Implementačná dokumentácia k 2. úlohe do IPP 2022/2023

Meno a priezvisko: Adam Pap

Login: xpapad11

Skript `interpret.py` sa začína príkazom `if __name__ == '__main__':` ktorý volá funkciu `main_fun()`, ktorá volá triedu `Main_interpret()`. V triede `Main_interpret()` sa následne rieši spracovanie argumentov, ktoré skript dostal pri spustení, kontrola XML kódu, ktorý dostal či už z `.xml` súboru alebo zo štandardného vstupu `STDIN`, spracovaním argumentov jednotlivých inštrukcií a v neposlednom rade zostrojenie zoznamu inštrukcií a ich následné spracovanie/interpretovanie zavolaním funkcie `execute_all()`. Tiež tu sú zadefinované rôzne polia, listy, s ktorými sa neskoršie ďalej pracuje, ako sa prechádza XML kódom a vytvárajú sa objekty jednotlivých argumentov, ktoré sú potom zaobalené jednotlivými objektami inštrukcií.

Avšak ako prvé sa najprv musia spracovať argumenty, ktoré užívateľ predal skriptu pri jeho spustení. Skript `interpret.py` sa spúšťa s dvomi argumentami a to buď: `--source=súbor.xml`, alebo `--input=súbor`. Avšak vždy musí byť zadaný aspoň jeden z týchto dvoch spomenutých parametrov, inak dôjde k chybe. V prípade ak nezadáme buďto parameter `--input` alebo `--source` , bude obsah nezadaného parametru očakávaný na štandardnom vstupe `STDIN`. V prípade potreby, si užívateľ vie nechať vypísať prepínačom `--help` nápovedu na spustenie daného skriptu. Avšak s prepínačom `--help` nesmie byť uvedený žiadny iný argument, inak skript vyhodí chybu 10. Spracovanie argumentov bolo vykonané pomocou `python` knižnice `argparse`.

Po úspešnom spracovaní argumentov skriptu dochádza k samotnému spracovaniu XML kódu, ktorý, už ako bolo spomenuté, sa získava buďto zo `STDIN` alebo z `.xml` súboru. Všetky tieto kontroly sú vykonávané funkciou `start_interpreting()`. Pomocou knižnice `xml.etree.ElementTree` sa získa koreň daného XML stromu, cez ktorý sa potom iteruje a kontrolujú sa veci ako napríklad: správnosť tagu `<program></program>` a jeho atribútov, menovite verzia kódu (IPPcode23), následne sa v ďalšom `for` cykle kontroluje správnosť jednotlivých inštrukcii a ich atribútov (správnosť `opcode`, a prítomnosť `order` atribútu prípadne či sa dané číslo `order` už nevyskytovalo predtým). V tomto cykle sa vytvára objekt typu `instruction`, ktorý ako parametre do konštruktora `__init__` berie `opcode`, číslo `order`, rámec, a v neposlednom rade parameter `self`, ktorý slúži na odkazovanie sa na inštanciu triedy `Main_interpret()`.

V ďalšom vnorenom `for` cykle sa skript zanoruje hlbšie do XML stromu a kontroluje jednotlivé argumenty daných inštrukcií. Kde sa opäť kontrolujú atribúty `<arg><arg>` tagu, kde sa kontroluje či typ, ktorý argument nesie, patrí medzi známe typy a v neposlednom rade sa kontroluje aj `label`, hlavne či sa daný `label` sa neopakuje v rámci daného XML kódu, inak to vedie na chybu 52. V prípade ak nejaká zo spomínaných kontrol zlyhá, a nie je explicitne uvedený chybový kód, vedie to na chybu 32.

