Implementačná dokumentácia k 2.úlohe do IPP 2020/2021

Meno a priezvisko: Jakub Sokolík
Login: xsokol14

1 Cieľ úlohy

Cieľom tejto úlohy bolo implementovať interpret pre jazyk IPPcode21 a test pre skripty z úlohy 1 a 2. Implementácia riešenia sa nachádza v súboroch **interpret.py**, **test.php**, **File.php**. **Tester.php**, **HtmlGen.php**. K úlohe bolo implementované rozšírenia **FLOAT**, **STACK** a **STATI**, ktoré sú súčastou skripti **interpret.py** a rozšírenie **FILES**, ktoré je súčasťou skriptu **test.php**.

2 Skript interpret.py

2.1 Spustenie skriptu interpret.py a kontrola prepínačov

Chovanie skriptu ovplyvňujú argumenty, s ktorými je skript spustený. Pomocou prepínača *–help* je možné na štandardný výstup vypísať hlásanie, takzvaný Usege, ktorý popisuje spustenie skriptu a funkciu jednotlivých prepínačov. Skript podporuje všetky prepínače, z rozšírenia **STATI**.

Pokiaľ je pri volaní skriptu zadaný prepínač *–help*, nie je kontrolované nič a skript vracia návratovú hodnotu 0 spolu s hlásením uvedením vyššie. Prepínače nemajú striktné poradie, a tak sa kontroluje iba ich existencia, ktorá potom ovplyvňuje chovanie programu. Povinný je jeden prepínač z dvojice *source*, *input*. V prípade, že jeden nie je zadaný, dáta sa načítavajú zo súboru. V prípade abstinencie oboch prepínačov je program ukončený s chybou 10. Rovnako na chybu 10 vedie použitie prepínačov *–insts*, *–hot* alebo *–vars* bez predchádzajúceho prepínača *–stati*.

2.2 Pársovanie XML vstupu a jeho validácia

Na spracovanie xml vstupu bola vytvorená funkcia *checkXML()*, ktorá zoradí inštrukcie podľa atribútu *order*, a argumenty podľa tagu. Následne prechádza všetkými inštrukciami a kontroluje atribút *order*. Za účelom uchovania inštrukcii v nejakom rozumnom formáte bola vytvorená trieda *instruction*. Táto trieda si pamätá meno inštrukcie, poradie a jej argumenty. Počas pársovania xml vstupu sú všetky inštrukcie ukladané do zoznamu. Tak isto je vytvorený slovník pre návestia, ktorý si pamätá umiestnenie definície daného návestia v zozname inštrukcií.

Pri spracovaní argumentov, sa kontroluje, či je argument validného typu, a či jeho hodnota zodpovedá danému typu. Následne je hodnota konvertovaná na správny typ (int, float). Za účelom uchovania argumentov, bola vytvorená trieda *arg*, ktorá uchováva typ a hodnotu argumentu.

2.3 Vykonávanie inštrukcii

Trieda *controller* bola vytvorená na vykonávanie inštrukcii. Každej inštrukcii odpovedá jedná metóda. Celý program prechádza vyššie uvedený zoznam inštrukcií. Podľa mena inštrukcie je potom zvolaná metóda controllera, ktorá odpovedá danej inštrukcii. Nakoľko všetky aritmetické funkcie majú rovnaké argumenty, bola pre ne vytvorená jedna spoločná metóda *arithmetic*.

Vždy na začiatku metódy sú skontrolované argumenty inštrukcie. Pre uľahčenie kontroly boli v triede *arg* implementované kontrolné metotódy *isSym()*, *isvar()*, *isLabel()* a *isType()*. Následne má samotný *controller* implementovanú metódu *getAll()*, ktorá z daného argumentu vráti typ a hodnotu. Pokiaľ ide o premennú, vráti typ a hodnotu z premennej. Následne sa skontroluje, či vstupné sú vstupné typy a hodnoty správne a vykoná sa nad nimi daná inštrukcia.

2.4 Pamäťový model

V triede *controller* sú vytvorené slovníky pre jednotlivé rámce. Podľa mena premennej tak môžeme pohodlne pristupovať jej hodnotám. Na reprezentáciu premenných bola vytvorená ďalšia trieda *variable*, ktorá uchová typ a hodnotu v danej premennej. Na získanie rámca a názvu premennej bola vytvorená pomocná funkcia *parseVar()*. Lokálny rámec reprezentuje zoznam slovníkov. Nové rámce sa pridávajú vždy na koniec a pristupuje sa vždy iba k poslednému rámcu.

Podobne ako lokálny rámec, aj dátový zásobník a zásobník volaní sú reprezentované zoznamom, kde sa pridáva vždy na koniec a vyberá z konca. Pri vkladaný na dátový zásobník, sa vytvára nová insťancia triedy

variable, pretože dokáže jednoducho uchovať typ a hodnotu. Na zásobník volaní sa ukladá poradie inštrukcie v zozname inštrukcii.

