Documentation of Project Implementation for IPP 2022/2023

Name and surname: Michal Ondrejka

Login: xondre15

My design:

I designed a program that interprets XML code containing IPPcode23 assembly instructions. To begin with, I handle command-line arguments and validate the source and input files. Once validated, I load the XML code into an ElementTree and sort the tree so that instructions are in the correct order.

Next, I check each instruction to ensure that it has the right tag and the correct number of attributes. After validating the XML, I create objects from the instructions for easier access. Each Instruction object has a list of Arg objects if the instruction has any.

Since this is an interpreter, it's possible to jump to a label that comes after the instruction currently being executed. To handle this scenario, I loop through the instructions and save the labels, which allows me to call and jump to them when necessary.

To execute the instructions, I make use of a while true loop and a pc counter, which is the index of the current instruction. By incrementing the pc counter after each instruction is executed (except in cases where a jump is called), I can execute the instructions in order and jump to specific instructions by setting the pc counter to a specific number.

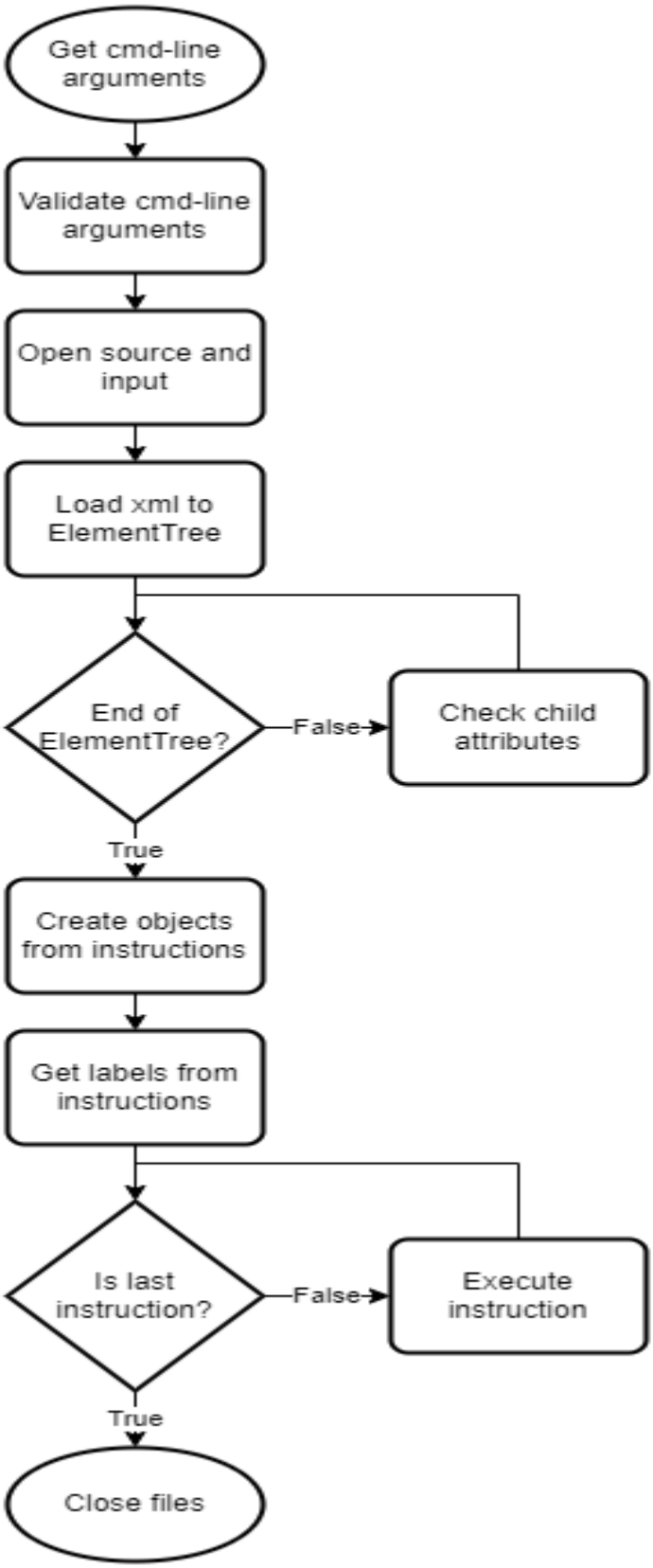
Initially, I didn't need to save the types of variables in my program since Python makes it easy to cast them. However, I had to implement functions for the initial save because IPPcode23 doesn't support bool and null types. Additionally, I created a getType() function for backwards casting.

To handle frames in my program, I created a Frame class with three attributes: parent (containing parent frame), undefined_variables, and variables (defined variables). Whenever a variable is created, it is stored in undefined_variables. Once its value is set, it is moved to variables.

In my Stack class, which can handle frames, there is a temporary_frame attribute and a list of frames. The frame at index 0 is the global frame. Whenever a new frame is pushed onto the stack, the temporary_frame is appended to the list of frames and then set to None.

Known problems:

- I wasn’t able to replace escape sequences with ascii value in strings.
- Accessing undefined variable will result in error 56.
- In READ instruction, reading a faulty input won’t result in “nil” value.


Get cmd-line arguments
Validate cmd-line arguments
Open source and input
Load xml to ElementTree
End of ElementTree?
False
Check child attributes
True
Create objects from instructions
Get labels from instructions
Is last instruction?
False
Execute instruction
True
Close files