Laborator 1 LFTC

Stefan Sebastian 235

Descrierea cerintei

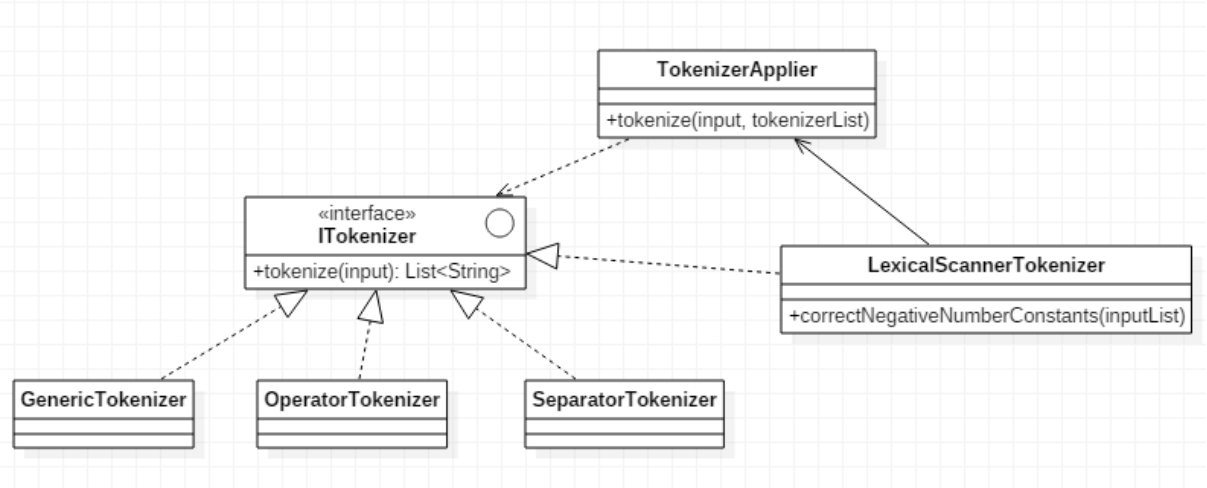
Sa se defineasca un limbaj de programare care contine urmatoarele: 2 tipuri de date simple si un tip definit de utilizator, instructiuni de atribuire, intrare/iesire, conditionala, ciclare. Sa se implementeze un analizor lexical care citeste un program, in limbajul de programare definit, dintr-un fisier si produce Forma Interna a Programului si Tabela de Simboluri.

Restrictii

Identificatorii au o lungime maxima de 250 de caractere. Tabela de simboluri este unica pentru identificatori si constante. Tabela de simboluri este organizata ca tabel ordonat lexicografic. Analizorul identifica erorile lexicale si liniile pe care acestea apar.