2.5 Rozšírenie FLOAT

Implementácia rozšírenia bola jednoduchá. Stačilo vytvoriť nové metódy pre inštrukcie *INT2FLOAT*, *FLOAT2INT* a *DIV*. v ostatných metódach boli iba rozšírené kontrolné podmienky, aby akceptovali typ float.

2.6 Rozšírenie STACK

Na začiatku každej metódy, ktorú podporuje toto rozšírenie bola pridaná podmienka. Ak posledné písmeno mena inštrukcie je 'S' jedná sa stack inštrukciu a nepracuje sa s parametrami, ale s dátami na dátovom zásobníku.

2.7 Rozšírenie STATI

Pre zbieranie inštrukcii bola do triedy *controller* doplnená metóda, ktorá počíta štatistiky. Volá sa pre každú inštrukciu. Po vykonaní všetkých inštrukcii, sa tieto štatistiky, vypíšu do súboru.

3 Skript test.php

3.1 Spustenie skriptu test.php a kontrola prepínačov

Tak ako pri skripte *interpret.py*, je podporovaný prepínač *–help*. Prepínače nemajú striktné poradie a skript podporuje všetky prepínače, z rozšírenia **FILES**.

Rozšírenie **FILES** zakazuje použitie *–dictionary* súčasne s *–testlist*, v prípade tejto kombinácie je skript ukončený s chybou 10. Na kontrolu existencie súborov bola vytvorená funkcia **CheckFiles()**. V prípade, že zadaný súbor alebo adresár neexistuje vracia chybu 41. Skript pracuje s absolútnou aj relatívnou cestou k súborom.

3.2 Štruktúra skriptu test.php

Súbor **test.php** slúži prevažne na kontrolu prepínačov a vytváranie inštancii jednotlivých tried. Dôležitá je funkcia **SortTestList()**, ktorá z prepínača *–testlist*, vytvorí pole súborov a priečinkov.

V súbore **File.php** je implementovaná trieda **Search**, jej inštancia sa vytvára zvlášť pre každý priečinok z vyššie uvedeného poľa. Funkcia **FindFiles()** vráti všetky súbory s koncovkou *.src*, ktoré sú následne pridané do poľa súborov, tak aby sa testy neopakovali. Súbor **File.php** tiež obsahuje triedu **File**, ktorej inštancia sa vytvára nad každým súborom z poľa. Pri vytvárani inštancie sa skontroluje, či k danému súboru s príponou *.src* existujú súbory *.in .out a .rc*. Ak chýbajú, sú vytvorené.

V súbore **Tester.php** je implementovaná trieda **Tester**, ktorá slúži na spúšťanie skriptov nad súbormi. Obsahuje 3 verejné triedy, každá testuje inú kombináciu skriptov a spúšťa generovanie html div blokov pre daný test. Konkrétne **ParseOnly()**, **InterpreterOnly()** a **TestBoth()**. Ďalej trieda obsahuje privátnu funkciu, **GetName()** ktorá vytvorí názov testu; **Parser()** a **Interpreter()**, ktoré spúšťajú skripty; **CheckXml()** a **CheckIntOut()**vytvárajú diff súbory; **DiffProces()** upravuje diff súbory, tak aby boli vhodné pre jazyk html a dali sa vypísať; **DeleteTmpFile()** maže všetky pomocné súbory.

V súbore **HtmlGen.php** je implementovaná trieda **HtmlGen**, ktorá sa stará o generovanie výsledného kódu. Skladá sa zo statických funkcii: **BuildHead()**, ktorá vygeneruje hlavičku s definíciou css štýlu; **OkDiv**, **ErrkDiv** a **DiffDiv()**, ktoré generujú jednotlivé div bloky pre úspešné aj neúspešné testy; **GetHtml** zostaví všetky časti reportu do hromady. Aby bolo možné všetky chybné testy zapísať na vrchu, sú v **parse.php** vytvorené stringy htmlpass a htmlfail, kde sa postupne zhromaždujú úspešné a neúspešné testy.

3.3 Rozšírenie FILES

Rozšírenie podporujúce prepínač *–regex*, nebolo takže implementovať a dokázali sme to pridaním pár podmienok. Avšak pre prepínač *–testlist* bolo nutné vytvoriť funkcie **SortTestList()**. Predpoklad je taký, že vstupné pole obsahuje ako súbory, tak aj priečinky, za tým účelom bolo vytvorené pole pre súbory a pre priečinky a vyššie uvedená funkcia roztriedi pôvodné pole. Následne nad každý priečinkom bola vytvorená inštancia triedy **Search** a funkcia **FindFiles()** vrátila všetky súbory s príponou *.src*. Tie následne boli pridané do poľa testov tak, aby sa tam žiadny test neopakoval. Tento proces sa opakuje aj v prípade, že skript **test.php** je spustený bez prepínača *–testlist*. V takom prípade pole súborov obsahuje jediný prvok.