FLCD Scanner Documentation

Alex Ovidiu Popa

936/1

Link to github repository:
https://github.com/alexovidiupopa/flcd/tree/main/scanner

Problem Requirements

Implement a scanner (lexical analyzer): Implement the scanning algorithm, Symbol Table and Program Internal Form.

Input: p1.txt, p2.txt, p3.txt, p1err.txt, Token.in

Output: st.out, pif.out, lexically correct/lexical error + location

Symbol Table

The underlying structure of the Symbol Table is a custom Hash Table, due to the fact that its operations run in O(1). The SymbolTable class will be, for the moment, a wrapper for HashTable.

The HashTable is kept as a Python list with a Deque (double ended linked list) on each position. Having a deque ensures that conflicts are solved, because when two elements hash to the same position, they will both be added to the same deque.

The hash function is computed as follows:

- Sum = 0
- For each char of the string, its ascii code is added to the sum
- Sum % SIZE is returned

The operations implemented by the HashTable are:

1. add (key) - adds the given key into de symbol table

2. remove (key) - removes the given key from the symbol table

3. contains (key) - return a boolean value which indicates whether the given value exists in the symbol table

4. getPosition(key)- returns a pair containing the index of the position in the list and the index of the position in the respective deque of the given key

Program Internal Form

The PIF is kept as a simple list having pairs consisting of (token, position), where position is another pair representing the position of said token in the Symbol Table.

In case of operators, separators and reserved words, the value of position is always (-1,-1), because those are not stored in the ST.

Tokenizing

The tokenizing algorithm goes character by character on each line and checks whether the current we have so far is part of an operator, is a separator, begins a string or is building a constant or identifier, and then appends the tokens to a list which is returned.

Scanning

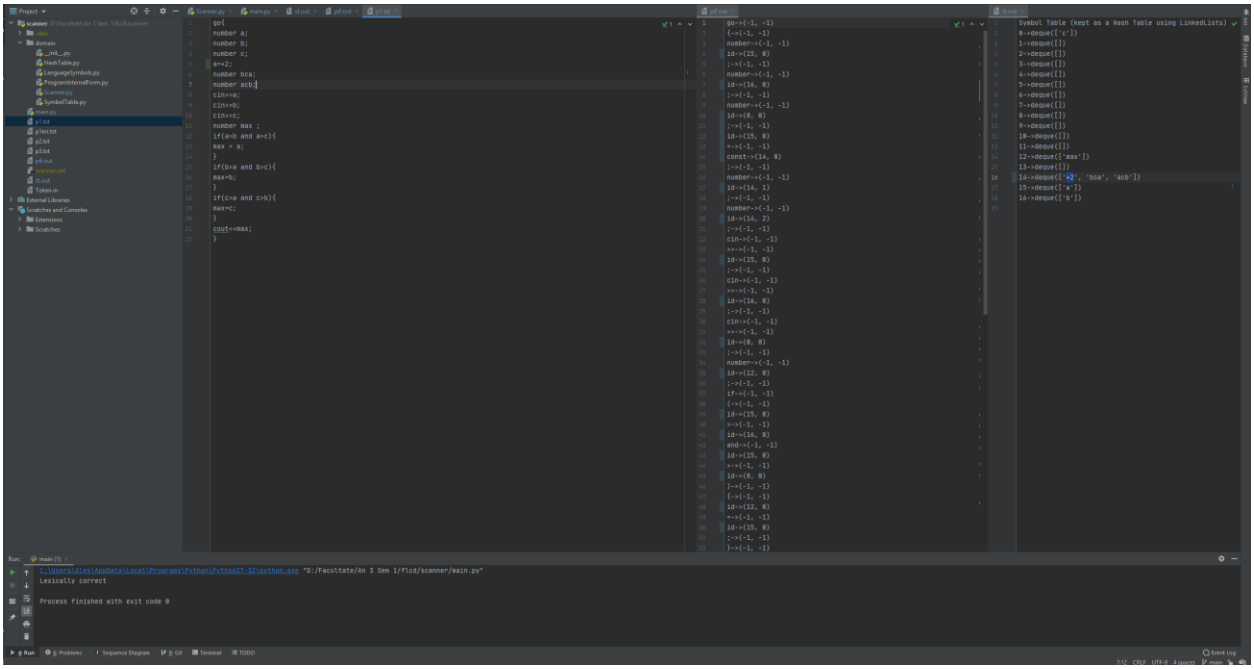
The scanning algorithm splits each line of the program into tokens, and for each token it acts as specified above, I.e. if it's a constant or identifier, look up its position in the ST, if it's an operator/separator/reserved word, its position is (-1,-1).