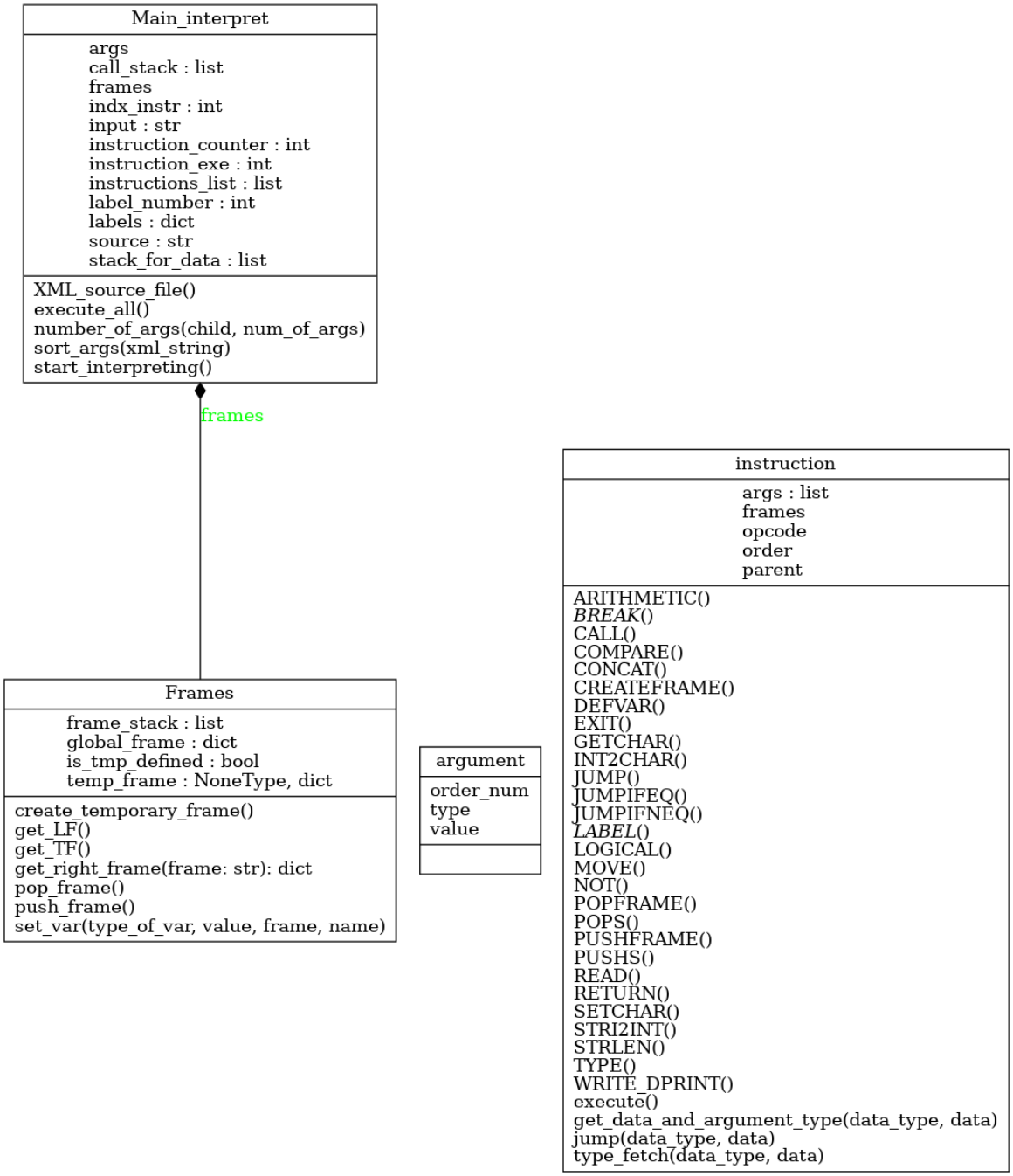
Po úspešnej kontrole správnosti `<arg><arg>` tagov sa vytvárajú objekty jednotlivých argumentov spracovávanej inštrukcie typu argument, ktorý ako parametre do konštruktora `__init__` berie dátový typ daného argumentu (`type`), hodnotu vo vnútri tagov, a `order` číslo inštrukcie ku ktorému patrí. Po úspešnom vytvorení jednotlivých inštancií daných argumentov sa pridajú do práve vytvorenej inštancie `instruction`. Po dobehnutí vnoreného `for` cyklu (ktorý zabezpečoval spracovanie týchto argumentov danej inštrukcie) sa práve vytvorená inštancia `instruction` pridá do finálneho listu inštrukcií, ktorý sa po prejdení celého XML kódu a vytvorení jednotlivých inštrukcií začne

interpretovať. Avšak ešte pred začiatkom interpretácie sa jednotlivé inštancie `instruction` zoradia podľa čísla inštrukcie (`order`).

