Implementační dokumentace k projektu do IPP 2017/2018

Jméno a příjmení: Alena Tesařová

Login: xtesar36

**Parse.php - postup řešení**

Rozdělila jsem si instrukce do 4 kategorií: A, B, C, D a vytvořila jsem si pro každé pravidlo (typ proměnné, identifikátor, konstanta, symbol atd.) jeden regulární výraz.

1. **3 operandy**
   1. ADD, SUB, MUL, IDIV, LT, GT, EQ, AND, OR, STRI2INT, GETCHAR, SETCHAR, CONCAT (3 operandy) <var> <symm1> <symb2>
   2. JUMPIFEQ, JUMPIFNEQ, <label> <symm1> <symb2>

1. **2 operandy**
   1. INT2CHAR, STRLEN, TYPE, MOVE, NOT <var> <symb>
   2. READ, <var> <type>

1. **1 operand**
   1. DEFVAR, POPS (1 operand) <var>
   2. PUSHS, WRITE, DPRINT <samb>
   3. CALL, LABEL, JUMP <label>
2. **0 operandů**
   1. BREAK, CREATEFRAME, PUSHFRAME, POPFRAME, RETURN

Nejdůležitější čast parsování se odehrává ve funkci instruction\_control, kde se nachází konečný automat, který zkontroluje instrukci podle toho, do jaké patří kategorie a data zpracuje do XML. V main pak inicializuji XML, procházím program řádek po řádku a tvořím si statistiky pro rozšíření *STATP*.

**Interpret.py – postup řešení**

Řešený pomocí 4 tříd: *SyntaxParse* (provede syntaktickou kontrolu vstupu), *Frame* (definuje rámec), Interpret (zpracovává instrukce, interpretuje výstup třídy *SyntaxParse*) a *Main*.

Třída **SyntaxParse** funguje stejně jako parser v první úloze. Mám zde stejné rozdělení do kategorií a podobný, trochu vylepšený konečný automat. Vytváří se zde struktura structTree, která vypadá například takto:

{1: {'arg': [{'type': 'label', 'text': 'MAIN'}], 'instruction\_name':JUMP'},

2:{'arg': [{'type': 'var', 'frame': 'LF', 'name': 'counter', 'text': 'LF@counter'}], 'instruction\_name': 'WRITE'}

3: {'arg': [], 'instruction\_name': 'END'} }

Jedná se o slovník slovníků ukončený instrukcí s názvem *END*, kde klíčem je pořadí instrukce. Významy položek by měly být jasné z názvů (arg – argumenty, instruction\_name – jméno instrukce atd.). Vytvořená struktura je pak předána Interpretu, aby ji zpracoval.

Interpret prochází strukturu 2x: jednou prohledává a ukládá si všechna návěští (realizuje metoda *findAllLabels*) a podruhé dochází k zpracování instrukcí (instrukce *proceed*). Obě funkce si na základě jména instrukce zavolají funkci realizující význam instrukce a dojde k provedení.

Strukturu rámce mám ve třídě *Frame*. Na začátku (v *Interpretu*) si vytvořím prázdný globální rámec a na základě instrukcí v zadání vytvářím nové rámce, přepisuji je a kontroluji definovanost.

**Poznámka k nejasnosti zadání**

Moje implementace počítá s tím, že na datový zásobník dávám pouze inicializované proměnné, jinak vracím chybu 56.

**Test.py - implementace**

Script obsahuje 2 třídy (*Help* – vypíše nápovědu, *Tests*) a funkci *main*. Nejzajímavější je třída *Tests*, která obsahuje metodu *get\_tests*. Tato metoda projde všechny složky daného adresáře (rekurzivně/nerekurzivně v závislosti na argumentech programu). Metoda volá *get\_tests\_src*, což je pomocná metoda, která případně vytváří pomocné dočasné soubory, vytváří zdrojové soubory, které nenalezneme v adresáři a vytváří HTML stránku s výsledky.