Kamila Akhmetova (xakhmek00)

The program can take a file as input if the user provides the file name after `source`. The file must be located in the same folder. Alternatively, the user can enter commands in the console (after `source`, you need to write `-`). To execute the commands, an extra line must be added. If no file name is provided, the default name will be 'name'.

While working on my code, I created classes for custom errors specified in the project document (second table) to catch them and display the appropriate error messages with the line number in ippecode file where the error occured.

I use a list named `operations` to store all allowed operations and define how they should appear in the resulting XML file (in all capital letters). Even if an operation is written in lowercase, the program converts it using the `upper()` function to ensure the correct format in the XML.

Regular expressions are used to validate variables and labels. Specifically, a variable can start with special characters or letters but not with numbers. Labels follow the same rule but must begin with the @ sign.

To store all variables and labels, I use a dictionary. Variables are stored with their names as keys and their values as either strings or numbers (numbers may include a sign). Leading zeros are not allowed, and division by zero is prohibited, triggering an error. Labels are stored with their names as keys and their order (location) as values.

For `call` and `return` operations, I use a list as a stack. Similarly, another list is used as a stack for `push` and `pop` operations.

I have implemented three functions to check variable types (integer, string, or both). For integers, regular expressions are used to check for signs. Additionally, when reading a string from standard input, the program automatically adds quotation marks.

Functions are also used to generate XML tags and handle error situations. To format the XML with proper indentation, I use the `to_indents` function.

The first line of the input is the program's name. Any other line starting with # is treated as a comment (must be on the separate from operation line) and ignored using the `continue` keyword. To keep track of line numbers, I use an `order` counter. It is initialized to `1` because line `0` is reserved for the program name. The counter increments within the loop.

For each operation or group of similar operations, I define a separate function that may also raise errors. Inside these functions, necessary operations are performed, and XML tags are generated. Variables can be reassigned, even changing their type, since they are stored in a dictionary.