Implementačná dokumentácia k 2. úlohe do IPP 2022/2023

Meno a priezvisko: Matej Keznikl

Login: xkezni01

Popis implementácie skriptu interpret.py:

Skript ako prvé skontroluje správnosť argumentov skriptu. V prípade, že bol zadaný nesprávny argument alebo bolo zadané väčšie množstvo argumentov, skript vráti chybovú návratovú hodnotu 10. resp. 11 v prípade chybnei kombinácie argumentov. Overenie argumentov skriptu má na starosti trieda Arguments. Po kontrole argumentov nasleduje spracovanie vstupu v podobe XML reprezentácie kódu. Spracovanie XML je vykonávané v triede xml, pričom je na spracovanie používaná knižnica xml.etree. Element Tree. Následne je spracované XML skontrolované na lexikálne a syntaktické chyby, ktoré mohli byť spôsobené užívateľom. V prípade negatívneho výskytu syntaktických a lexikálnych chýb je XML rozparsované na inštancie triedy Instruction, ktoré sú následne vložené do zoznamu inštrukcií. Zoznam inštrukcií je taktiež nutné zoradiť podľa argumentu order daných inštrukcií, čo je vykonané metódou triedy xml, sort(). Po skontrolovaní argumentov skriptu a spracovaní XML reprezentácie kódu nasledovala samotná interpretácia jednotlivých inštrukcií v rámci triedy Interpret. V rámci statických kontrol boli dopredu pozisťované návestia, ošetrenie ich redefinície, pričom v prípade chyby skript vráti chybovú návratovú hodnotu 52. Interpretáciu inštrukcií vykonáva metóda handleInstructions() triedy Interpret, kde sa v rámci while cyklu interpretujú inštrukcie zo zoznamu Instruction list podľa premennej instruction counter, ktorá v prípade skoku skočí na inštrukciu za návestím. Samotná interpretácia inštrukcie je volanie inštančnej metódy triedy Instruction, ktorá volá konkrétnu metódu (statickú alebo inštančnú) podľa typu inštrukcie, kde inštrukcie môžu byť rozdelené do tried Arithmetic, Relational, Logical, String, ProgramFlow, IO, Stack a Frame.

Popis jednotlivých tried:

Trieda Arguments

Trieda, ktorá má na starosti spracovanie a overenie vstupných argumentov skriptu. Obsahuje metódy *Source* a *Input*, ktoré vracajú popisovače (file descriptors) vstupných súborov.

Trieda Exit

Trieda má na starosti ukončenie programu s korektným návratovým kódom interpreta a taktiež obsahuje všetky potrebné návratové kódy.

Trieda XML

Trieda, ktorej úlohou je spracovanie XML reprezentácie kódu zo vstupu. Obsahuje metódu *check*, ktorá skontroluje syntaktickú a lexikálnu korektnosť vstupnej XML reprezentácie kódu, metódu *arg_count*, ktorá vracia počet argumentov jednotlivých inštrukcií a v prípade chybného počtu ukončí program s korektným návratovým kódom, metódu *parse*, ktorej úlohou je rozparsovanie inštrukcie a jej atribútov na inštancie tried Instruction, pričom je výsledná inštancia uložená do zoznamu inštrukcií Instruction_list a metódu *sort*, pričom jej úlohou je zoradenie inštrukcií podľa argumentu order.

Trieda Interpret

Trieda má na starosti interpretáciu jednotlivých inštrukcií. Obsahuje metódu *handleInstructions* v rámci ktorej sa volajú jednotlivé inštrukcie určené na interpretáciu a metódu *handleLabels*, ktorej úlohou je spracovanie návestí a v prípade redefinície návestí ukončenie programu s korektným chybovým návratovým kódom.

