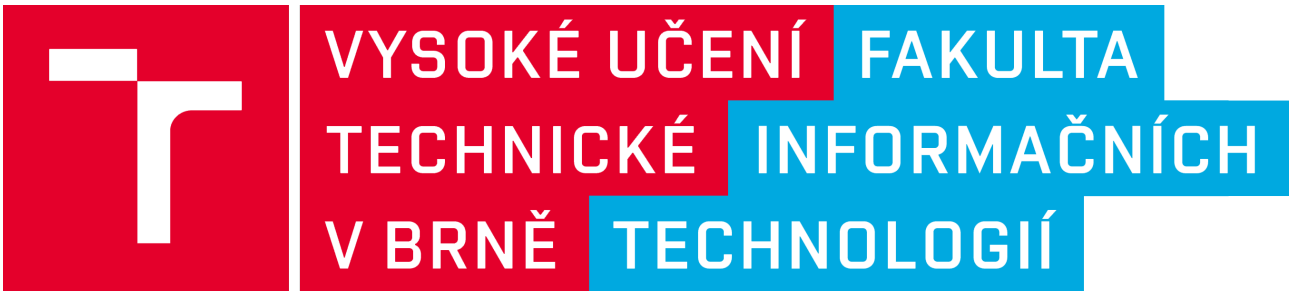
Vysoké učení technické v Brně
Fakulta informačních technologií



Principy programovacích jazyků a OOP
Projekt: 2. Časť

Martin Bublavý (xbubla02)

27. marca 2022

Úloha:

Cieľom bolo vytvoriť skript **interpret.py**, ktorý interpretuje XML reprezentáciu kódu v jazyku IPPcode22.

Vypracovanie:

Hlavný skript **interpret.py** vykonáva tieto operácie:

1. Overenie správnosti XML reprezentácie
2. Spracovanie a overenie správnosti argumentov na vstupe
3. Načítanie a uloženie XML reprezentácie pomocou knižnice ElementTree
4. Načítanie a uloženie inštancií objektu inštrukcie a argumentov do zoznamu inštrukcií
5. Spracovanie inštrukcíi zo zoznamu inštrukcií a vykonanie syntaktickej a sémantickej analýzy
6. Nájdenie escape sekvencií v reťazcoch a nahradenie príslušňymi znakmi
7. Interpretácia kódu

Slovný popis funkčnosti:

Interpretácia XML reprezentácie sa vykonáva vo funkcii **Process_Instructions(instr_list)**, ktorá berie ako parameter zoznam inštrukcií naplnený objektami inštrukcií. Zoznam inštrukcií je ešte pred spracovaním zoradený podľa atribútu **order** a sú nájdené všetky inštrukcie **LABEL**, ktoré označujú návestia, ktoré sú následne uložené do zoznamu návestí spolu s ich inkrementovaným indexom v poli inštrukcií. Následne sa inštrukcie spracovávajú jedna po jednej inkrementovaním hodnoty **instr_counter**. Pri každej inštrukcií sa kontroluje počet, typy a hodnoty argumentov, existencia premenných alebo samotných hodnôt pomocou funcií **Check_Var()** a **Check_Types()**. Argumenty inštrukcie sa získavajú pomocou funkcie **Get_Vars()**, ktorá vracia pole objektov premenných alebo symbolov, podľa toho či boli argumenty premenné alebo samotné hodnoty (napr. int@25). Ak sa jedná o premennú tak sa kontroluje či existuje. Pri inštrukcii **READ** sa berie vstup zo súboru, ak bol pri spustení špecifikovaný parameter –input=. V opačnom prípade berie vstup zo štandartného vstupu. Pre inštrukciu **JUMP** alebo jej variácie sa kontroluje existencia návestia v zozname návestí a následne sa mení index inštrukcie na spracovanie na hodnotu indexu pre toto návestie. V prípade uloženia hodnoty typu **string** do hodnoty premennej alebo symbolu (string@something) sa tento reťazec prehľadá a nájdu sa všetky escape sekvencie, ktoré sú následne nahradené príslušnými znakmi.

Triedy:

Error:
Malá trieda pre uchovávanie error kódov. Implementuje funkciu **error_out** pre výpis správy na STDERR a ukončenie skriptu s príslušným kódom.

