Implementační dokumentace k projektu do IPP 2017/2018
Jméno a příjmení: Tomáš Lapšanský
Login: xlapsa00

Our task was to create a part of compiler that takes three address code as an input file and at first stage, parse it into xml file and then takes the xml output and run it in interpret. Our first task was to create file parse.php. I've decided to split implementation into three main part. First was lexical analyzer. It takes characters one by one until it reaches white character and then decide if the given string is keyword or special type of token. We decided to implement token as a class with two public attributes, $keyword and $str. Function getToken from lexical analyzer is creating Token instances and returns it to the main program, which is Syntax analyzer. There are more private methods that lexical analyzer uses to create and determine parameters of created Token. Main part of our parser is syntax analyzer, which is also implemented as an independent class. We decided to use method of recursive descent. So at first, syntax analyzer calls function p_start that is using lexical analyzer that returns tokens, and decides if the given sequence of tokens is correct. For clearness of the code, we created function checkToken, that decides if the given token is correct in dependence of syntax. Syntax analyzer is also creating XML output file in dependence of given input sequence. The last, third part of our implementation of parser is __Global class, that has quitting method and method for printing help log, and also adding elements to our output XML file. Our second task was implementing the interpret in language Python 3.6. Implementation has to parse output XML file and interprets it. We use build in python library ElementTree to work with input XML file and creates function to split it into sequence of dictionaries and lists. Interpret itself is checking the ElementTree structure one by one and calling function from our Core class. It has to contains while cycle with index counter to perform jumps and calls. It validates given opcode from ET and Tag from arguments of interpretations for every statement. We've created special class Core for implementation of every single statement. It also contains special functions for working with datatypes and values of given parameters. It also analyzes the accessibility of variable in Global, Local and Temporary frame. Global frame is implemented as dictionary structure that is initialized at the beginning of program. Local frame is basically "pointer" (we know that Python doesn't implements pointer references) to the top of special stack with Local frames. Local frames are created in runtime by pushing Temporary frames (they are also created in runtime) into this stack. We also had to implement some other stacks for correctness of given input xml interpretation. Our special feature is function label_analysis, that runs through the whole input file at the beginning of program and adds every single index of line with label name to the special dictionary for next implementation calls or jumps. The demonstrations of this function is displayed below.

```
index = 0
while index < len(xml_list):
    if xml_list[index]['Tag'] == 'instruction' and
xml_list[index]['Attrib']['opcode'] == 'LABEL':
        if xml_list[index+1]['Tag'] == 'arg1':
            if xml_list[index+1]['Text'] in self.labels:
                exit_error("Duplicated labels", 52)
            else:
                self.labels[xml_list[index+1]['Text']] = index
    index += 1
```

Third part should be the implementations of script test.php which has to takes test files from selected directory (and in special conditions also subdirectories) and runs it through to parse and interpret. We were testing our code by hand. We didn't finish implementation of test.php in time so we decided to not pass it.