SAKHI PATEL wz7995

PARSER

```
package Parse;
/*
* Recursive descent parser that checks for parse tree or leftmost     derivation for the input program in
  baby Algol and is implemented.

It takes baby Algol program as input and checks if the input file has parse tree or derivation. It has
routines for all the  * non terminals.
        *
* 1.program              :  blockst '.'
        1

        2.blockst              :  BEGINTOK stats ENDTOK                                          2


        3.stats                :   statmt ';'  stats   | <empty>                                 3,4


         5. decl               :  BASICTYPETOK IDTOK


        5.statmt          : decl | ifstat | assstat |  blockst | loopst | iostat | <empty>6,7,8,9,10,11,12
        6.assstat           :  idref   ASTOK  expression                                       13


        7.ifstat            :  IFTOK  expression THENTOK statmt                          14


        8.loopst            :  WHILETOK expression DOTOK statmt                               15


        9.iostat           :  READTOK ( idref ) | WRITETOK (idref)                            16


        10.expression        :  term expprime                            E -> T E'
                                                                                      17
        11.expprime          :  ADDOPTOK  term expprime   |  <empty>      E' -> ADD T E' | eps
                                                                        18,19
        12.term              :  relfactor termprime                     T -> RF T'
                                                                20
        13.termprime         :  MULOPTOK  relfactor termprime | <empty>  T'-> MUL RF T' | eps
                                                            21,22
        14.relfactor         :  factor factorprime                      RF -> F F'
                                                    23
         15.factorprime       :  RELOPTOK  factor          |  <empty>       F' -> REL F   | eps  24,25

        16.factor            :  NOTTOK   factor        F -> NOT ID | ID | LIT |
                                                                                  26,27,28,29
                          |  idref
                          |  LITTOK
                          |  '('  expression  ')'         ( E )
        17. idref            : IDTOK                                                           30



        *
        *
```

```
	*	*/



import java.io.IOException;
import java.text.ParseException;
import java.util.ArrayList;
import Parse.Scanner;
import Parse.Token;


public class Parse
	{

	 static Scanner lexer; //object of the Scanner Class used to access Scanner methods
	static String inFile = "C:\\Users\\sakhi\\workspace\\Lex\\src\\Test";  //Input File

	protected static Object[] gettoken; //array that stores all the tokens from the scanner
	protected static Object token;
	protected static int i=0;


	public static void blockst(Object tok) throws IOException, ParseException
	{
	 //RULE [1] program start
		System.out.println("1");
		S(tok);
		match(token,18);				//(Token no : 18 for DOT(.))

	}



	public static void S(Object tok) throws IOException, ParseException
	{
	//RULE [1] program start
		System.out.println("1");
		if(tok.equals(7))				//(Token no : 7 for BEGIN)
		{	// RULE [2] BEGINTOK stats ENDTOK
			System.out.println("2");
		match(tok,7);
			Stats(token);
			match(token,8);				//(Token no : 8 for END)
		}


		else
			System.exit(0);

	}
```

```
// RULE [3] || [4] -> stats :   statmt ';'  stats   | <empty>
/*token no.   1 -> identifier, token no. 3 -> Keyword, token no. 7 -> BEGIN, token no. 9 -> IF,
token no. 11 -> WHILE, token no. 13 -> READ, token no. 16 -> ';' */

        private static void Stats(Object tok) throws IOException, ParseException
        {
        if(tok.equals(1) || tok.equals(3) || tok.equals(7) || tok.equals(9) || tok.equals(11) ||
        tok.equals(13) || tok.equals(16))
                {
                System.out.println("3");
                Statmt(tok);
                match(token,16);
                Stats(token);

  else          }


{

                        System.out.println("4");
                        System.out.println("Success");
                        System.exit(0);
 }

 }



//check if token is a statement
// RULE[6,7,8,9,10,11,12] statmt: decl | ifstat | assstat |  blockst | loopst | iostat | <empty>
public static void Statmt(Object token) throws IOException //check if token is a statement, ParseException
, ParseException
        {
                if(token.equals(1) )                    //token 1 -> identifier
                {
                 System.out.println("8");
                asst(token);

                }
                else if(token.equals(3))                        //token 3 -> keywords
                {
                System.out.println("6");
                decl(token);
                }
                else if(token.equals(7))                        //token 7 -> BEGIN
                {
                System.out.println("9");
                S(token);
                }
                else if(token.equals(9))                          //token 9 -> IF
                {
                System.out.println("7");
                If(token);
                }
                else if(token.equals(11))                       //token 11 -> WHILE
                {
                System.out.println("10");
                loop(token);

                }
                else if(token.equals(13))                      //token 13 -> READ WRITE WRITELN
```

```
                {
                System.out.println("11");
                IO(token);
                }
                  else
                {
                        System.out.println("12");

                }

        }



         //RULE [5] decl:  BASICTYPETOK IDTOK
        public static void decl(Object tok) throws IOException, ParseException
        {
                System.out.println("5");
                match(tok,3);                   //token 3 -> keywords
                match(token,1);                         //token 1 -> identifier

        }



//RULE [13] assstat ->  idref   ASTOK  expression
        public static void asst(Object tok) throws IOException, ParseException
        {
                System.out.println("13");
                idref(tok);                             //call  idref
                match(token,19);                        // := (ASTOK) token
