skoku se nastaví iterátor všech elementů `<instruction>` na požadovanou instrukci podle atributu `opcode`. Jelikož kontrola syntaxe probíhá těsně před interpretací jednotlivé instrukce, může se stát, že některé instrukce budou zkontrolovány vícekrát.

2.3 Implementace rámců

Jednotlivý rámec je popsaný ve třídě `Frame`. Komunikace s nimi je implementovaná ve třídě `Frames`. Při volání konstruktoru třídy `Frames` se automaticky vytvoří globální rámec a uloží do pole rámců. Toto pole existuje kvůli jednodušší práci s rámcemi - aby k nim šlo přistupovat podle předdefinovaných konstant `TF,LF,GF`. Kvůli tomu je lokální rámec implementován rozdílněji, než by se čekalo. Lokální rámec není na vrcholu zásobníku, ale chová se tak a ve skutečnosti je mimo zásobník. Až při volání instrukce `PUSHFRAME` se lokální rámec přesune na vrchol zásobníku a dočasný rámec, se stane lokální. Tento způsob implementace rámců nijak neomezuje interpretaci instrukcí, pracujícími s rámci. Instrukce se chovají podle zadání.