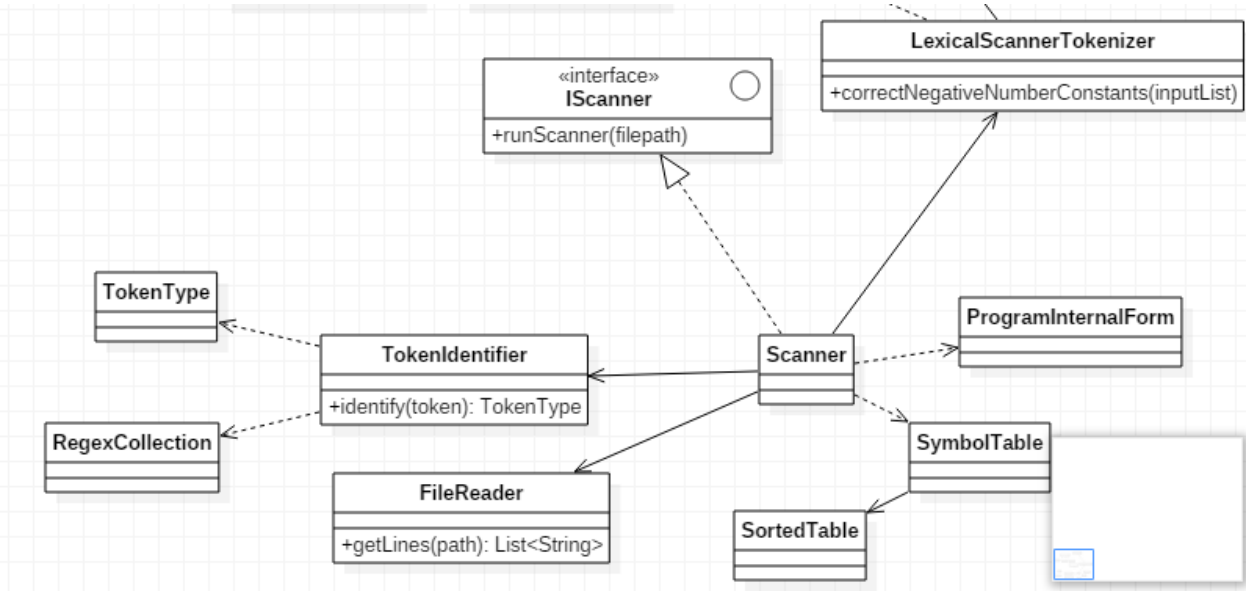
Proiectare

Modulul de ‘tokenizare’ expune interfata ITokenizer. Mai multe clase concrete tokenizeaza inputul dupa anumiti parametri. TokenizerApplier inlantuieste mai multi ITokenizer pe acelasi input. LexicalScannerTokenizer foloseste TokenizerApplier pentru a inlantui tokenizare dupa separatori si operatori si defineste o metoda de diferentiere intre constante negative si pozitive cu operatorul “-“.



Functionalitatea aplicatiei este expusa de interfata IScanner care primeste calea unui fisier si returneaza o structura formata din Forma Interna a Programului si Tabela de Simboluri.

Scanner este clasa concreta care implementeaza interfata astfel: folosind metoda getLines din clasa FileReader obtine liniile din fisierul dat, pe fiecare linie aplica metoda de tokenizare din modulul prezentat anterior, fiecare token este identificat de catre TokenIdentifier care foloseste o colectie de expresii regulare (definite in RegexCollection), in functie de tip tokenul este salvat in ProgramInternalForm si daca e constanta sau identificator in SymbolTable care contine un SortedTable.



Detalii de implementare

Pentru functia de tokenizare am folosit un regex de forma "((?<=%1$s)|(?=%1$s))" unde %1$s va fi inlocuit cu delimitatori concreti. Aceasta expresie captureaza si delimitatorii. Am facut split dupa separatori dar si dupa operatori pentru a permite expresii de forma a=b+c (fara separatori intre elemente). Pentru a rezolva problemele de look-ahead generate de operatorii formati din mai multe caractere (>=, <=, ==, !=) am facut prima data split dupa acesti operatori apoi am aplicat functia de split cu operatorii: <, >, =, .. pe rezultatele care nu contineau operatorii initiali.

Pentru a clasifica tokenii am folosit urmatoarele expresii regulare:

- Cuvinte rezervate : int|char|read|write|if|else|while|begin|end
- Operatori : \+|-|*|/|%|<|>|=|<=|>=|==|!=
- Separatori : |;|\(|\)|\{|\}|\[|\]
- Identificatori : (_|[a-zA-Z])[a-zA-Z0-9_]*
- Lungimea identificatorilor : ^.{1,250}$

- Constante de tip int : [-]?[1-9]+[0-9]*|0
- Constante de tip char : '[0-9a-zA-Z]'
- Constante de tip string : "[_0-9a-zA-Z]+"

Daca un token nu se potriveste cu niciuna din expresiile enumerate mai sus atunci se arunca o exceptie care marcheaza o eroare lexicala. Deoarece am parcurs fisierul linie cu linie se poate identifica si linia pe care a aparut exceptia.

Tabela de simboluri e organizata dupa un tabel ordonat lexicografic. Inserarea se face in O(n) si cautarea in O(logn) prin cautare binara. Tabela de simboluri contine perechi de forma : simbol, identificator; unde simbolul poate fi identificator sau constanta. Forma Interna a Programului este o lista de perechi : un cod din tabela de codificare si pozitia in tabela de simboluri sau -1 (daca nu e identificator sau constanta).

