Documentation of Second Project Implementation for IPP 2018/2019
Name and surname: Adrián Tulušák
Login: xtulus00

Interpret.py

1. Source code and input
   Source code and input are taken from *stdin* or from *file* depending on arguments and then they are assigned to variables that arrange better work with them. Source code is then transformed from XML format into my own format which is made of dictionaries. During this is performed basic inspection of XML file. This transformation is done for better work and better calling functions with arguments consisting of instructions and operands from source code. Input is not changed and whenever it is needed, script reads one line from it.

2. Arguments
   Arguments are checked in cycle and then are put into dictionary with value true or false depending on whether it was used or not. Some arguments that needs to be used in order in which they were given have special variable determining their order.

3. Instructions
   First cycle going through source code is looking for all *labels* and saves it into dictionary with name and instruction order number so that script will know where to jump if there is any jump or *call* instruction. Then starts main cycle, but before that is found the highest order number of instruction. The main cycle ends when it reaches this higher number – the last instruction of the source code. In this cycle script gets actual instruction – depending on actual order number (instruction counter) – and calls function to handle instruction and all its operands. At the end of iteration is instruction counter incremented. Instructions goes through a switch and the is called function to handle specific instruction. In every instruction function are at first checked arguments (number, type, value if needed) and then they are handled. Every time any instruction uses frames, particular frame is checked if is not empty and if has required variable.

4. Extensions
   STATI – output file for statistics is saved in variable (just like all others) and if file do not end with run time error, statistics are written into this file. If it does not exist it is created. Statistics are made in main cycle. Counter of executed instructions is increased in every iteration of the main cycle what means it will count labels and instructions in cycles too. After every executed instruction is checked number of initialized variables and is compared to highest count.

Test.php

1. Arguments
   Arguments are checked in cycle just like in other scripts. After checking which arguments are used are checked combinations of arguments (allowed and forbidden). Directory path and parse/interpret file are saved into global variables. Both files are checked if they exists and if directory path is real path. If any of these arguments is missing variables get their implicit values.

2. Executing tests
   After handling arguments script executes all parsing tests. If argument *int-only* was used, script skips this section. Similarly if was used *parse-only* is skipped interpreting tests. Both parts have separate parts for going through directory if argument *recursive* was used and if not. If not, script uses built-in function *glob* to get all files with suffix *.src* and then executes them with *exec*. Returned values from executed file are compared to the expected. Parsing and interpreting tests has their own comparing functions because of different output formats. If *recursive* was used, script has recursive function to get all files with theirs paths. Function returns array of paths to files and then is used the same process like without *recursive*. Comparing outputs of executed parsing tests uses temporary files. *Exec* puts output of *jexamxml.jar* into temporary file and then script reads from this file and tries to find there string "*Two files are identical*". There are built-in functions *md5* and *md5_file* used for comparing interpreting tests outputs.

3. HTML output
   HTML output is concatenated into one string and at the end it is written to the *stdout*. Header of HTML is created at the beginning of the script and footer at the end of it. Each executed test concatenates its own statistics into the string. If test was successful, is colored green. If one of comparisons was not successful (return value or output) test is colored red.