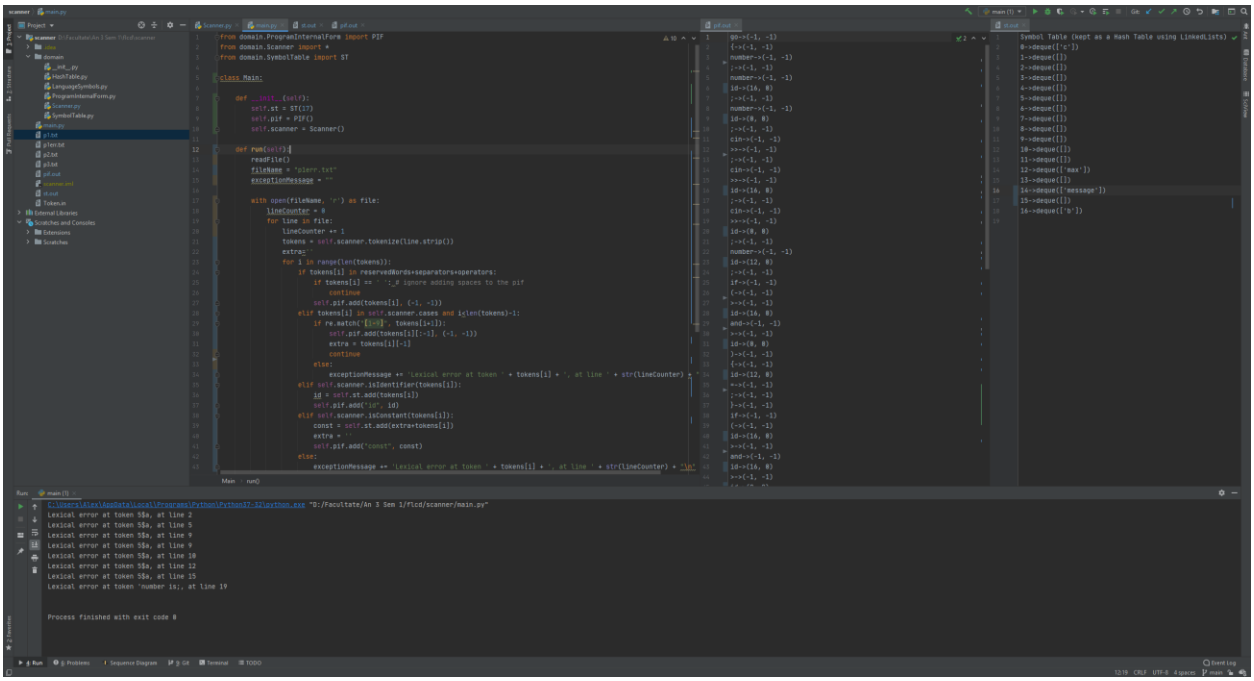
Also, if it's a constant or identifier, instead of keeping the variable name/ constant value, it will be added into the PIF with the code "const" or "id", respectively.

If the token is none of the above, that means we have a lexical error at that line, and the error is appended to the message.

```
with open(fileName, 'r') as file:
    lineCounter = 0
    for line in file:
        lineCounter += 1
        tokens = scanner.tokenize(line.strip())
        extra=''
        for i in range(len(tokens)):
            if tokens[i] in reservedWords+separators+operators:
                if tokens[i] == ' ':# ignore adding spaces to the pif
                    continue
                pif.add(tokens[i], (-1, -1))
            elif tokens[i] in scanner.cases and i<len(tokens)-1:
                if re.match("[1-9]", tokens[i+1]):
                    pif.add(tokens[i][:-1], (-1, -1))
                    extra = tokens[i][-1]
                    continue
                else:
                    exceptionMessage += 'Lexical error at token ' + tokens[i] + ', at line ' + str(lineCounter) + "\n"
            elif scanner.isIdentifier(tokens[i]):
                id = st.add(tokens[i])
                pif.add("id", id)
            elif scanner.isConstant(tokens[i]):
                const = st.add(extra+tokens[i])
                extra = ''
                pif.add("const", const)
            else:
                exceptionMessage += 'Lexical error at token ' + tokens[i] + ', at line ' + str(lineCounter) + "\n"
```

Tests:





Class Diagram