Daca un token este clasificat ca identificator sau constanta este cautat in tabela de simboluri. Daca e gasit se salveaza in FIP codul corespunzator (0 – identificator, 1 – constanta) si pozitia din tabela de simboluri, in caz contrar este inserat in tabela de simboluri si apoi in FIP. Tabela de simboluri contine un atribut Identifier care se incrementeaza la fiecare inserare si care este folosit pentru a identifica simbolurile inserate. Pentru operatori, separatori si cuvinte rezervate se salveaza in FIP codul si valoarea -1.

Tabela de codificare are urmatoarea forma:
identifier 0, constant 1, begin 2, end 3, int 4, char 5, read 6, write 7, if 8, else 9, while 10
{ 11, } 12, (13,) 14, ; 15, space 16, + 17, - 18, * 19, / 20, % 21, == 22, = 23, != 24, < 25
<= 26, > 27, >= 28, [29,] 30

Exemple de rulare

Obs. In PIF este afisat si tokenul de pe fiecare pozitie. Spatiul este codificat cu 16.

begin
int a;
a = 3;
write(a);
end
	PIF
	2 -1 begin
	4 -1 int
	16 -1
	0 1 a
	15 -1 ;
	0 1 a
	16 -1
	23 -1 =
	16 -1
	1 2 3
		15 -1 ;
		7 -1 write
		13 -1 (
		0 1 a
		14 -1)
		15 -1 ;
		3 -1 end
		
		TS
		2 3
		1 a

```
begin
int a;
read(a);
if(-3<a){
   write(a);
}else{
   write("un_string");
}
end
```

```
PIF
2 -1 begin
4 -1 int
16 -1
0 1 a
15 -1 ;
6 -1 read
13 -1 (
0 1 a
14 -1 )
15 -1 ;
8 -1 if
13 -1 (
1 2 -3
25 -1 <
0 1 a
14 -1 )
11 -1 {
16 -1
16 -1
16 -1
16 -1
7 -1 write
13 -1 (
0 1 a
14 -1 )
15 -1 ;
12 -1 }
9 -1 else
11 -1 {
16 -1
16 -1
16 -1
16 -1
7 -1 write
13 -1 (
1 3 "un_string"
14 -1 )
15 -1 ;
12 -1 }
3 -1 end

TS
3 "un_string"
2 -3
1 a
```

```
begin
int a[10];
int i;
i = 0;
while(i<10){
   read(a[i]);
   i=i+1;
}
end
```

```
FIP
2 -1 begin
4 -1 int
16 -1
0 1 a
29 -1 [
1 2 10
30 -1 ]
15 -1 ;
4 -1 int
16 -1
0 3 i
15 -1 ;
0 3 i
16 -1
23 -1 =
16 -1
1 4 0
15 -1 ;
10 -1 while
13 -1 (
0 3 i
25 -1 <
1 2 10
14 -1 )
11 -1 {
16 -1
16 -1
16 -1
16 -1
6 -1 read
13 -1 (
0 1 a
29 -1 [
0 3 i
30 -1 ]
14 -1 )
15 -1 ;
16 -1
16 -1
16 -1
16 -1
0 3 i
23 -1 =
0 3 i
17 -1 +
1 5 1
15 -1 ;
12 -1 }
3 -1 end

TS
4 0
5 1
2 10
1 a
3 i
```

begin
char c;
c='b';
write(c);
char 12abc;
end
	Invalid token : 12abc on line 5

begin
char c;
c='b';
int a;
a=((2+3)-2)*-2;
end
	FIP
	2 -1 begin
	5 -1 char
	16 -1
	0 1 c
	15 -1 ;
	0 1 c
	23 -1 =
	1 2 'b'
	15 -1 ;
	4 -1 int
	16 -1
	0 3 a
	15 -1 ;
	0 3 a
	23 -1 =
	13 -1 (
	13 -1 (
	1 4 2
		17 -1 +
		1 5 3
		14 -1)
		18 -1 -
		1 4 2
		14 -1)
		19 -1 *
		1 6 -2
		15 -1 ;
		3 -1 end
		
		TS
		2 'b'
		6 -2
		4 2
		5 3
		3 a
		1 c