                Exp(token);

        }




//RULE [14] IFTOK  expression THENTOK condition
 public static void If(Object tok) throws IOException //If Condition , ParseException
, ParseException
        {


                System.out.println("14");
                match(tok,9);                                   //token for IF

                Exp(token);
                match(token,10);                                //token for THEN
                Statmt(token);

         }
```

```
//RULE [15] WHILETOK expression DOTOK statmt
        private static void loop(Object tok) throws IOException //While Loop , ParseException
, ParseException
{
        System.out.println("15");
        match(tok,11);                          //token num for WHILE
        Exp(token);
        match(token,12);                         //token num for DO
        Statmt(token);


}




//RULE[16] || [17] READTOK ( idref ) | WRITETOK (Exp) public
static void IO(Object tok) throws IOException //IO
, ParseException
        {
                        if (tok.equals(13))
                          {
                         System.out.print("16 ");
                          match(tok, 13);                //token for Read
                        match(token, 14);       //token for '('
                             idref(token);
                             match(token, 15);                  //token num for ')'
                          }

                        else if
                        (tok.equals(20)
                        )
                            {
                                 System.out.print("17 ");
                                 match(tok, 20);          //token num for WRITE
                                 match(token, 14);                //token num for (
                                 Exp(token);
                                match(token, 15);                //token num for )
                           }
                        else
                        System.exit(0);



        }


 // RULE [18] E -> T E_
public static void Exp(Object tok) throws IOException  //Expression
, ParseException
        {
                System.out.println("18");
                T(tok);
                E_(token);

        }

//RULE [19]||[20] E_ -> ADD T E_ | eps
public static void E_(Object tok) throws IOException //E Prime
, ParseException
        {
                if(tok.equals(4))
```

```
                {
                System.out.println("19");
                match(tok,4);                           //token for '+'
                T(token);
                E_(token);

                }

                else
                {
                        System.out.println("20");

        }

}



//RULE [21] T -> RF T_
public static void T(Object tok) throws IOException //Term
, ParseException
        {
                System.out.println("21");
                RF(tok);
                T_(token);

        }


//RULE [22] || [23] T-> MUL RF T_ | eps

private static void T_(Object tok) throws IOException //T Prime
, ParseException
        {

                if(tok.equals(5))
                {
                        System.out.println("22");
                        match(tok,5);   //token for *
                        RF(token);
                        T_(token);

                }

                else

                {

                        System.out.println("23");

}

}


//RULE [24] RF -> F F_

private static void RF(Object tok) throws IOException, ParseException
  {
                System.out.println("24");
                F(tok);
                F_(token);

}
```

```
//RULE [25] || [26] F_ -> REL F  | eps
private static void F_(Object tok) throws IOException //F Prime
, ParseException
        {
                if(tok.equals(6))
                        {
                        System.out.println("25");
                        match(tok,6);           //token num 6 for relational operators
                        F(token);
                        }
                else
                {
                        System.out.println("26");

                }

        }


//RULE [27||28||29||30] F ->NOT ID | LIT | ID | (E)
private static void F(Object tok) throws IOException             //Factor
, ParseException
        {
                if(tok.equals(17)) //NOT ID
                {
                        System.out.println("27");
                        match(tok,17);
                        idref(token);

                }
                else if(tok.equals(2))//LITERAL
                {
                        System.out.println("29");
                         match(tok,2);

                }
                else if(tok.equals(1))//ID
                {
                        System.out.println("28");
                        idref(tok);

                }
                else if(tok.equals(14))//(F)
                {
                        System.out.println("30");
                         match(tok,14);
                         Exp(token);
                        match(token,15);

                }

        }


//RULE [31] idref -> IDTOK       public static void idref(Object tok) throws
IOException, ParseException
        {
        if(tok.equals(1))

                {
                        System.out.println("31");
                             match(token,1);
```

```
                }

}

 //MATCH public static void match(Object tok,
int exp)
{

                if(tok.equals(exp))
                {               i++;
                                token = gettoken[i];

                }                               //returns next character.
                else
                {
                System.out.println("Error");
                System.exit(0);
                }

        }

        public static void main(String[] args) throws IOException, ParseException
        {


                 // Buffer of lines from input file
                ArrayList<Integer> list = new ArrayList<Integer>();
                lexer = new Scanner(inFile);//call to line 1
                Token t;

                 while ((t = lexer.nextToken()) != null)
                 {

                        int k = t.tok();
                        //System.out.println(k);
                        list.add(k);
                }
                 gettoken = list.toArray();

                  token = gettoken[i];
                     blockst(token);
                        // token no. -1 -> end of file
                   System.exit(0);
        }

        }
```

