Documentation of interpret.py for IPP 2019/2020

Name and surname: Zhamilya Abikenova

Login: xabike00

1 Interpret.py

The script *interpret.py* was implemented in Python 3.8

main()
In order to get our source XML code, we call function *file_read()* to open a given file (or the standard input if not specified) which is then parsed and tokenized via imported *xml.etree.ElementTree*. This structure is then sorted for better handling of JUMP instructions via *sort_xml()* function. The XML code is then checked for integrity and if it passes, we can start interpreting it. Here we're also checking for all other arguments of the script, such as specified input file important for the function READ, or *--help*, which provides a brief summary of the script.

First cycle

We run through the XML code in this cycle just to collect all LABEL instructions and save their instruction order inside a class *LabelStack*. This ensures jumps in the entire scope of the code are possible right from the start of interpreting.

Second cycle

This cycle is for the actual interpreting of the given XML Code. It parses it line by line, saving all important tokens of the line in the class *line_object*. Correct number of arguments is ensured via the method *arg_code()* after which we can finally start interpreting instructions based on their *opcode* token. Instructions that get executed are checked for correct argument types, their assigned values, and if everything proceeds without fail we move onto the next one.

Frames

Frame functionality is ensured in the class *FramesClass* the only frame available at the start is GF (global frame). If we want to initiate the local and temporary frames, the XML code must cointant instructions such as CREATEFRAME and PUSHFRAME. When the code requests the saving of a variable on a given frame, we check if the frame exists, then if the variable isn't already defined within the frame, and only then can we successfully push it to the frame. This happens in method *defVar*. *setVal* is for assigning a value of a given type to this pre-existing variable. *getVal* is for looking up the variable and returning it's assigned value.

Jumps

Jumps within the program are possible due to the second cycle having an iterator variable *order* which can be adjusted if we meet a CALL or JUMP function. CALL has an additional functionality of adding it's position to the *call_stack* list within *LabelStack*. The RETURN function then uses this list to know to which line of the XML code should it return.