Instruction:
Reprezentácia inštrukcie v IPPcode22.

- *opcode*: Meno inštrukcie
- *order*: Poradie inštrukcie v kóde
- *arguments*: Zoznam inštancií objektu argumentu inštrukcie

Argument:
Reprezentácia argumentu v IPPcode22.

- *type*: Typ argumentu (var, int, string....)
- *order*: Poradie argumentu v príslušnej inštrukcii
- *content*: Obsah argumentu (hodnota)

Stack:
Reprezentácia zásobníku v jazyku python.

- *stack*: Zoznam s funkčnosťou zásobníku

var:
Reprezentácia premennej v IPPcode22.

- *name*: Meno premennej
- *value*: Hodnota premennej
- *type*: Typ premennej

symbol:
Reprezentácia hodnoty (napr. int@25, string@something ...).

- *value*: Hodnota (25, something....)
- *type*: Typ hodnoty (int, string ...)

label:
Reprezentácia návestia.

- *name*: Názov návestia
- *instr_index*: Inkrementovaný index inštrukcie

frame:
Reprezentácia rámca v IPPcode22.

- *var_list*: Zoznam premenných v danom rámci

Implementované rozšírenia:

FLOAT:

Hodnoty typu float sa ukladajú v hexadecimálnom zápise. V prípade, že inštrukcia vykonáva operácie s typom float tak volá funkciu **Get_Float**, ktorá overí správnosť hodnoty a vracia hexadecimálny zápis tejto hodnoty ak nebol špecifikovaný nepovinný argument **return_type**. V prípade, že bol nastavený na hodnotu **'nohex'** vracia číslo v desiatkovej sústave. Konverzie z desiatkovej sústavy do hexadecimálneho zápisu a naopak sa vykonávajú pomocou funckií **float.fromhex** a **float.hex**.

STACK:

Implementované podobne ako nezásobníkové inštrukcie. Rozdiel spočíva v tom, že hodnoty sú zo zásobníku brané v opačnom poradí a výsledok sa neukladá do premennej ale naspäť na zásobník.

Test.php:

Cieľom bolo vytvoriť skript **test.php**, ktorý testuje funkčnosť skriptov **parse.php** a **interpret.py**. Výsledky testov vypisuje na štandartný výstup v HTML kóde.

Vypracovanie:

1. Spracovanie argumentov na vstupe
2. Nájdenie a uloženie všetkých ciest k '.src' súborom z adresára zadaného užívatelom alebo súboru daným parametrom "**testlist**". V prípade argumentu "**match**"sa taktiež kontroluje či názov súboru vyhovuje zadanému regulárnemu výrazu.
3. Pre každý súbor:
   (a) Vypísanie cesty k súboru na výstup
   (b) Ak existuje '.rc' súbor, načítanie jeho obsahu, inak vytvorenie tohoto súboru a zapísanie hodnoty "0"
   (c) Ak neexistujú súbory '.out' alebo '.in', tak vytvorenie týchto súborov
   (d) Podľa parametrov na vstupe spustí parser.php alebo interpret.py alebo obydve
   (e) Overenie či sa návratový kód zhoduje s očakávaným a prípadné porovnanie výstupov
   (f) Výpis výsledku do HTML tabulky
   (g) Ak nebol použitý argument –noclean, zmaže dočasné súbory
4. Výpis ukončenia HTML kódu

Triedy:

HTML_gen:
Vypisuje rôzne časti HTML kódu na výstup.

Implementované rozšírenia:

FILES:

Pri použití argumentu "**testlist**"sa cesty k súborom a priečinkom načítajú do poľa. Následne funkcia **Get_Files_from_file** prechádza toto pole. V prípade že sa jedná o priečinok, načíta jeho obsah do poľa a rekurzívne volá túto fukciu s parametrom **recurse_counter** nastaveným na hodnotu 1, v prípade že nebol použitý argument '**recursive**', inak 0. Táto hodnota zaisťuje, že ak bol v súbore testlist zadaný priečinok, ktorý obsahuje ďalšie priečinky tak nebudú prehľadaváné (ak nebol použitý argument 'recursive'). V prípade použitia argumentu '**match**' sa názvy súborov overia pomocou funckie **preg_match**, ktorá overí či vyhovujú zadanému regulárnemu výrazu.