Documentation of Project 2 Implementation for IPP 2022/2023

Name and surname: Matyáš Strelec
Login: `xstrel03`

1 Introduction

The `interpret.py` is a script which takes a source file of an IPPcode23 program, which has been checked and converted to an XML representation by a parser which was implemented in the 1st project, and interprets the source, therefore running the code.

2 Prototype design pattern

For the implementation using OOP, I chose the Prototype design pattern[1]. Here, the `Instruction` class is a prototype for all the sub-classes for each separate instructions. This achieves abstraction of implementation, as well as unifying the execution method of each instruction.

3 Implementation

The program is contained in the `interpret.py` file, most of the code is structured into classes. Important parts of the code have explanatory comments.

3.1 Main function

The main function is simple, as it only instantiates the necessary classes and uses them to begin executing the program itself. The respective classes are described in following sections.

3.2 `InputFiles`

The class is initialized by creating a parser using `argparse`, from which it reads the needed arguments, also printing the help message if prompted. As the script supports the input or source file both from a file using the command line argument, reading from `stdin` also must be supported. Therefore, the correct source for both is decided and the files are opened. Errors while opening or reading the files are also handled here.

3.3 `Xml`

The class takes the input file in form of an XML representation of IPPcode23. Using the `ElementTree` library, XML is easily parsed. The needed attributes of the XML are checked, returning respective errors if necessary.

The opcode is read and it's expected arguments are loaded as an array from the `Helper` class, for each argument, the order, type and value is checked. If they're correct, a object is created which represents the argument, and this object is appended to the instruction.

Using the `eval()` function, an instance of the class with the same name as the opcode is created. The instructions are then ordered by the order tag. The `get_instructions()` method is used to return them.

3.4 `Prog`

The most important class is `Prog`, where the interpretation happens. When initializing, an empty program is created, with instructions, variable storage in frames and the stack and the control flow.

The `run()` method takes given instructions and fills the list. First, all labels are collected to prevent their redefinition. Then, the interpretation begins from the first instruction.

The class contains many useful methods that are used when working with variables. The storage system of frames is used here. For reading and writing into variables, they must be defined first. If they are defined, first, the local frame gets searched, then the temporary frame, and finally the global frame. If the variable is not present, an error gets returned.

3.5 `Instruction`

Each instruction has the `opcode`, `args` and `program` attributes, as well as the `execute()` function. `Args` is a list of `Arg` objects, and `program` is the instance of `Prog` class currently executing the code.

For every opcode, there's a sub-class which has it's own `execute()` method, according to the Prototype design pattern.[1]. The execution implementation varies by instruction, but each one acceses it's argument using `get_argN()` function of the `Instruction` class. The `program` argument is an instance of the `Prog` class, storing all information about the executed code needed for the instruction to work with variables and program's control flow .

3.6 `Arg`

`Arg`, and it's many sub-classes are a representation of data in IPPcode23. They contain the `data` attribute, which can be the numerical value for integers or text for string.

These objects are the arguments of functions in IPPcode23, and are stored as attributes of their corresponding instructions.

For a visual representation of class inheritance refer to the UML diagram1.

3.7 `Helper`

The `Helper` class contains methods and attributes that are used through the whole program. There are two static methods, `string_escape()` and `error_exit()`, a list of opcodes, and a dictionary of their expected arguments.

`String_escape()` takes a string and converts all corresponding escape sequences to their respective characters. It checks if the escape sequences are valid and returns the result string.

`Error_exit()` is used to print debug information when exiting with an error. It prints the given error and it's given exit code. If the method was called from a running program, which was passed as an argument, it also prints the instruction and position in code.

`Opcodes` are a list of permitted operation codes. `ExpectedArgs` are used to determine if an argument that was given when reading the source code has the correct type.

References

[1] Wikipedia. Prototype pattern — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Prototype%20pattern&oldid=1146922755`, 2023. [Online; accessed 18-April-2023].

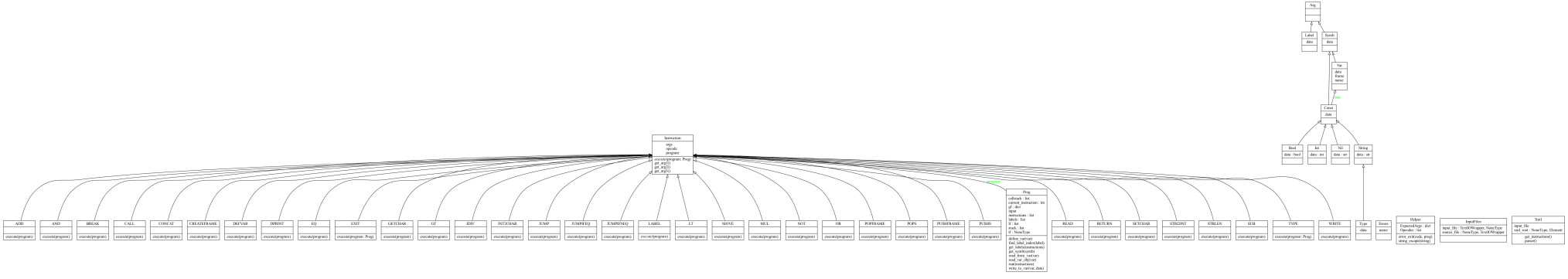

Figure 1: UML diagram