Task documentation XQR: XML Query in Python 3 for IPP 2016/2017
Name and surname: Dávid Bolvanský
Login: xbolva00

1 Initial Decision

At first we decided that we would do both extensions because the precence of extensions has impact on the grammar so if we wanted to do some extension later, mainly the parser would have to be rewritten. Also it would have required more time and more testing. This project can be divided to some parts, which we will describe below:
→ Processing Arguments
→ Lexical Analysis
→ Grammar, Parser
→ Filter Elements
→ Output

2 Processing Arguments

We used the **argparse** to detect options and their values, if any. In this step we check entered options for various errors, e.g. not specified values for some arguments. We also check arguments for duplicated options. If any error, an error message is printed and the script exists with the code 1.

3 Lexical Analysis

As an initial step, we need to identify lexemes which are represented by tokens. We written our own scanner to process the query. The scanner is a simple automaton with a few states. We get the char by char the using created function **fgetc**. The char can be returned back using the function **ungetc**. The query is basically transformed to the list of tokens. We get the token by token by calling the function **get_token**.

4 Grammar, Parser

Since we have got the grammar in the project description, we took this grammar and we constructed the *LL* table. We wrote the query parser which followed the rules in the *LL* table. We added additional semantic checks into the parser too. The scripts exits with the code 80 if any lexical, parser or semantic error.

5 Filter Elements

As we mentioned above, we decided to do both extentions so we our condition parsing is done using precedence analysis. We decided to use the library **minidom** which provides useful functions to find certain elements - e.g. function **getElementsByTagName**. A condition is parsed using functon **filter_elements** where we find certain element which value or attribute is compared with value in the condition. Operators for comparations are: *CONTAINS*, >, < and =. If type of element is not a text node, we exits script with the error code 4.

6 Output

At first we load a *XML* header from input file and write it to output file, if **-n** is not set. Then we add root element, if option –**root-element** is set. The value of this option is used as name of the root element. We print all elements which meet the requirements of the entered query. We add the ending block of the root element, if the root element was requested. The output is printed to the standard output stream, if –**output** is not defined. Otherwise, the output is printed to the specified file.

7 Extensions

Both extensions were implemented in this task.

7.1 FUN (1.0pt)

To implement this extension we used Python functions **sort** to sort the result list of output elements. Also, an additional check was added to analyse if element to be used for sorting, exists. If not, the script exits and an error is printed to the standard error stream. We used **list(reversed(xml_selected_elements))** to reverse list, needed for the descending sort. As the last step, we add an order number to every element using the function **setAttribute**.

7.2 LOG (2.0pt)

To process conditions with more operators, like logical *NOT*, *AND*, *OR* and conditions with brackets, we created the precedence table which follows provided information about operator priorities. Result table you can see below:

	NOT	AND	OR	(	)	Cond	$
NOT	$<$	$>$	$>$	$<$	$>$	$<$	$>$
AND	$<$	$>$	$>$	$<$	$>$	$<$	$>$
OR	$<$	$<$	$>$	$<$	$>$	$<$	$>$
(	$<$	$<$	$>$	$<$	$=$	$<$	
)		$>$	$>$		$>$		$>$
Cond	$>$	$>$	$>$		$>$		$>$
$	$<$	$<$	$<$		$<$	$<$	

We created the class **PrecedenceStack** which encapsulates a stack and methods working with it, especially the functions **reduce_by_rule** and **insert_handle**. In this stack we have the filtered lists of result elements. The top of the stacj contains final list of elements which have required elements, as defined in the condition(s) in the query.