Token class

```
package Parse; import
        java.text.ParseException;

        public class Token
        {
                private int token;

                public Token(int token)
                {
                        this.token = token;

                }

                //@Override     public
int tok() {
```

```
                    return token;
                }


            }

        }
```

Input file:

```
WHEN A> B PRINT A
```

OUTPUT:

```
fail
Input:
```

InputFile1:

```
BEGIN
INTEGER X ;
X := (2* 3+6);
READ ( X ) ;
END
```

Output:

```
1
2
3
6
5
3
8
13
31
18
21
24
30
18
21
24
29
26
22
24
29
26
23
19
21
24
29
26
23
20
26
23
20
3
11
```

```
16 31
4
Success
```

Input:

```
BEGIN
LOGICAL A;
INTEGER c ;
A := 5 ;
c := TRUE ;
WHILE ( ! A
)
DO
c := TRUE  ;
IF (A> 0 )
THEN A := 6
;
END.
```

Output:

```
1
2
3
6
5
3
6
5
3
8
13
31
18
21
24
29
26
23
20
3
8
13
31
18
21
24
29
26
23
20
3
10
15
18
21
24
30
18
21
24
27
28
```

```
31
26
23
20
26
23
20
3
8
13
31
18
21
24
29
26
23
20
3
7
14
18
21
24
30
18
21
24
28
31
25
29
23
20
26
23
20
8
13
31
18
21
24
29
26
23
20
4
Success
```

Input:while nested in if

```
BEGIN
INTEGER X;
x := 0;
INTEGER Y;
IF( X != y)
THEN
WHILE(!X)
DO
X := y;
END.
```

OUTPUT:

```
1
2
3
6
5
3
8
13
31
18
21
24
29
26
23
20
3
6
5
3
7
14
18
21
24
30
18
21
24
28
31
25
28
31
23
20
26
23
20
10
15
18
21
24
30
18
21
24
27
28
31
26
23
20
26
23
20
3
8
13
31
18
21
24
28
31
26
```

```
23
20
4
Success
```

Input: if nested in while

```
BEGIN
LOGICAL y;
INTEGER A;
WHILE(!A)
DO
IF(!A)
THEN
x := x + 2;
END.
```

OUTPUT:

```
1
2
3
6
5
3
6
5
3
10
15
18
21
24
30
18
21
24
27
28
31
26
23
20
26
23
20
3
7
14
18
21
24
30
18
21
24
27
28
31
26
23
20
26
23
20
```

```
8
13
31
18
21
24
28
31
26
23
19
21
24
29
26
23
20
4
Success
```