Documentation
Gabor Petru-Andrei 933/2

Statement: Implement a scanner (lexical analyzer) with the scanning algorithm and use ST from lab 2 for the symbol table.

Input: Programs p1/p2/p3/p1err and token.in

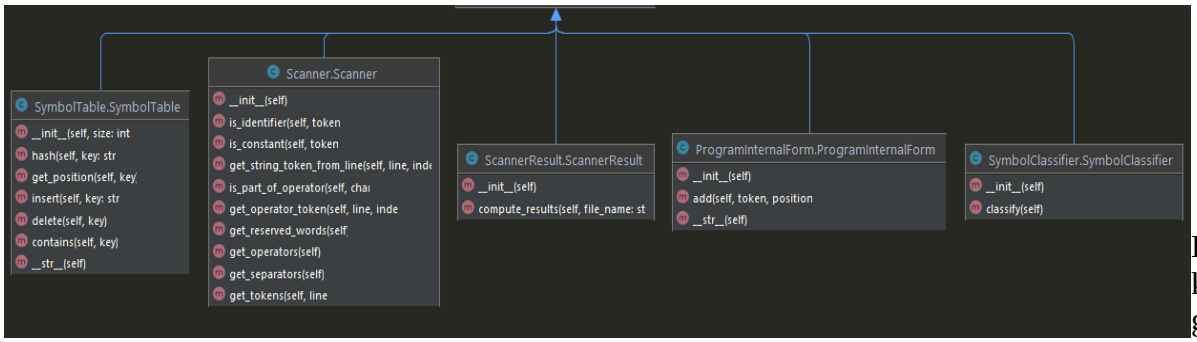
Output: PIF.out, ST.out, message “lexically correct” or “lexical error + location”

Details:

- ST.out should give information about the data structure used in representation
- If there exists an error the program should give a description and the location (line and token)

Solution:

Class Diagram



Link to github: https://github.com/andreigabor21/Formal-Languages-and-Compiler-Design

Symbol Table

I have chosen to represent the Symbol Table using a custom Hash Table.
Technically, the programming language is Python. The hash function is the sum of all the ASCII codes of the given key divided by the length of the table.
Using this representation, the conflicts are resolved such that when two elements having the same hash, they will both be added to the same list.

Operations:

insert(key) – Adds the given key into the symbol table
delete(key) – Deletes the given key
contains(key) – Checks if the key is inside

Program Internal Form

The Program Internal Form is defined as a list of pairs - (token, position), where position is 0 for reserved words, operators and separators, but for identifiers and constants it will be defined like (hash from symbol table, position in the corresponding list).

Scanner

The algorithm splits each line of the program into tokens, and for each token it acts adequate: if it's a constant or identifier, look up for its position in the ST, if it's an operator or separator or reserved word, its position is 0. 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.

Example:

program:

```
fn main() {
  int a;
  int b;
  int c;
  int avg;
  read(a);
  read(b);
  read(c);
  int sum;
  int avg;
  sum = a + b + c;
  avg = sum / 3;

  println(avg);
}
```

st.out :

```
0->['a']
1->['b', 'sum']
2->['c']
3->['3']
4->[]
5->[]
6->['avg']
```

pif.out :

```
fn->0
main->0
(->0
```

)->0
{->0
int->0
id->(0, 0)
;->0
int->0
id->(1, 0)
;->0
int->0
id->(2, 0)
;->0
int->0
id->(6, 0)
;->0
read->0
(->0
id->(0, 0)
)->0
;->0
read->0
(->0
id->(1, 0)
)->0
;->0
read->0
(->0
id->(2, 0)
)->0
;->0
int->0
id->(1, 1)
;->0
int->0
id->(6, 0)
;->0
id->(1, 1)
=->0
id->(0, 0)
+->0
id->(1, 0)
+->0
id->(2, 0)
;->0
id->(6, 0)
=->0
id->(1, 1)
/->0
const->(3, 0)
;->0
println->0
(->0
id->(6, 0)
)->0

;->0
}->0