Implementation documentation of the 2nd IPP task.
Name and surname: Andrea Chimenti
Login: xchime00

INTERPRET.PY

Class `Interpret`

This is the most important class of the script. It provides most of the essential functionality. In order to successfully execute the interpretation, it is needed to call these methods in exact order as shown below.

```
interpret = Interpret()
interpret.check_root_element()
interpret.create_instructions_array()
interpret.check_syntax()
interpret.search_labels()
interpret.execute()
```

Constructor

The `Interpret.__init__` method is used to create a new instance of the `Interpret` class. Command line arguments passed to the script are parsed and important instance attributes are initiated. Data structures as dictionaries and lists representing frames or labels etc. are initialized.

Instruction list

The XML representation of IPPcode20 is parsed and a list of `Instruction` objects is created. Each instruction in the XML equals to one object in the list. Each object contains all the information of the instruction provided by the XML file, i. e. argument types and values, opcode etc.

Lexical and syntactic analysis

After a list of instructions is created, the list is looped over and every object is checked for lexical or syntactic errors. A new instance of `SyntaxAnalyser` is created for this purpose. Each `Instruction` object is then passed to the `SyntaxAnalyser.check_instruction(self, ins)` function and correctness of every attribute is checked.

Execution

A simple `while` loop iterates over the sorted instruction array and calls the proper function based on the processed instruction.

Architecture drawbacks

Errors occurred during interpretation are handled improperly. When an error occurs no exceptions are thrown, instead the script is immediately terminated (with the right return code). This method was chosen because the script will very probably be never worked on again. A proper error handling and propagation should be used for project of a greater importance.

TEST.PHP

Philosophy

The implementation of `test.php` is straightforward and simple. No special architecture or design pattern is used.

Argument parsing

Arguments are parsed directly in the main body of the script. Each argument corresponds to one global variable as follows:

`$recursive`: bool, `$parse_only`: bool, `$int_only`: bool, `$parse_script`: string, `$int_script`: string, `$jexamxml`: string

Test files list

After that, a list of test files is created using the function `create_file_list($test_directory, &$output_array, $recursive)`. It uses PHP built-in `glob()` function to list files and directories. If recursion flag is set to true, the function `create_file_list()` calls itself when a subdirectory is found.

`TesterInterface`

`TesterInterface` is implemented by `ParseOnlyTester`, `IntOnlyTester` and `BothTester` classes. The interface obliges to implement the `run_test_file($file_path)` method. The function runs only one test (one test file) and should return a `TestResult` object with information about the result.

Main functionality

Depending on the selected testing mode (both, int-only, parse-only) an instance of the proper tester class is created. The test files list is then looped over and the method `run_test_file()` is called on each element. All the results are stored in the `$test_result_list` array.

Output

Once all tests are completed, an html template `template.html` is filled with the results and printed on standard output.