Po vytvorení a zoradení listu inštrukcií sa prechádza k samotnej interpretácií jednotlivých inštrukcií a to zavolaním funkcie `execute_all()` z konštruktora triedy `Main_interpret()`. V tejto funkcií sa postupne iteruje cez daný list inštrukcií. Zavolaním funkcie `execute()` z triedy `instruction` sa daná inštrukcia interpretuje na základe jej `opcode`.

Samotná interpretácia prebieha spôsobom, že najprv sa cez `if elif` príkazy zistí na základe `opcode` o akú inštrukciu sa jedná. Potom sa len zavolá funkcia, ktorá danú inštrukciu spracuje, získa jej parametre z argumentov, ku ktorým vieme pristúpiť cez danú inštanciu triedy `argument` ktorá sa v danej inštancií inštrukcie nachádza, a tiež rámec nad ktorým má byť daná inštrukcia spracovaná. Získanie týchto kľúčových parametrov zabezpečuje funkcia `get_data_and_argument_type(argument type, argument value)` ktorá, ako je možné vidieť berie dva parametre typ argumentu (`nil`, `int`, `string`, `bool`, `var` apod.) a hodnotu, ktorý daný argument nesie. V prípade že typ argumentu je premenná (`var`) zistí sa na akom rámci pracuje a pomocou funkcie `get_right_frame(frame)` s parametrom `frame`, čo reprezuntuje typ rámca GF, TF, LF, z triedy `Frames` sa vráti príslušný rámec z rámcového zásobníku (frame stack) s ktorým sa potom ďalej pracuje. Samozrejme funkcia kontroluje či sa vôbec príslušný rámec našiel a či hodnota existuje na danom rámci alebo nie. Po získaní týchto hodnôt sa vykoná daná inštrukcia a jej výsledok (ak nejaký má) sa vloží na príslušný rámec.

Čo sa triedy `Frames` týka tak prakticky táto trieda obsahuje celý pamäťový model jazyka IPPcode23. V konštruktore sa vytvára prázdny rámcový zásobník, prázdny GF a TF. Samotná inštancia rámca sa vytvára po spustení skriptu v konštruktore triedy `Main_interpret()`.


Main_interpret
args
call_stack : list
frames
indx_instr : int
input : str
instruction_counter : int
instruction_exe : int
instructions_list : list
label_number : int
labels : dict
source : str
stack_for_data : list
XML_source_file()
execute_all()
number_of_args(child, num_of_args)
sort_args(xml_string)
start_interpreting()
frames
Frames
frame_stack : list
global_frame : dict
is_tmp_defined : bool
temp_frame : NoneType, dict
create_temporary_frame()
get_LF()
get_TF()
get_right_frame(frame: str): dict
pop_frame()
push_frame()
set_var(type_of_var, value, frame, name)
argument
order_num
type
value
instruction
args : list
frames
opcode
order
parent
ARITHMETIC()
BREAK()
CALL()
COMPARE()
CONCAT()
CREATEFRAME()
DEFVAR()
EXIT()
GETCHAR()
INT2CHAR()
JUMP()
JUMPIFEQ()
JUMPIFNEQ()
LABEL()
LOGICAL()
MOVE()
NOT()
POPFRAME()
POPS()
PUSHFRAME()
PUSHS()
READ()
RETURN()
SETCHAR()
STRI2INT()
STRLEN()
TYPE()
WRITE_DPRINT()
execute()
get_data_and_argument_type(data_type, data)
jump(data_type, data)
type_fetch(data_type, data)