/\*Recursive descent parser for expressions

To simplify the programming tokens will be single characters

consisting of digits, letters, \*, + ,(, ) , =, $

Input strings will look like :

"x + a \* ( 9 + b)$"

White space is ignored.

S -> E$

E -> T + E | T

T -> F \* T | F

F -> (E) | letters | digits

\*/

package csun.comp333.project2;

import java.util.\*;

public class RecDescentParser

{

public char nextchar;

public String inputString;

private int pos;

public RecDescentParser ( String s)

{

inputString = s;

pos= 0;

System.out.println("\nParsing: " + s);

}

public void start()throws Exception

{

try {

getChar();

S();

System.out.println("Successful parse");

}

catch (Exception e)

{

System.out.println("Unsuccessful parse");

}

}

public void getChar()throws Exception

{

//skip over blank chars

while ( pos < inputString.length()

&& inputString.charAt(pos) == ' ')

{

pos++;

}

if( pos < inputString.length() )

{

nextchar = inputString.charAt(pos);

System.out.println("getChar: " + pos + " " + nextchar);

pos++;

}

else

error();

}

public void S()throws Exception

{

STList();

match('$');

}

public void STList()throws Exception

{

ST();

if(nextchar == ';')

{

match(';');

STList();

}

}

public void ST()throws Exception

{

N();

if( nextchar == '=')

{

match('=');

E();

}

else

error();

}

public void E()throws Exception

{

T();

if( nextchar == '+')

{

match('+');

E();

}

else if ( nextchar == '-')

{

match('-');

E();

}

}

public void T()throws Exception

{

F();

if( nextchar == '\*')

{

match('\*');

T();

}

else if( nextchar == '/')

{

match('/');

T();

}

}

public void F()throws Exception

{

if ( Character.isLetter(nextchar))

{

match(nextchar);

return;

}

else if ( Character.isDigit(nextchar)){

match(nextchar);

I();

}

else if (nextchar == '(')

{

match('(');

E();

match(')');

}

else

error();

}

public void I() throws Exception

{

if(Character.isDigit(nextchar))

{

match(nextchar);

I();

}

}

public void N() throws Exception

{

if(Character.isLetter(nextchar))

{

match(nextchar);

return;

}

}

public void error() throws Exception

{

System.out.println( "Syntax error at position : "

+ pos + " with character: " + nextchar);

throw new Exception ("Syntax Error");

}

public void match( char u) throws Exception

{

if(nextchar == u)

{

if ( u != '$')

getChar();

return ;

}

else

error();

}

public static void main( String[] args)throws Exception

{

RecDescentParser rdp0 =

new RecDescentParser("x = y \* (a + 2 - c) + A $" );

rdp0.start();

RecDescentParser rdp1 =

new RecDescentParser("z = a + (b + 1)) $" );

rdp1.start();

RecDescentParser rdp2 =

new RecDescentParser("u = 7 ; v = a + 8 ; x = 3$" );

rdp2.start();

RecDescentParser rdp3 =

new RecDescentParser("c = z + 341 + 5 ; x = 100$" );

rdp3.start();

}

}