Triedy obsahujúce metódy, ktoré implementujú jednotlivé inštrukcie:

- o Trieda Arithmetic aritmetické operácie
- o Trieda Relational relačné operácie
- o Trieda Logical logické operácie
- o Trieda String operácie nad reťazcami
- o Trieda ProgramFlow operácie pre zmenu toku programu
- o Trieda IO vstupno-výstupné operácie
- o Trieda Stack operácie nad dátovým zásobníkom, vrátane funkcií z rozšírenia STACK
- o Trieda Frame operácie nad rámcami

Trieda Var

Trieda, ktorá reprezentuje premennú, tzn. obsahuje meno, rámec, typ a hodnotu premennej.

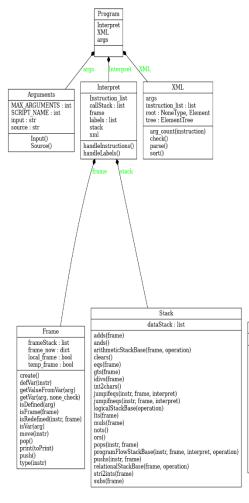
Trieda Constant

Trieda reprezentuje konštantu v zásobníku, tzn. obsahuje typ a hodnotu.

Rozšírenia

Z rozšírení boli implementované rozšírenia STACK a FLOAT, ktoré umožňujú interpretu interpret.py podporu funkcií operujúcich s plávajúcou desatinnou čiarkou a taktiež implementujú zásobníkové verzie inštrukcií z klasického zadania

UML diagram:



Arithmetic add(instr, frame) base(instr. frame, operation) div(instr, frame) float2int(instr, frame) getDecFloat(frame, instruction argument) getFloatFromString(frame, instruction argument) getHexFloat(frame, instruction_argument) getIntValue(frame, instruction_argument, readFlag) hexToDec(number) idiv(instr, frame) int2float(instr, frame) isNone(number) mul(instr. frame) octaToDec(number) setNone(variable, type) stringToInt(number) sub(instr. frame)

Exit EXIT_FRAME : int EXIT_INPUT : int EXIT OPERAND : int EXIT_OUTPUT : int EXIT_PARAM : int EXIT_SEMANTIC : int EXIT STRING : int EXIT_SUCCESS : int EXIT_TYPE : int EXIT_VALUE : int EXIT VARIABLE : int EXIT XML FORMAT : int EXIT XML STRUCTURE : int

Constant

type value

IO breakl(instr, frame) dprint(instr, frame) exit(instr, frame) handleString(value) read(instr, frame, xml) writel(instr, frame, output stream) eqs(frame, stack) exit(frame) float2int(frame) getchar(frame) gt(frame) gts(frame, stack) idiv(frame) idivs(frame, stack) int2char(frame) int2chars(stack) int2float(frame) jump(interpret) jumpifeq(frame, interpret) jumpifeqs(frame, interpret, stack) jumpifneq(frame, interpret) jumpifneqs(frame, interpret, stack) lt(frame) lts(frame, stack) move(frame) mul(frame) muls(frame, stack) notI(frame) nots(stack) orI(frame) ors(stack) popFrame(frame) pops(frame, stack) pushFrame(frame)

pushs(frame, stack) read(frame, xml) setchar(frame) stri2int(frame) stri2ints(frame, stack) strlen(frame) sub(frame) subs(frame, stack) writeI(frame)

Instruction arg1 arg2 arg3 aras opcode order

add(frame) adds(frame, stack) andI(frame) ands(stack) breakI(frame) clears(stack) concat(frame) createFrame(frame) defVar(frame)

div(frame) dprint(frame)

eg(frame)

Logical andI(instr, frame)

andImpl(value1, value2) base(instr, frame, operation notI(instr. frame) notImpl(value) orI(instr, frame) orImpl(value1, value2)

ProgramFlow jump(instr, interpret) jumpifBase(instr, frame, interpret, operation) jumpifeq(instr, frame, interpret) jumpifneq(instr, frame, interpret)

base(instr, frame, operation) eq(instr, frame) gt(instr, frame) lt(instr. frame)

Relational

frame type : NoneType value : NoneType stri2int(instr, frame) strlen(instr, frame)

String var concat(instr, frame) getchar(instr, frame) int2char(instr, frame) setchar(instr, frame)