Documentation of Project Implementation for IPP 2018/2019
Name and surname: Adam Hostin
Login: xhosti02

1. Interpret.py

1.1. Options

If --help option is used script will display basic description of the script`s options and ends with exit code zero. Other options which are used are --source, which sets a file with a source code for interpret, and --input, which sets a file with an input for interpreted code. Script also includes STATI extension. If any option from STATI is used it is necessary to use option --stats. --stats option sets file where result(s) of other STATI extension options are written. Option --insts writes number of interpreted instructions and option --vars writes number of maximal amount of initialized variables in all frames together at one time. All results from STATI options are written to set file in given order.

1.2. Check functions

`Check_frame` function is used in function `check_var,` has one argument, frame, which is type of string. If frame is equal to GF (global frame), function ends successfully. In case that frame is equal to LF (local frame), if local frame is an empty function ends with exit code 55, otherwise ends successfully. In case that frame is equal to TF (temporary frame), if temporary frame is uninitialized function ends with exit code 55, otherwise ends successfully. If frame does not match any of the above, function ends with exit code 32. `Check_type` is used to check nonterminal <type> and has one argument, which is xml object named argument. Function check if argument.attrib["type"] is equal to type and if argument.text is equal to int, string or bool. If conditions are not returning True, function ends with exit code 32 otherwise ends successfully. `Check_if_bool` is used to decide if list of 2 elements represent bool value. First element of list represents value type and second element represents value. If first element does not match bool, function returns second element of list. If first element does match bool, function returns pythons boolean value (True if second element match true and False if second element match false). `Check_var` function checks if argument is a variable. If function is a variable, list of 2 elements is returned. List consists of variable current data type and value. Function can also exit with exit codes 32 and 54. `Check_symb` function checks if argument is a symbol, which means bool, nil, string, int or variable. Variable is checked as the last option of symbols because `check_var` function is used. `Check_args` function checks if arguments from command line are valid. If invalid, combination of arguments was given, function exits with exit code 10 otherwise returns 1, 2 or 3 to make a difference among --input= and --source= options combinations.

1.3. Other functions

`Create_my_string_from_STDIN` has no argument and read lines from stdin. Function returns string containing input from stdin. `create_my_string_from_file` has one argument option. Option sets from which file function is going to read. Function returns string with content of given file. Function can also exit with exit codes 10 and 11. `Prepare_dicts` function has one argument my_source_string used to create and xml object. Function iterates trough xml object fill instruction_dict containg keys representing orders of instruction and values representing objects from xml. Function also prepares label_dict. Label_dict fills with keys representing labels and values representing orders of given label instruction. Function returns the biggest instruction number found. `defvar_function` function has no argument and handles defvar instruction. Function is able to exit with exit codes 52, 55 and 32.

1.4. Main

At first main create dictionaries, frames and stacks. After that command line arguments are checked and inputs from --source option= and --input option= are processed. Then label_dict and instruction_dict are prepared via `prepare_dicts` function. At last function iterates trough instruction and handles it one by one. If instruction is missing, scripts exit with exit code 32.

2. Test.php

2.1. Options

If --help option is used script will display basic description of the script`s options and ends with exit code zero. Other options which are used are --int-only, which means script starts tests only with interpret, and --parse-only, which means script starts tests only with parse script. Combination of --int-only and --parse-only is not allowed. Option --recursive means that tests are going to be searched for recursively if directory is found. --directory option sets a path with directory where test.php will be searching for tests. Default value is current directory. --int-script sets a path to interpret.py. If --int-script option was not given, default value is „interpret.py“. --parse-script sets a path to parse.php. If --parse-script option was not given default value is „parse.php“.

2.2 Functions

`Check_file` function checks if given argument is a file. It was also used in parse.php. `check_args` function gets array with arguments and checks if script arguments are valid and if one or more argument, which should include directory or file, is missing, function sets its default value. `write_help` function handles --help option. `test_parse_in_globe` and `test_int_in_globe` functions are very similar at first, they create ".out", ".in", and ".rc" if missing. After that, scripts are executed via exec() function. Output of interpret is compared with expected output via JExamXML. JExamXML is executed via exec(). However, `test_parse_in_globe` uses diff instead of JExamXML. After comparing expected output and scripts, output result of test is written to to html variable, which is string, and variables counting failed and passed tests are changed if necessary. `recursion_function` function handles recursive searching for ".src" files. Function glob is used. Function returns array of files. `run_int_tests` and `run_parse_tests` are once again very similar, both of them call function `test_parse_in_globe` or `test_int_in_globe` function, depending on which script is currently tested. If --recursive was given, `recursion_function` is used, function iterates trough array of files returned from `recursion_function`. If --recursive was not given, simple glob is used.

2.3. Main

At first a few global variables are created, and html header is stored in variable named "html". After that, function `check_args` is called. Then script decides, which script is going to be tested. This information is also added to "html". After that `test_parse_in_globe` or `test_int_in_globe` function is called. The final results of tests are written to "html" and foot is added. Finally, script prints "html" to stdout.